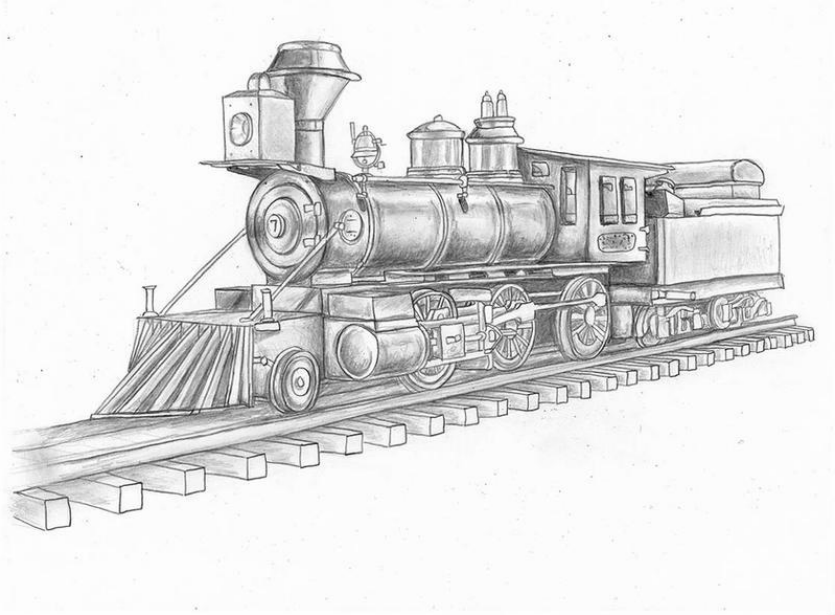
IoT Hugs The Rail



Arushi Garg, Lasya Josyula, Sanjana Madhu, Troy Tomasch

S.A.L.T. (Team 6)

CS 347

Spring 2021

Table of Contents

1. Introduction:

1.1 Project Overview:

1.1.1 Problem Statement

Trains are one of the most integral means of transportation. In this Internet of Things (IoT) *Hugs the Rail Project,* our team will work to create a system that communicates with an IoT engine to receive commands when the train does not have access to WiFi/Cellular Data. This system will also regulate the functionalities of the train depending on external surroundings and data.

1.1.2 Purpose of the Product (IoT Hugs The Rail)

The purpose of our product will be to create a backup system for the train when there is no internet connection. The system and protocols we use will build awareness of the external factors around the train and manipulate the mechanism for ideal train functionality. We will also analyze how to manipulate the functionality of the train based on hazardous weather conditions and other moving objects. Our purpose revolves around using IoT and sensors to analyze external factors and give the operator appropriate commands. By doing so, this product will ensure proper, efficient, and safe train transportation for passengers.

1.1.3 Importance and Value to Operation:

The product is important because trains are a common form of transportation. Even when communication through the internet is not possible, we need to ensure that the train will safely arrive at its destination.

1.2 Project Operation and Approach:

1.2.1 Evolving the Current Operation:

In order to successfully implement this project, we will need to examine the existing operations of the Hugs the Rail train. To improve railroad capabilities, this project will heavily rely on the Internet of Things to communicate and run all functionalities. Under normal circumstances, trains have the ability to communicate utilizing cellular data, local networks, and WiFi; the trains normally communicate with the Fog Servers and the Cloud. However, under severe weather conditions and outages, trains will require a new mode of communication between sensors, other trains, satellites, and other types of external data. This project will work to improve the safety and security of trains under these special

circumstances. This will allow a seamless transition between the train having access to WiFi/Cellular and the train shifting to utilizing the IoT engine.

1.2.2 Approach to the Problem:

We plan to use a Unified Process Model to develop this product as a team. This model is effective because it contains heavy communication with customers. Also it has an efficient iterative system. To complete the project, we will require access to IoT APIs. This will allow us to use the basic functionalities of IoT and implement them in the project. Additionally, we will require recent train data including but not limited to: speed during different weather conditions, breaking acceleration, speeding acceleration. Additionally, we will require data from satellites that store these train specifications as well as means of communicating with the satellite. Using this data, the sensors can better advise trains on their next course of action when there is no internet access.

The specific approach to the problem will include IoT edge devices that capture data from other locomotives and the environment. Using the train's analytical engine, data will be collected, processed, and passed to the Locomotive Control System (LCS). The analytical engine will already have the latest rules of operation from the Cloud and Fog Servers. The core of the project includes allowing train operators the ability to execute commands for the train based on the information received. This in turn will allow *Hug the Rails* to supply safer, less costly, and more efficient operations.

1.3 Team Description:

1.3.1 Our Talented Team:

For any project to be successful, it requires a multidisciplinary team. All four members of the team are highly qualified students, with experience in multiple programming languages including Python, Java, and C++. All successful projects require immense collaboration and communication, which are integral for superb teamwork.

Arushi Garg is a detail-oriented member, an excellent communicator, and values analysis. Analysis skills are extremely important for teams to be able to see what must be improved upon. Troy Tomasch has a knack for problem-solving, is another member who deeply values analysis, and understands the importance of project management. Much of the project management process requires concise and well-written documentation. Sanjana Madhu is a people-oriented member and articulate writer, who also excels in coding in the languages listed above. When working on any project, it is very easy to sway away from the task at hand.

Lasya Josyula is a task-oriented member who also values the importance of project management. Our team is filled with a diverse group of people, who will add a lot of value to this project through their experiences and technical expertise.

2. Overview:

2.1. Project Background:

2.1.1 Background:

Trains depend on WiFi/Cellular networks to receive live data about their environments and surrounding traffic. When trains lose internet connectivity, they depend on local information to operate safely. The local information provided to trains must contain information about hazardous conditions (ice, snow, hail, high wind speeds, obstructive objects, etc.), especially in the case that these conditions are not detected by the train operator. In order to combat this issue, an array of sensors will be installed that can communicate with the IoT engine and alert the operator with pertinent information.

2.1.2 Timeline:

The timeline of this project will be about eight weeks.
02/09/2021 - 02/21/2021: Client communication and Planning
02/22/2021 - 02/28/2021: Project Planning and Setup
03/01/2021 - 03/31/2021: Project Implementation
04/01/2021: **Deployment**
04/02/2021 - 04/11/2021: Test and Debug
04/12/2021 - 05/04/2021: Fix Problems and Re-test
05/05/2021 - 05/09/2021: Finish Project

The first few weeks will be dedicated to intensive planning and understanding of the problem. The remainder of the timeline will be dedicated to programming, building, and testing our software. By 04/01/2021, the team plans for the product to be deployed. The team will also communicate with *Hug the Rails* and shareholders of the company to give updates on our progress and show prototypes.

2.2. Use of Sensors:

2.2.1. Distance Sensors

By using distance sensors, the train will be able to detect its surroundings. If there appears to be an obstacle in its tracks (either behind or in front of the train), the distance sensor will be able to alert the operator of this issue. It will also be able to gauge how far away the object is away from the train.

2.2.2. Hazardous Weather Conditions

This project also aims to protect trains from drastic weather conditions, specifically ice, snow, and the like. For this, we will utilize sensors on the wheels of the train to track the current rotations per minute (rpm) of the wheel. If the rpm exceeds a given threshold (the regular rpm of the train), this means that it is likely the train is "slipping." The sensors will then communicate to the LCS to slow down.

Additionally, trains will be equipped with temperature and humidity/precipitation sensors. If the temperature goes below freezing, the temperature sensor will communicate to the LCS to slow down by default and add sand to the tracks underneath as there may be ice on the tracks. If the humidity/precipitation are triggered, the train will slow down to prevent further slipping.

2.2.3. Surrounding Trains and Environment

The project will use sensors in order to decide whether there are additional trains in its path that could be stopped or moving. Utilizing a forward facing distance sensor, the train will be able to determine how far away another train is in front of it. Then using information such as how fast the other train is approaching and how fast the train is currently moving, the project will be able to calculate whether the train in front of it is moving or stopped. Then using this additional information, it will make a suggestion for the operator. For instance, a stopped train ahead will require the train to stop whereas a train in front of it moving in the same direction might only require the train to slow down as to not collide with the other train.

2.2.4. Recommended Action

This project should cover specific actions based on any situation to the operator. For example, if the distance sensors are activated, the operator should be aware of what is on the track. If there is another train, the operator should be immediately notified so they can take an appropriate course of action. If the weather conditions cause the sensors to be activated, they will indicate that the operator should slow down, so that the train does not slip. Lastly, if there are any other moving objects that trigger the distance sensor, the operator should immediately assess what it is that is on the tracks, so that they may take the right steps.

3. Requirements

3.1. Non-Functional Requirements

3.1.1. Hardware

The hardware for the sensors in this project will be waterproof. Their durability should enable them to last for 1000 hours underwater and operate 99.99% of the time (this should be approved by the company we are purchasing sensors from). Since the sensors are waterproof, they will be allowed to sustain heavy rainfall and other types of extreme weather.

Additionally, the IoT HTR shall be supported by a hardwired LoRaWan network. This will also help the IoT communicate with the train operator. The LoRaWan protocol is a media access control protocol that is generally used for wider networks because it uses a lower radiofrequency. It mainly uses low-powered devices, and since the sensors we plan to use are low-powered, this will work effectively. Additionally, since LoRaWan uses an unlicensed radio spectrum, it is inexpensive to use. Finally, LoRaWan is used over a long-range, can last years on a low-powered battery, is inexpensive, has low bandwidth, and is secure with 128 bit end-to-end encryption.

To implement this feature, we need the following requirements:

- R1: The sensors purchased must be waterproof.
- R2: The sensors must be able to last 1000 hours and operate 99.99% of the time to be approved.
- R3: The LoRaWan protocol is the network that will support this project.

3.1.2. Performance

An extremely crucial aspect of this project is timing and execution. The speed of transmission of messages is vital for the proper functionality of the train and ability for the conductor to take necessary precautions. Without accurate and precise functionality of the sensors, the safety of the train will be put at risk. For this reason, the train must process an event within 0.5 seconds of its occurrence and the sensors of the train must process events within 0.1 seconds.

To implement this feature, we need the following requirements:

- R4: The sensor will be able to process an event on time and give a note to the conductor within 0.1 seconds.

- R5: The train and IoT engine will process the event within 0.5 seconds of the occurrence to enable proper speed of transmission of a message from the distance sensor to the train.

3.1.3. Reliability

Reliability is an extremely important factor in being able to confidently produce hardware. With high reliability, there is a high probability that the hardware at hand will be failure-free for a specific time and in specific conditions. Therefore, it is imperative that the reliability of the IoT HTR be no less than 99.99%, so that the hardware at hand can be used confidently.

To implement this feature, we need the following requirements:

- R6: The hardware must be failure free for 1000 hours, and have a high probability of lasting this long.
- R7: The reliability of hardware should be no less than 99.99%.

3.1.4. Security

The system will need to maintain tight security to ensure that the train is safe and that it is not in danger of being tampered with. In order to access the system, the operator or the technician will need a username and password. Additionally, the data transfer from the sensors to the IoT engine will occur locally to discourage remote attacks on the system. Furthermore, the operator of the train will still play a large role in the operation of the train. The IoT engine will be mainly there to provide data and suggestions to the operator to ensure the proper operation of the train. The operator will always have the last say and any decisions that the system makes can be manually overridden by the operator in the event of a malfunction or an attack on the train's computer. These measures will help to combat the effectiveness of any attacks on the system.

To implement this feature, we need the following requirements:

- R8: The system will require a secure username and password to access or operate it from both employees and administrators.
- R9: The system will feature a manual override option in the event of tampering or faulty sensor data.
- R10: The data transfer from the sensors to the IoT engine will occur locally to prevent outside attacks.

3.2. Functional Requirements

The system must be able to determine the weather conditions based on data it receives from the temperature and humidity/precipitation sensors. The conditions it must be able to recognize include ice, snow, and high winds. Then it will be able to make a recommendation to the operator. Additionally, the system must be able to utilize distance sensors in order to determine if there are objects or other trains on the tracks ahead of the trains. The system will be able to recommend actions to the operator for any situation based on recent train data. Upon meeting the CTO of HTR, it was concluded that at minimum the following four features are needed:

3.2.1. Usage of Sensors for Standing Objects

The function of distance sensors, which are installed in the front of the train, is to detect standing objects on the path of the train and suggest action to the operator; plausible actions include braking, increasing speed, decreasing speed, or stopping the train. To complete this, there will be three distance sensors installed in front of the train. Multiple sensors will provide a failsafe if one is faulty or breaks.

The sensors will provide the distance away from a given object every five seconds. The sensors will also have cognizance of the speed of the train, which will be retrieved from the GPS. Using distance and speed data, the sensors will make decisions on which command to give the conductor. An example scenario in which the distance sensors will be useful is when a train detects an object on the rails 1000 yards away while it is going 100 mph. Here, the suggestion would be to stop the train. If this object is no longer present, the command will change and tell the operator to increase speed.

To implement this feature, we need the following requirements:

- R11: Three distance sensors placed in the front of the train.
- R12: The distance sensors will send information to IoT, which will recommend to the operator: braking, increasing speed, decreasing speed, or stopping the train.
- R13: The distance sensors will provide information on how close an object is in five-second increments.
- R14: The distance sensors will gather data about the speed of the train from the GPS.
- R15: The distance sensors will use both the speed and distance data to make decisions about what to tell the conductor.

- R16: If the sensor detects that action is required, it will display the information on the console.

3.2.2. Usage of Sensors for Moving Objects

Speed and distance sensors, which are installed in the front and the back of the train, will detect moving objects in the path of the train, in front or behind the train. These sensors should suggest action to the operator; once the operator is informed, they should take the appropriate action. In order to effectively implement this feature, as mentioned above, multiple sensors should be installed so that there may be no faults. These sensors should give an accurate reading of the speed of moving objects so that the operator can take the appropriate action. For example, if there was a train in front of the current train, that is going at 60 mph, and the current train is traveling at 90 mph, the operator should get a notification to either slow down or come to a complete stop.

To implement this feature, we need the following requirements:

- R17: Three speed sensors each on the front and the back of the train.
- R18: The speed sensors will provide information about how fast other moving objects are going, which will be resolved into a whole number.
- R19: The distance sensors should alert the speed sensors and give one of the following commands to the operator: brake, increase speed, decrease speed, or stop the train.
- R20: The console will display recommended action if it is required.

3.2.3 Detection of Gate Crossing

With locomotive transportation, a crucial aspect of safety includes having gate crossings. These gate crossings enable trains and locomotive vehicles to continue on the tracks based on whether or not there are obstacles in the specific intersection. By detecting open and closed gate crossing and the distance away from the crossing, the train will be able to safely calculate whether there should be speed change or brake or brake stop to the operator. The distance sensor will be able to detect whether or not the bars are open or closed and adjust speed according to the distance and signal. Additionally, the conductor will be alerted to sound their horn near the gate.

To implement this feature, we need the following requirements:

- R21: A GPS sensor will detect the visual of a gate crossing coming soon on the tracks.
- R22: A camera will show the visibility of the gate crossing from 1000 miles away.

- R23: A distance sensor will recognize the signal and bars from 1000 miles away and notify the operator of what action to take: braking, increasing speed, decreasing speed, or stopping the train.
- R24:The gate crossing will send a signal to the IoT engine 60 seconds before the train arrives.
- R25: The conductor will be alerted to sound the horn for 15 seconds when the gate is 1500 to 1700 meters away. The conductor will then blow the horn for 5 seconds when the gate is less than 100 meters away.

3.2.4. Detection of Wheel Slippage

The system will be able to detect wheel slippage using speed data from the GPS and comparing it with sensor data about the RPM of the wheels. Then after performing calculations and using this data, the train will be able to make a recommendation to the operator whether they should brake or slow down. This recommendation will be based on a threshold for slippage for the train. This feature will require the technician to input the size of the wheel prior to the operation of the train in order for the calculations to be performed.

To implement this feature, we need the following requirements:

- R26: The wheel sensors will record the RPM of the train.
- R27: The IoT will store the theoretical RPM based on the wheel size and the speed received from GPS data.
- R28: The IoT will make a recommendation to the operator based on any discrepancies between the RPM from the sensor and the RPM from the GPS data.
- R29: The console will display the recommendation to the operator on the screen so the operator can act on the recommendation.

4. Requirements Modeling:

4.1 Use Cases:

4.1.1 Activate the IoT Engine:

Use Case No: 1

Primary Actor: Conductor or Administrator

Secondary Actor: IoT Engine

Goal: To activate the IoT Engine

Preconditions: IoT Engine is turned off and not yet activated.

Trigger: Operator activates the IoT Engine.

Scenario:

1) Operator turns on IoT Engine.
2) IoT Engine connects to the server.
3) IoT Engine loads all processes.
4) IoT Engine awaits username and password.

Exceptions:

1) If the IoT Engine fails to start properly, the operator will be asked to restart the system.

4.1.2 Conductor logs into IoT engine:

Use Case No: 2

Primary Actor: Conductor

Secondary Actor: IoT Engine

Goal: To start the IoT Engine and secure access to the IoT Engine.

Preconditions: The system is activated but locked.

Trigger: The conductor enters a username and password.

Scenario:

1) The conductor enters a username and password.
2) The IoT Engine ensures that the username and password are valid.
3) The IoT Engine begins to display data.

Exceptions:

1) If the username and password are incorrect then the display screen will re-prompt the conductor for a different username and password and display an error message.

4.1.3 Administrator logs into IoT engine:

Use Case No: 3

Primary Actor: Administrator

Secondary Actor: IoT Engine

Goal: To allow the administrator access to the IoT Engine.

Preconditions: The system is activated but locked.

Trigger: The administrator enters a username and password.

Scenario:

1) The administrator enters a username and password.
2) The IoT Engine ensures that the username and password are valid.
3) The IoT Engine begins to display administrator data and gives access to the administrative system.

Exceptions:

1) If the username and password are incorrect then the display screen will re-prompt the administrator for a different username and password and display an error message.

4.1.4 Detection of hazardous weather conditions:

Use Case No: 4

Primary Actor: Precipitation and Humidity Sensors

Secondary Actor: GPS Speed Sensor, Display Screen, Conductor

Goal: Notify the conductor about the hazardous conditions on the tracks

Preconditions: The train engine is started and the conductor is logged in

Trigger: Precipitation and Humidity Sensors detect high levels of precipitation and humidity

Scenario:

1) Precipitation and humidity sensors also detect abnormal levels of precipitation and humidity
2) Sensors send data to IoT Engine, which compares current precipitation and humidity level to precipitation and humidity level on an average day
3) Data is captured by the Time Sensitive Networking Router (TSNR) and is ready for the IoT Engine
4) If precipitation and humidity are above a certain threshold, then the IoT Engine alerts the conductor via the display screen
5) IoT Engine has a warning appear on the display to notify the conductor to decrease the speed due to external conditions
6) The conductor makes necessary adjustments to the train speed

Exceptions:

1) If the Precipitation Sensor is unable to retrieve information about the distance from stationary objects on the tracks, then the display screen will display an error message
2) If the Humidity Sensor is unable to retrieve information about the distance from stationary objects on the tracks, then the display screen will display an error message

4.1.5 Detection of a stationary obstacle on the tracks

Use Case No: 5
Primary Actor: Distance Sensor
Secondary Actor: IoT Engine, Display Screen, GPS Speed Sensor, Conductor
Goal: Notify the conductor that there is an object on the tracks
Preconditions: The train engine is started and the conductor is logged in
Trigger: Distance Sensor senses an object on the tracks
Scenario:

1) IoT Distance Sensor detects distance to another foreign object within 1000 miles
2) The distance sensors will provide information on how close an object is in five-second increments.
3) The distance sensors will send information to IoT, which will recommend to the operator
4) The GPS speed and information about how far the object is from the distance sensor will be sent to the IoT Engine to calculate the amount of time until collision.
5) Data is captured by the Time Sensitive Networking Router (TSNR) and is ready for the IoT Engine
6) The display screen will display a recommendation to the IoT Engine to brake, decrease speed, and stop the train
7) If the object is no longer detected by the distance sensor, the display screen will prompt the conductor to increase speed
8) Warning notification appears on Display Screen regarding the object in the tracks

Exceptions:

1) If the Distance Sensor is unable to retrieve information about the distance from stationary objects on the tracks, then the display screen will display an error message

4.1.6 Detection of a moving obstacle on the tracks:

Use Case No: 6

Primary Actor: Distance Sensor
Secondary Actor: IoT Engine, GPS Speed Sensor, Display Screen, Conductor
Goal: Notify the conductor that there is a moving object on the tracks
Preconditions: The train engine is started and the conductor is logged in
Trigger: Distance Sensor senses a moving object on the tracks
Scenario:

1) The distance sensor detects distance to another foreign object within 1000 miles that is not stationary
2) The distance sensors will provide information on how close an object is in five-second increments.
3) The distance sensors will send information to IoT, which will recommend to the operator
4) The GPS speed and information about how far the object is from the distance sensor will be sent to the IoT Engine to calculate the amount of time until collision
    a) If the distance between the train and the object is increasing, then the moving object is getting farther away from the train
    b) If the distance between the train and the object is decreasing, then the moving object is getting closer to the train
5) Data is captured by the Time Sensitive Networking Router (TSNR) and is ready for the IoT Engine
6) The display screen will display a recommendation to the IoT Engine to brake, decrease speed, and stop the train
7) If the object is no longer detected by the distance sensor, the display screen will prompt the conductor to increase speed
8) Warning notification appears on Display Screen regarding the object in the tracks

Exceptions:

1) If the Distance Sensor is unable to retrieve information about the distance from moving objects on the tracks, then the display screen will display an error message

4.1.7 Detection of wheel slippage:
Use Case No: 7
Primary Actor: Wheel Sensors
Secondary Actor: IoT Engine, GPS Speed Sensor, Display Screen, Conductor
Goal: Notify the conductor about the abnormal conditions of the tracks
Preconditions: The train engine is started
Trigger: Discrepancy between speed detected by wheel sensor and GPS speed

Scenario:

1) Wheel Sensors detect wheel RPM and send information to IoT Engine
2) IoT Engine takes the average of the given information to calculate the speed of the train based on RPM average and wheel diameter
3) GPS Speed Sensor sends train speed to the IoT Engine
4) If the IoT detects a 0.1 difference between the GPS speed and wheel speed, then there is apparent slippage
5) Data is captured by the Time Sensitive Networking Router (TSNR) and is ready for the IoT Engine
6) The IoT will make a recommendation to the operator based on any discrepancies between the RPM from the sensor and the RPM from the GPS data
7) IoT Engine has the slippage warning appear on the display to notify the conductor to put sand on the tracks and slow down

Exceptions:

1) If the GPS Speed Sensor is unable to retrieve train's average speed, then the display screen will display an error message
2) If the Wheel Sensor is unable to retrieve wheel's RPM, then the display screen will display an error message

4.1.8 Detection of gate crossing ahead:

Use Case No: 8

Primary Actor: Distance Sensor

Secondary Actor: IoT Engine, Display Screen, Conductor

Goal: Notify the conductor about upcoming gate

Preconditions: The train engine is started and the conductor is logged in

Trigger: Distance Sensor detects an upcoming gate crossing

Scenario:

1) The distance sensor detects gate crossing ahead
2) The distance sensor will recognize the signal and bars from 1000 miles away and notify the operator of what action to take
3) The gate crossing will send a signal to the IoT engine 60 seconds before the train arrives. The gates will be fully risen 30 seconds before the train arrives and lower 60 seconds after the end of the train clears the island circuit.
4) The conductor will be alerted to sound the horn for 15 seconds when the gate is 1500 to 1700 meters away. The conductor will then blow the horn for 5 seconds when the gate is less than 100 meters away.

5) Data is captured by the Time Sensitive Networking Router (TSNR) and is ready for the IoT Engine
6) If the distance sensor detects that the gate is not open, then it will prompt the conductor to brake, decrease speed, and stop the train
7) If the distance sensor detects that the gate is open, then it will prompt the conductor to maintain current speed

Exceptions:

1) If the Distance Sensor is unable to retrieve information about the distance from moving objects on the tracks, then the display screen will display an error message

4.1.9 Detection of a disconnection from network:

Use Case No: 9

Primary Actor: IoT Engine

Secondary Actor: Display Screen, Conductor, Administrator

Goal: Alert the conductor that IoT Engine must be switched on

Preconditions: The train engine is started and the conductor is logged in

Trigger: IoT Engine detects that there is no access to WiFi or Cellular

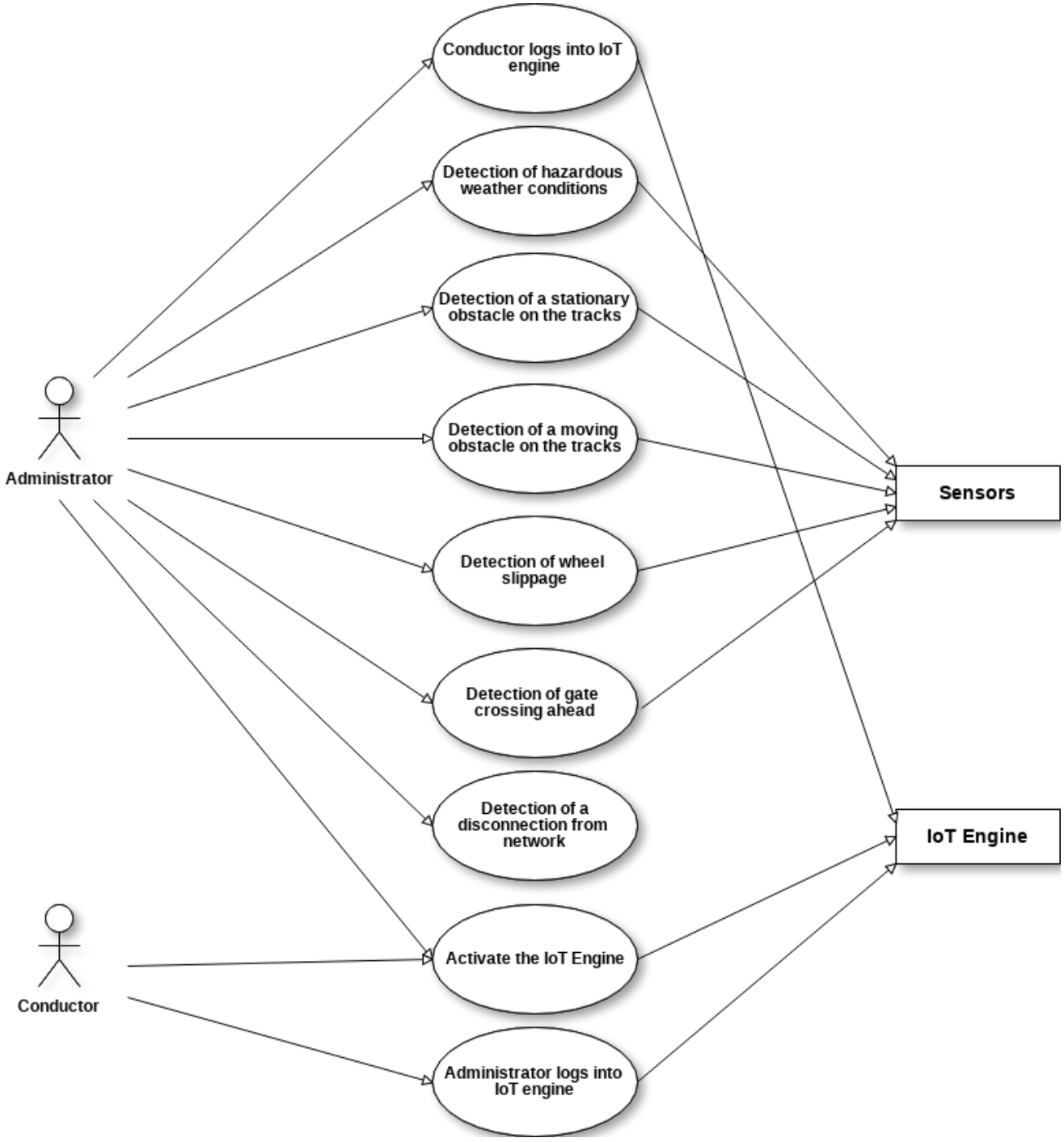
Scenario:

1) The IoT Engine detects that there is no access to WiFi or Cellular
2) The display screen displays a message that alerts the conductor that there is no connectivity and that the IoT Engine must be turned on
3) If connectivity is reestablished, then the display screen alerts the conductor that there is WiFi and Cellular and prompts the conductor to turn off the IoT Engine

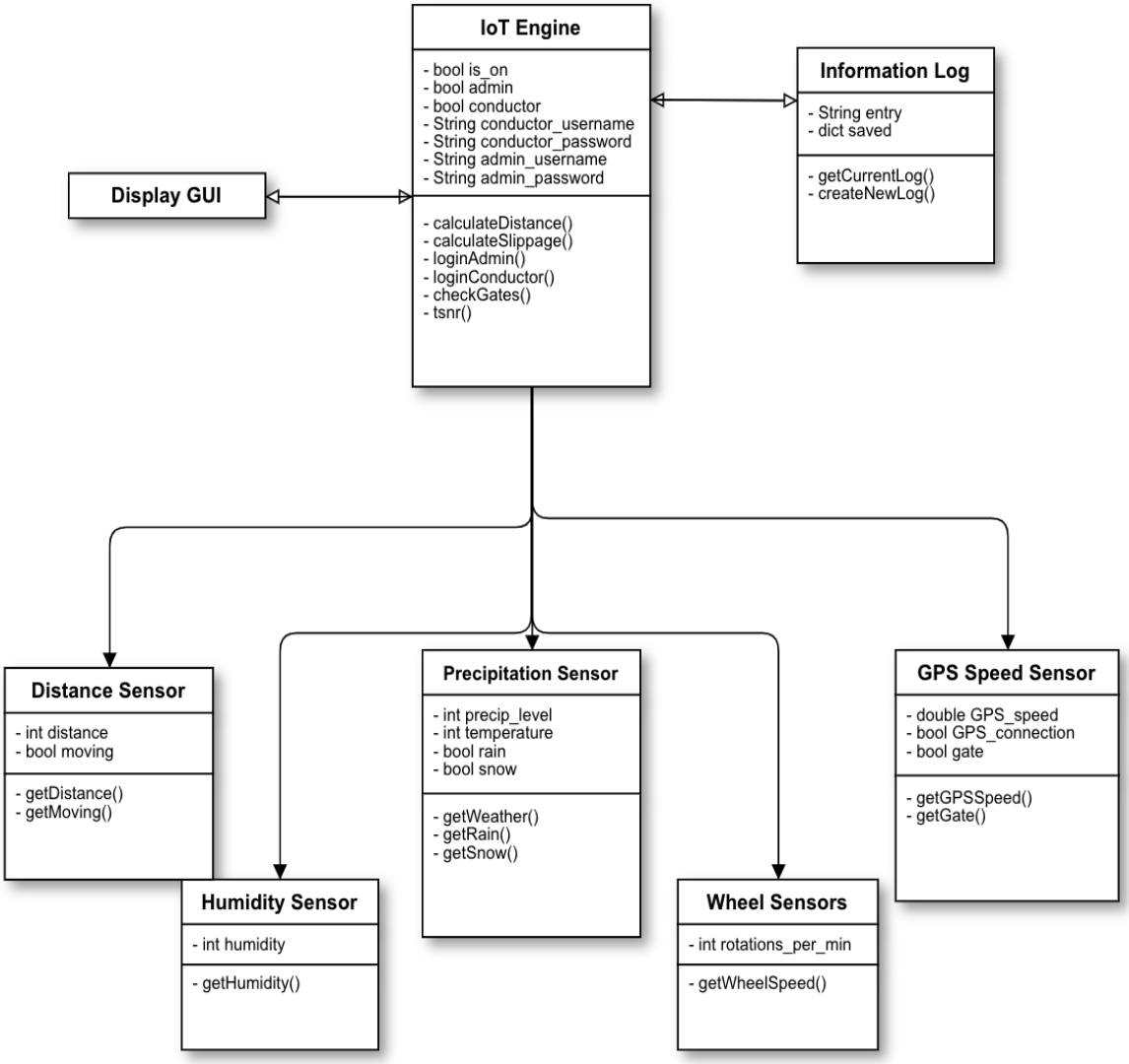
Exceptions:

1) If the IoT Engine is unable to retrieve information about connectivity, then the display screen will display an error message

4.2 Use Case Diagram



4.3 Class-Based Modeling



4.4 CRC Modeling/Card

4.4.1 IoT Engine:

RESPONSIBILITY	COLLABORATORS
calculateDistance(): Calculates the distance away from a moving object and displays a warning	Sensors, Display Screen
calculateSlippage(): Calculates if there is slippage	Sensors, Display Screen
loginAdmin(): logs in through admin username and password	Display Screen
loginConductor(): logs in through conductor username and password	Display Screen
checkGates(): checks if there is a gate nearby	Sensors, Display Screen
tsnr(): gets data from sensors and does calculations	Sensors, Display Screen

4.4.2 Information Log:

RESPONSIBILITY	COLLABORATORS
getCurrentLog(): retrieves information in the current data log	IoT Engine
createNewLog(): creates a new log for every train trip	IoT Engine

4.4.3 GPS Speed Sensor:

RESPONSIBILITY	COLLABORATORS
getGate(): returns true is a gate is nearby	IoT Engine, Sensors

getGPSSpeed(): gets normal train speed from GPS Speed Sensor	IoT Engine, Sensors

4.4.4 Distance Sensor:

RESPONSIBILITY	COLLABORATORS
getDistance(): calculates the distance away from a given object surrounding the train that is sensed on the tracks	IoT Engine, Sensors
getMoving(): checks if object is moving	IoT Engine, Sensors

4.4.5 Humidity Sensor:

RESPONSIBILITY	COLLABORATORS
getHumidity(): calculates the humidity	IoT Engine, Sensors

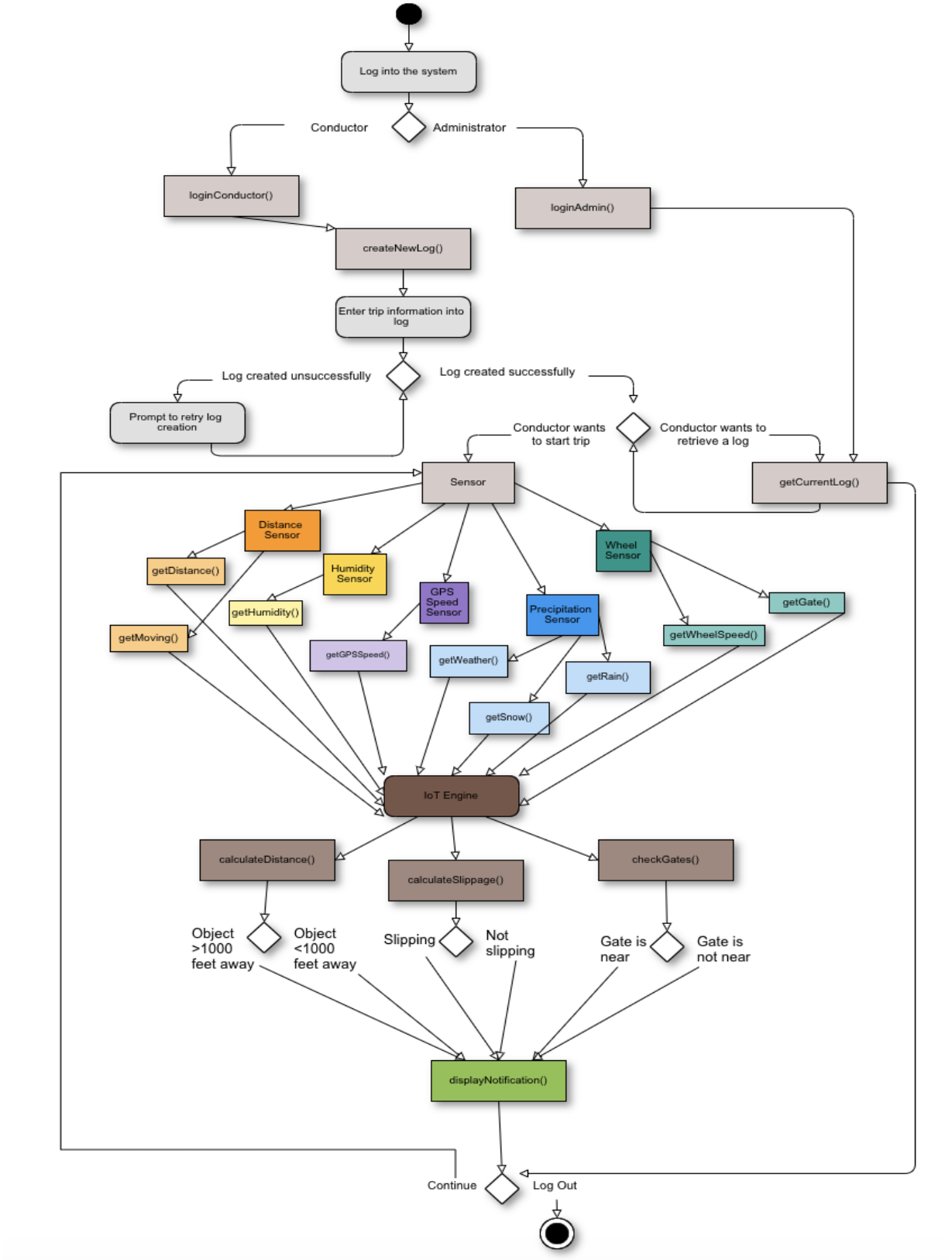
4.4.6 Precipitation Sensor:

RESPONSIBILITY	COLLABORATORS
getWeather(): checks the weather conditions	IoT Engine, Sensors
getRain(): checks if there is rain	IoT Engine, Sensors
gerSnow(): checks if there is snow	IoT Engine, Sensors

4.4.7 Wheel Sensor:

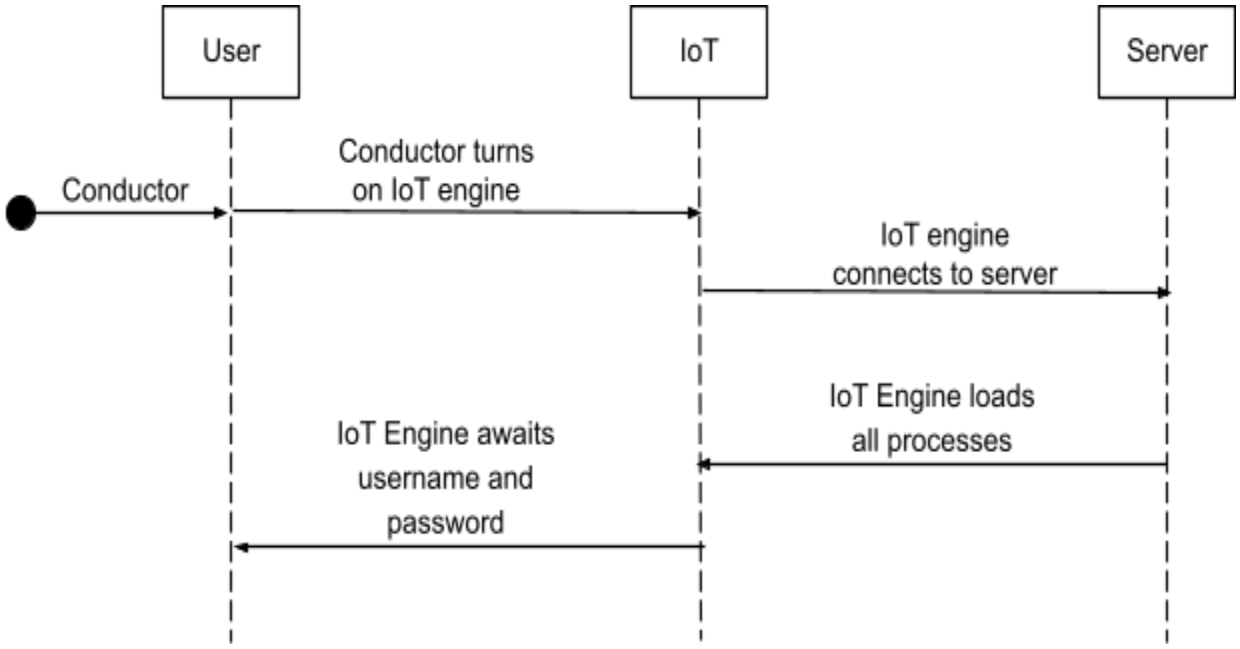
RESPONSIBILITY	COLLABORATORS
getWheelSpeed(): calculate the rotations per minute of the wheels	IoT Engine, Sensors

4.5 Activity Diagram

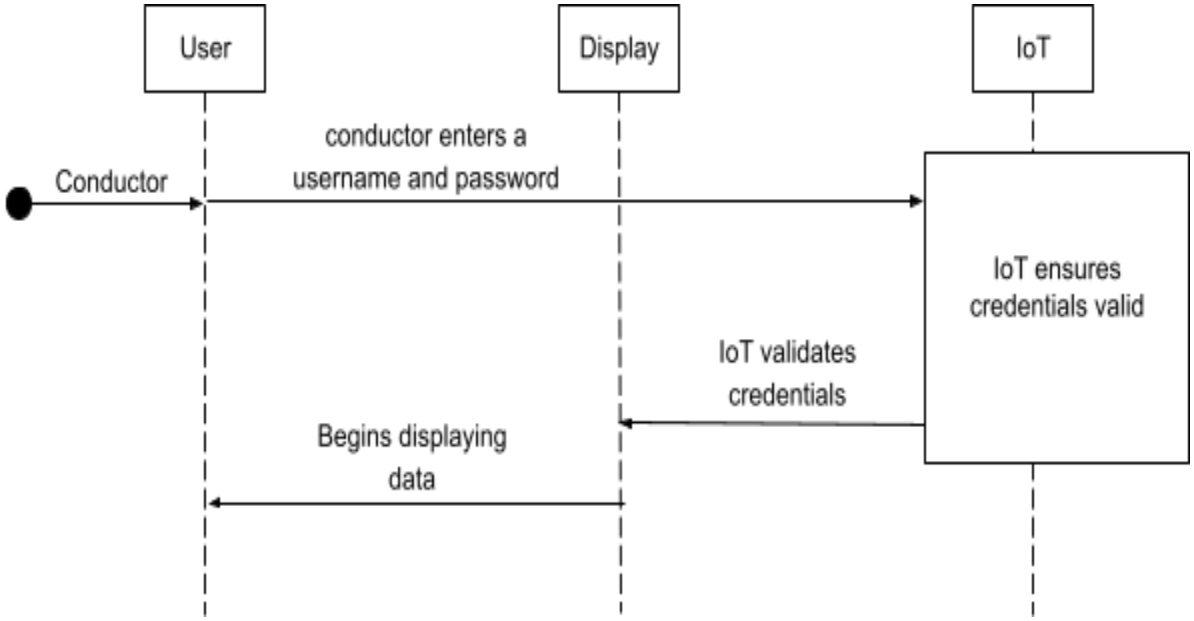


4.6 Sequence Diagram

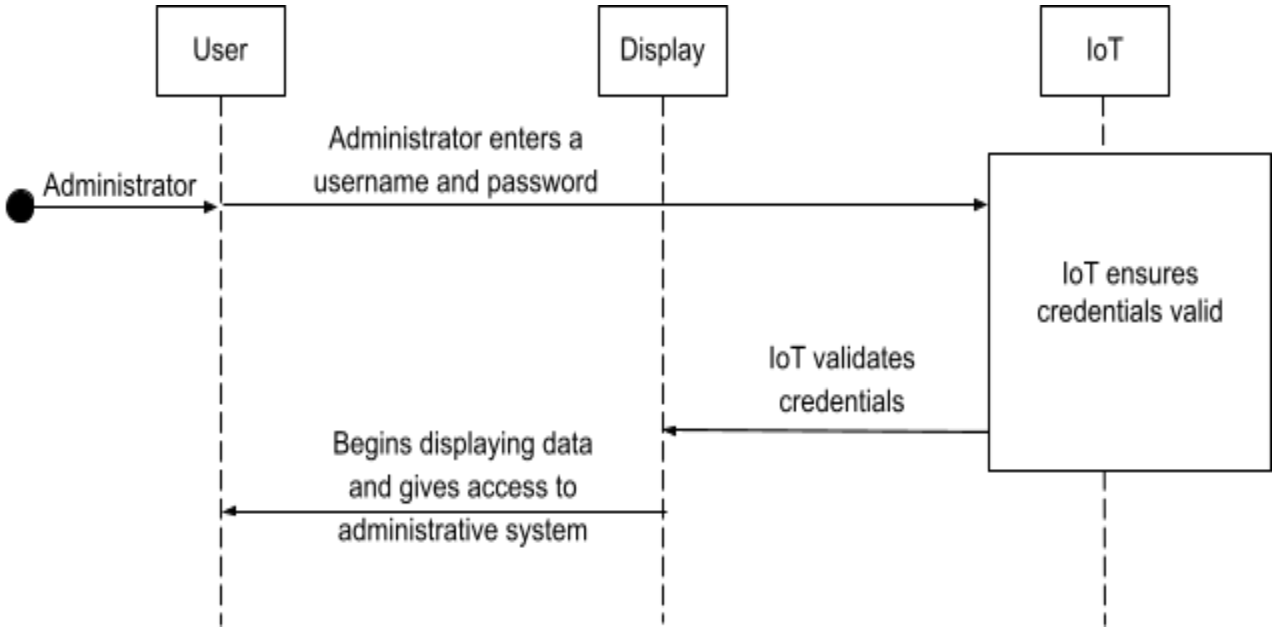
4.6.1 Sequence Diagram for Activate the IoT Engine:



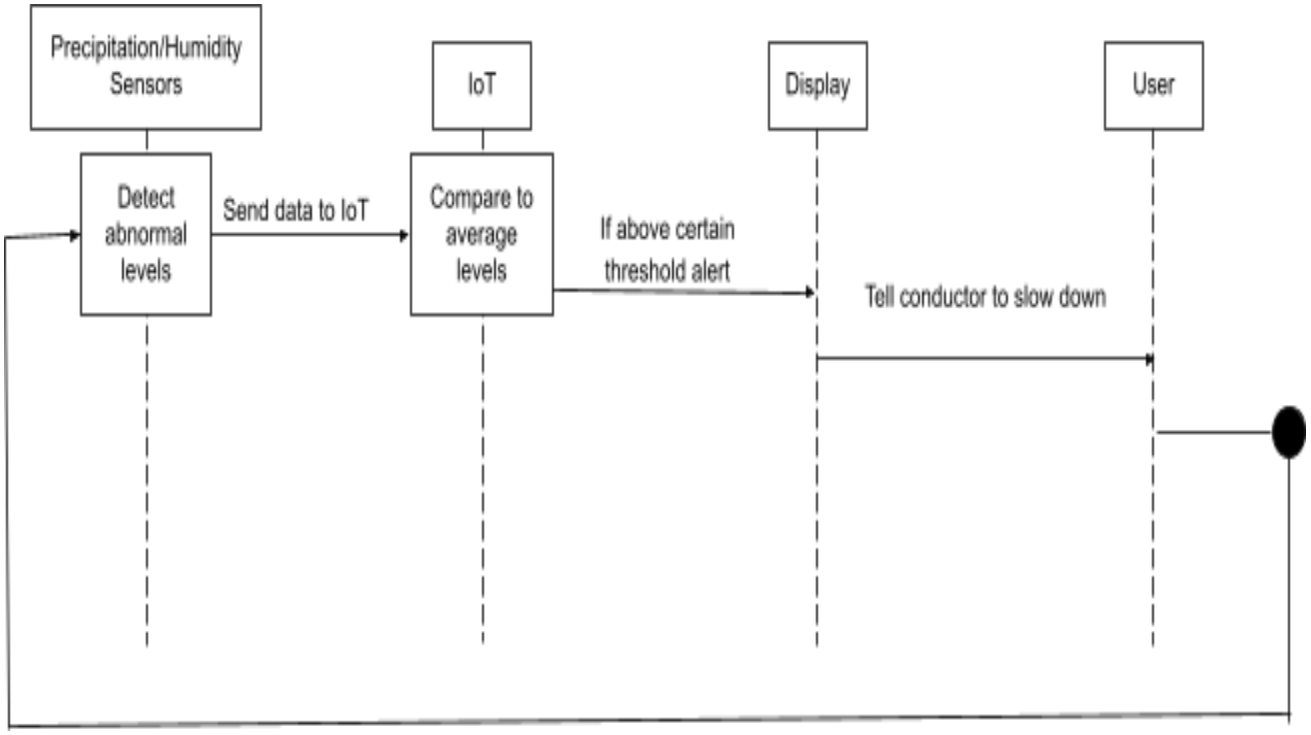
4.6.2 Sequence Diagram for Conductor logs into IoT engine:



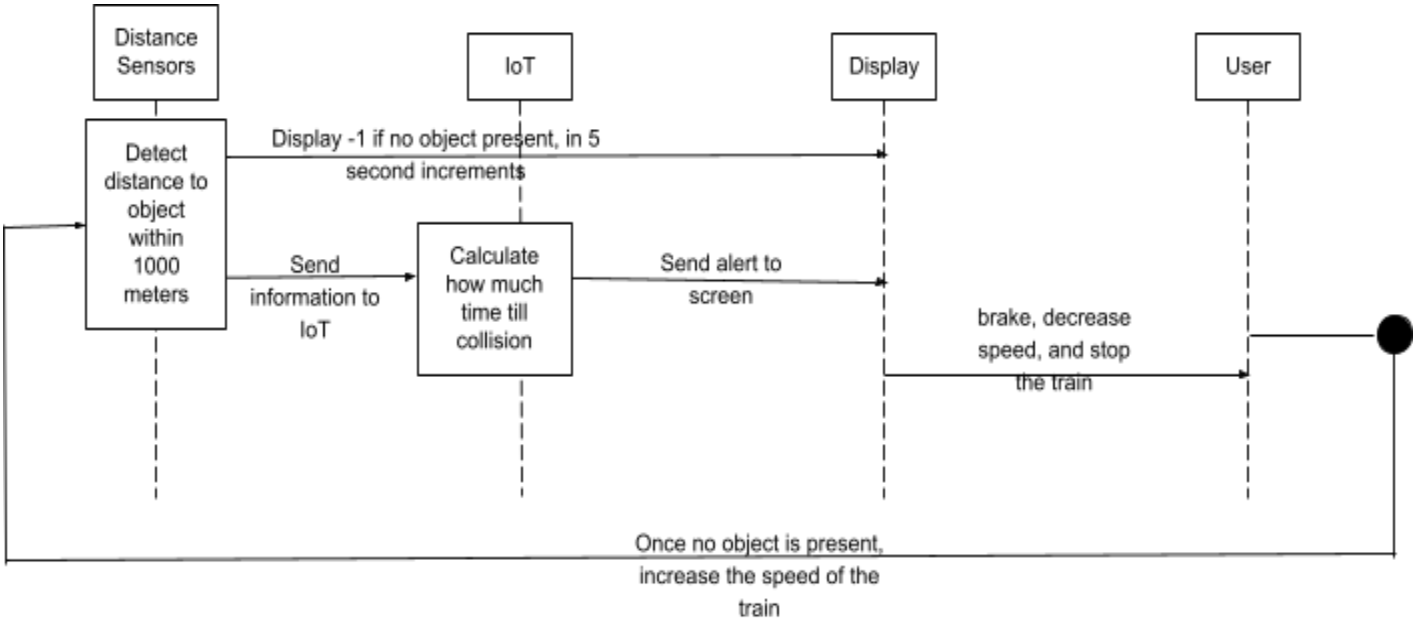
4.6.3 Sequence Diagram for Administrator logs into IoT engine:



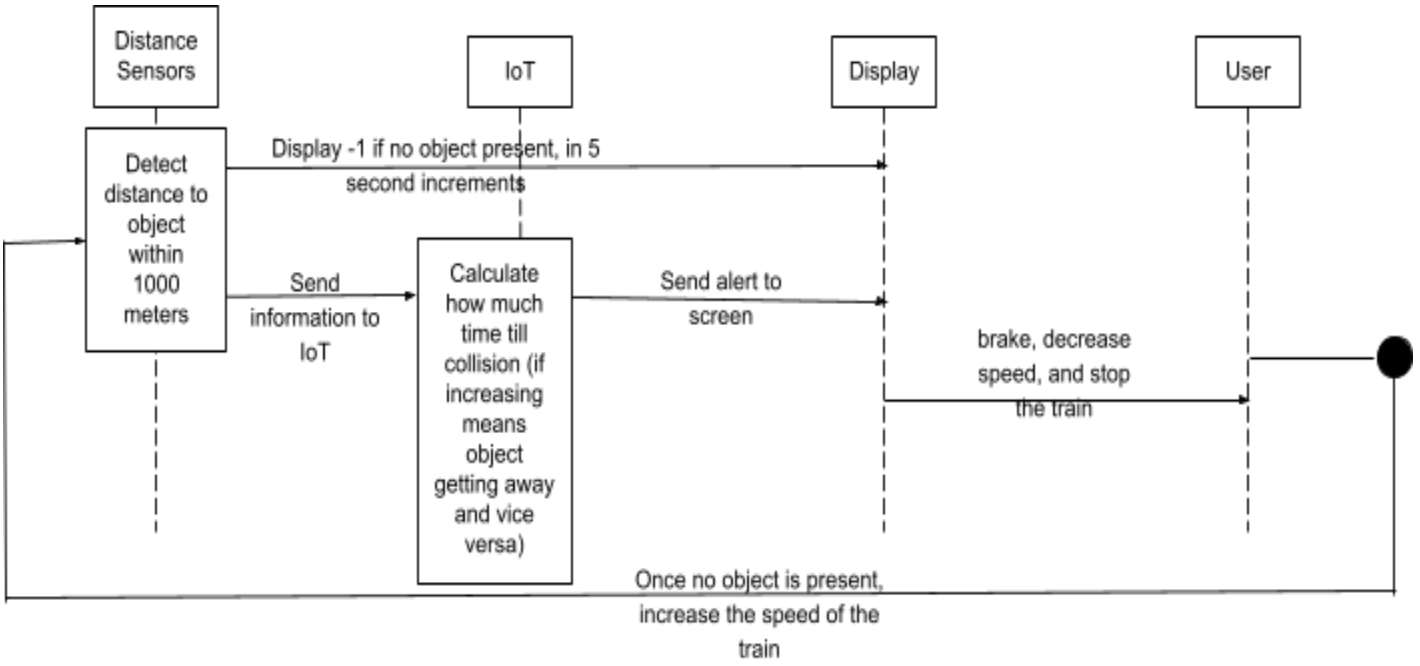
4.6.4 Sequence Diagram for Detection of Hazardous Weather Conditions:



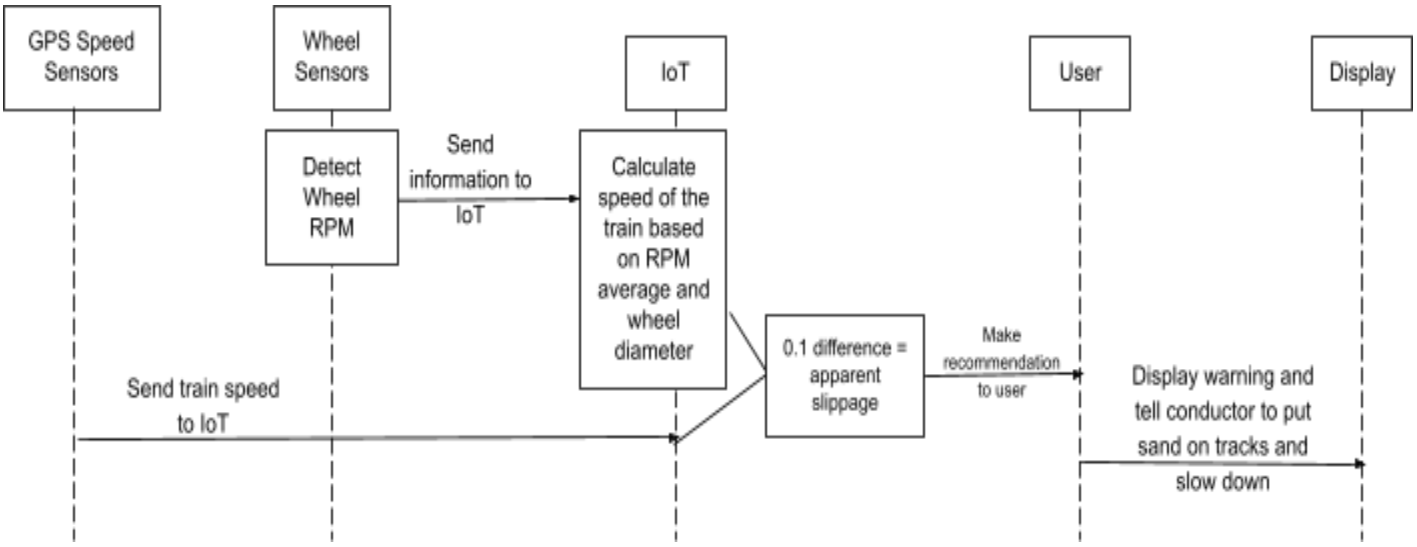
4.6.5 Sequence Diagram for Detection of a stationary obstacle on the tracks:



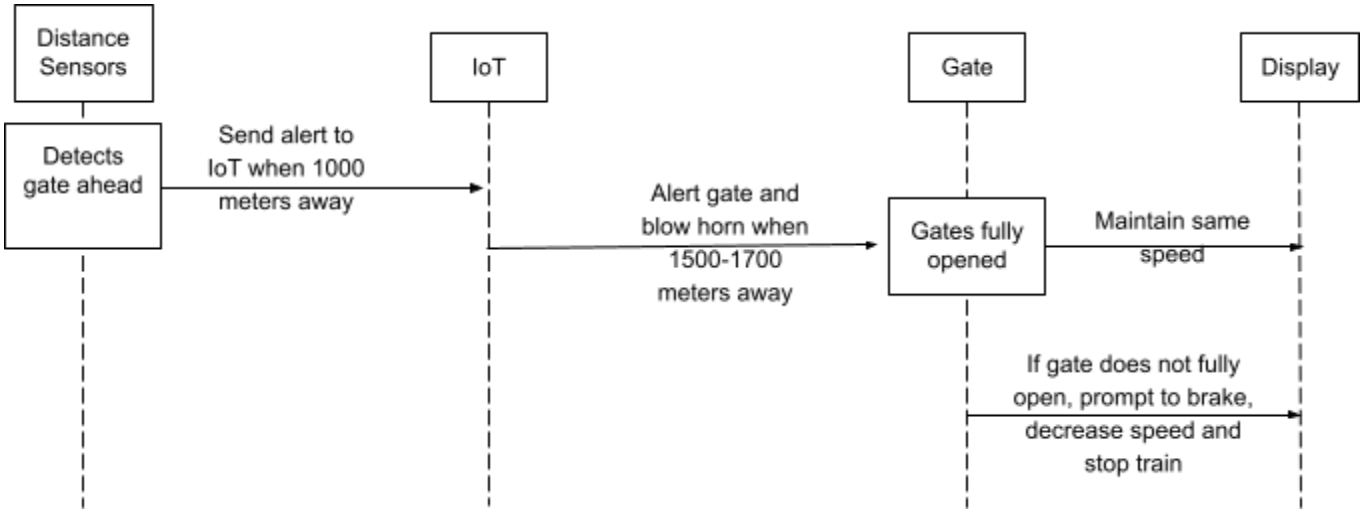
4.6.6 Sequence Diagram for Detection of a moving obstacle on the tracks:



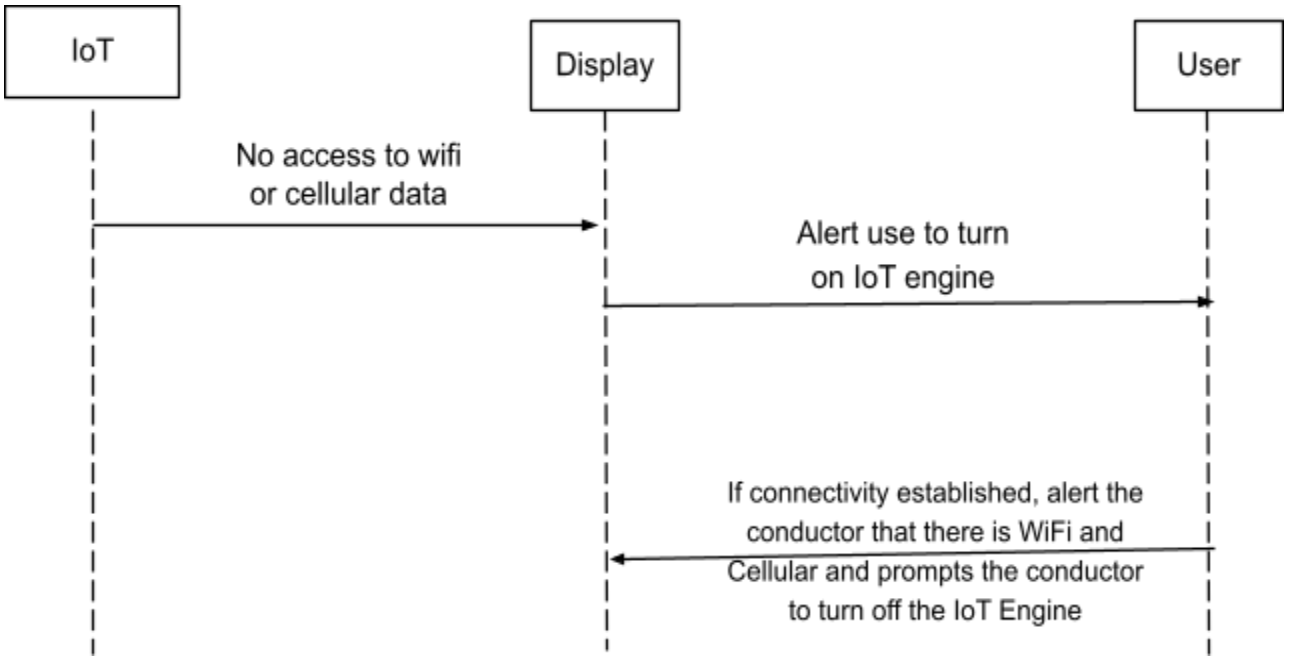
4.6.7 Sequence Diagram for Detection of wheel slippage:



4.6.8 Sequence Diagram for Detection of gate crossing ahead:

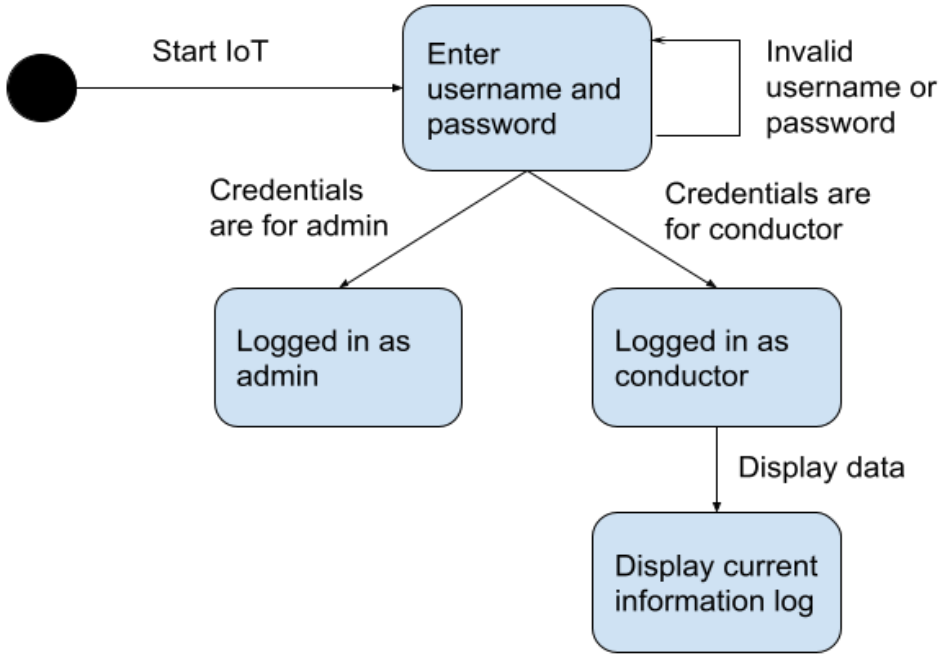


4.6.9 Sequence Diagram for Detection of a disconnection from network:

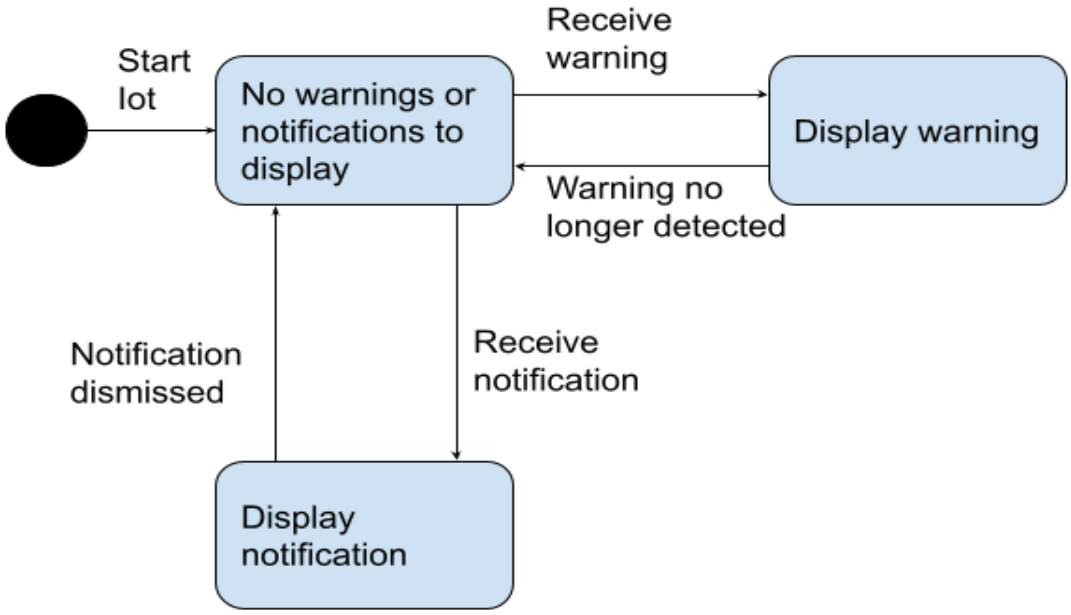


4.7 State Diagrams

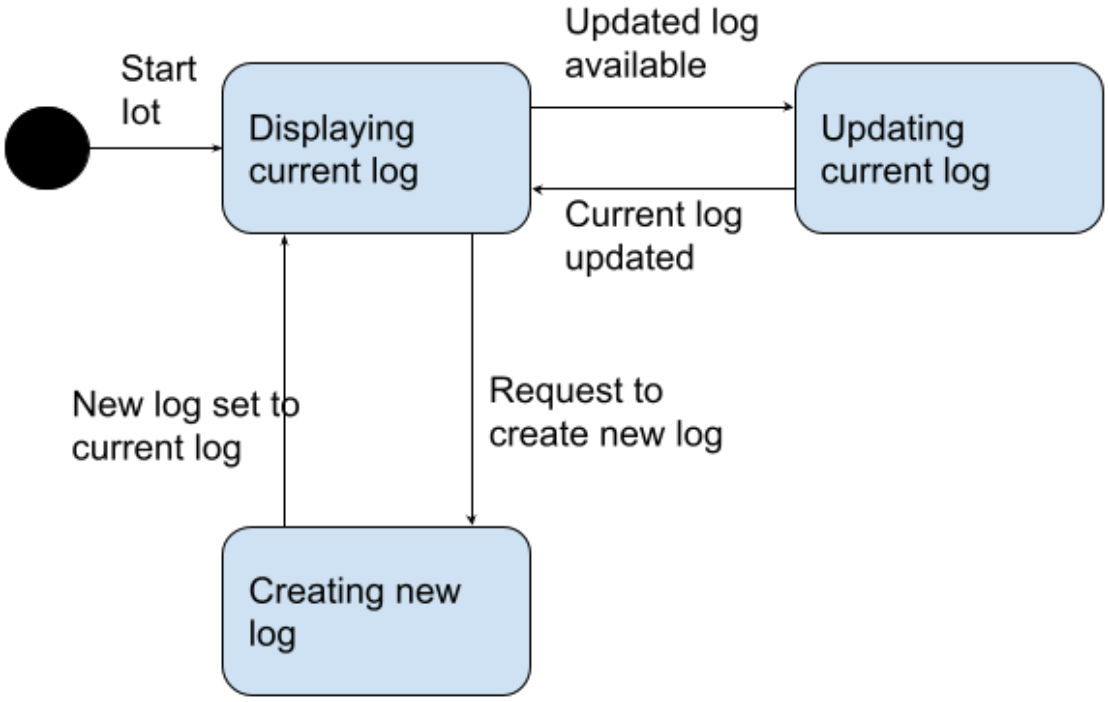
4.7.1 IoT Engine State Diagram:



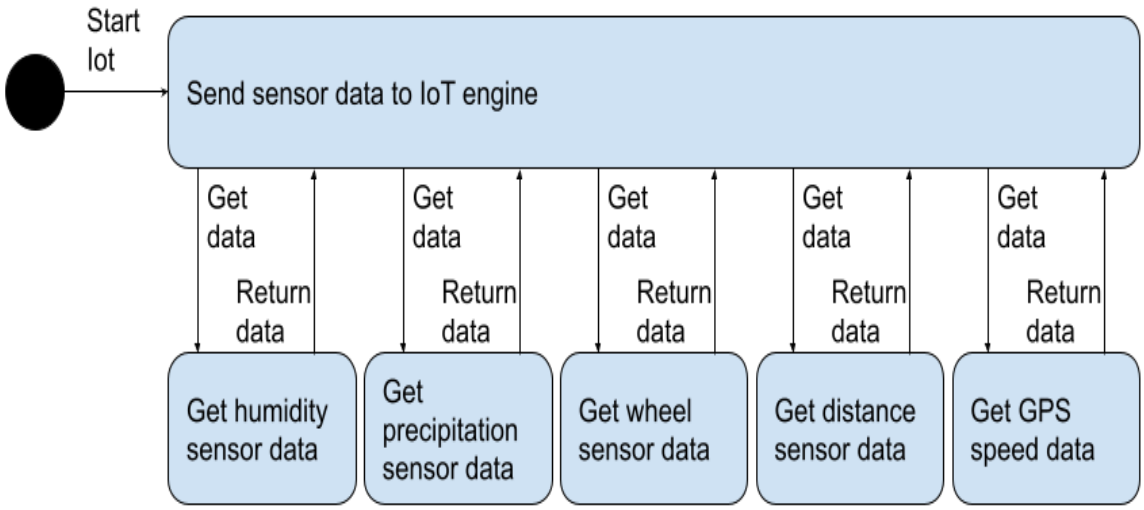
4.7.2 Display Screen State Diagram:



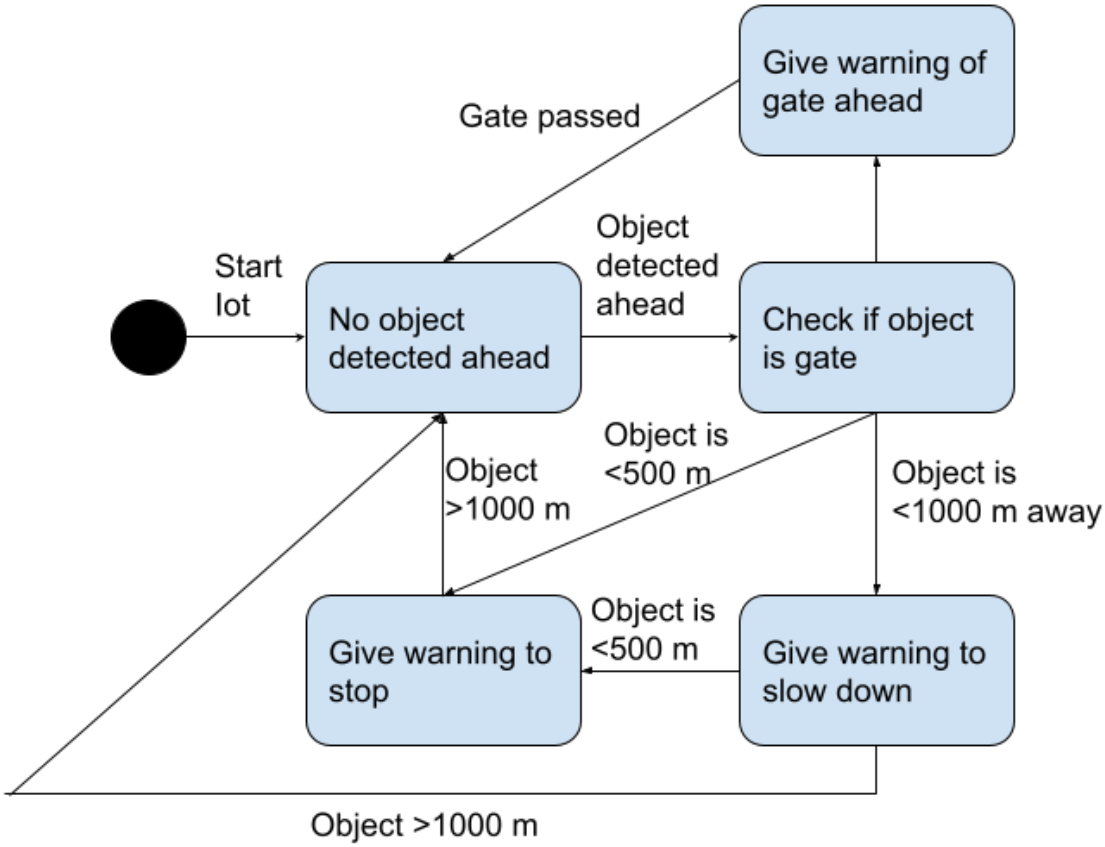
4.7.3 Information Log State Diagram:



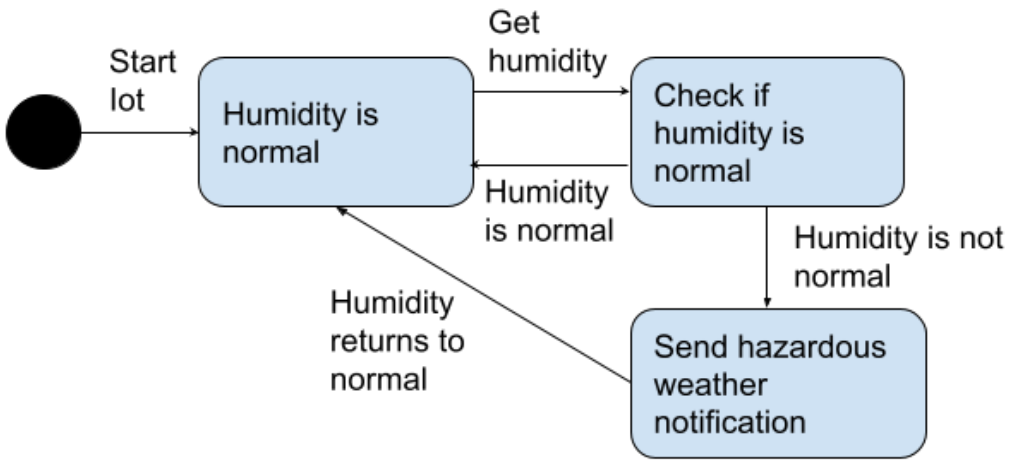
4.7.4 Sensors State Diagram:



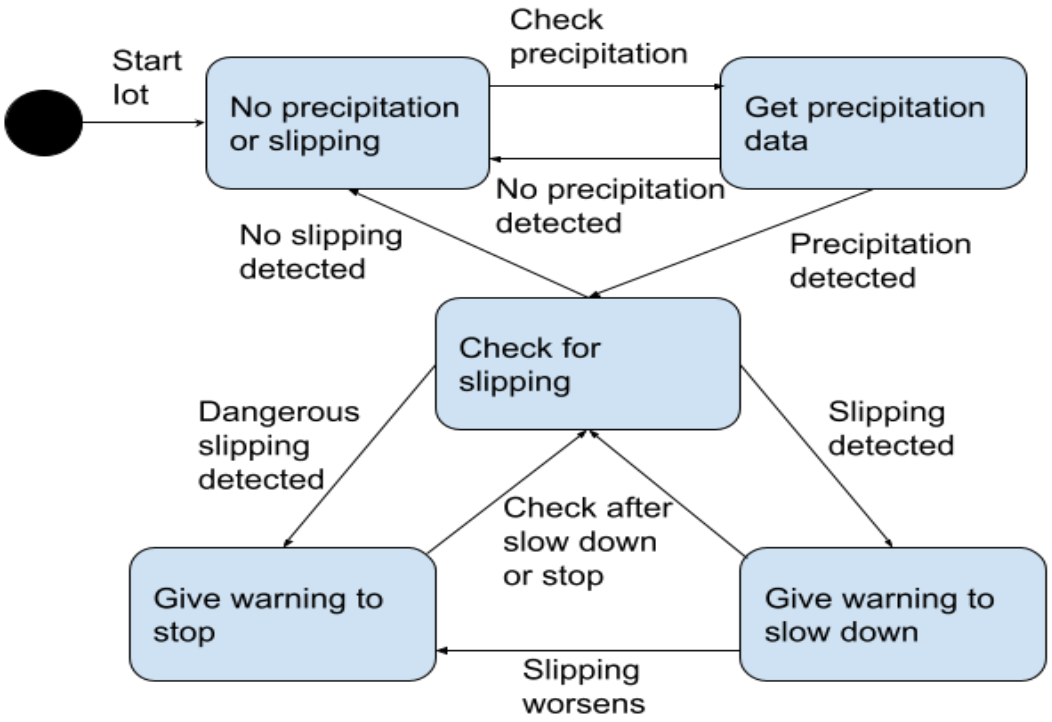
4.7.5 Distance Sensor State Diagram:



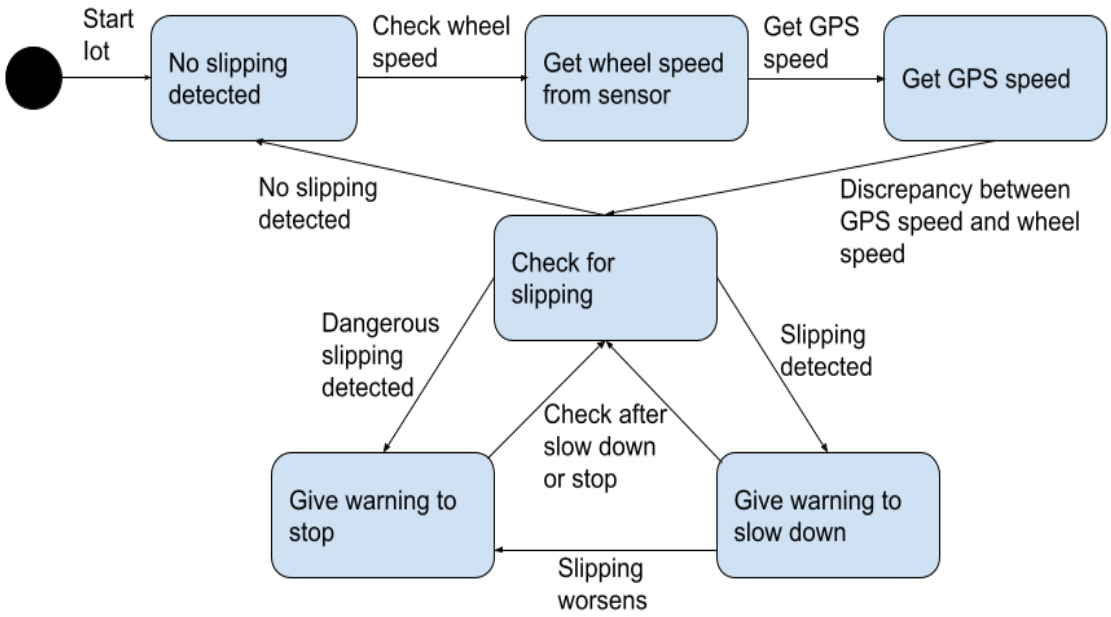
4.7.6 Humidity Sensor State Diagram:



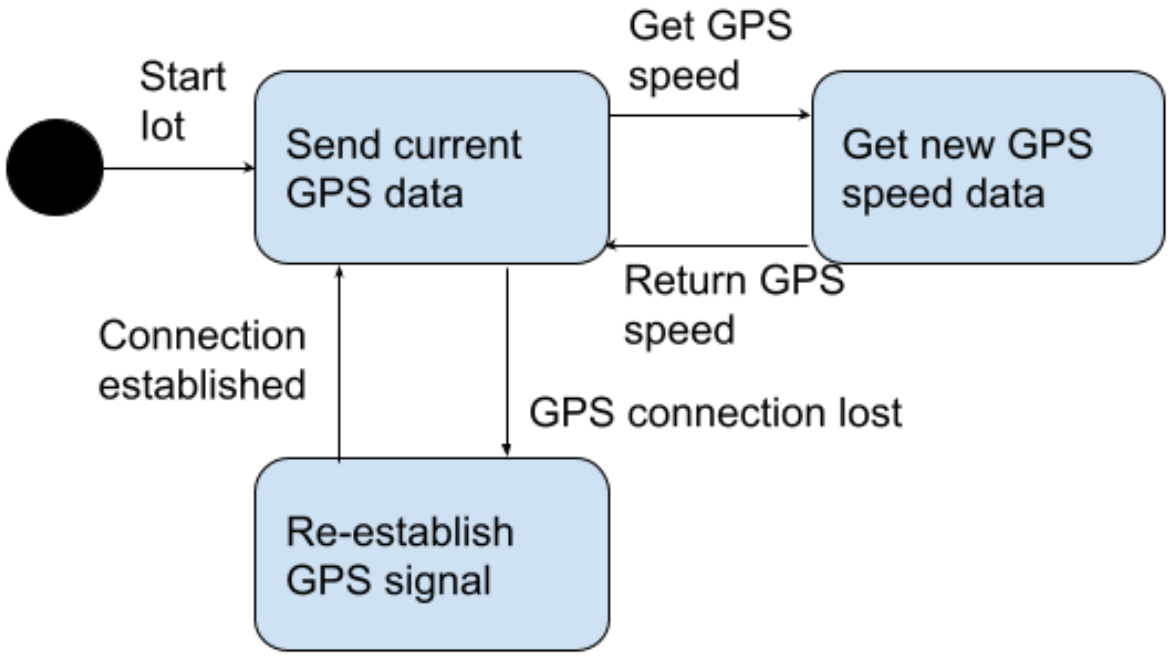
4.7.7 Precipitation Sensor State Diagram:



4.7.8 Wheel Sensor State Diagram:



4.7.9 GPS Speed Sensor State Diagram:

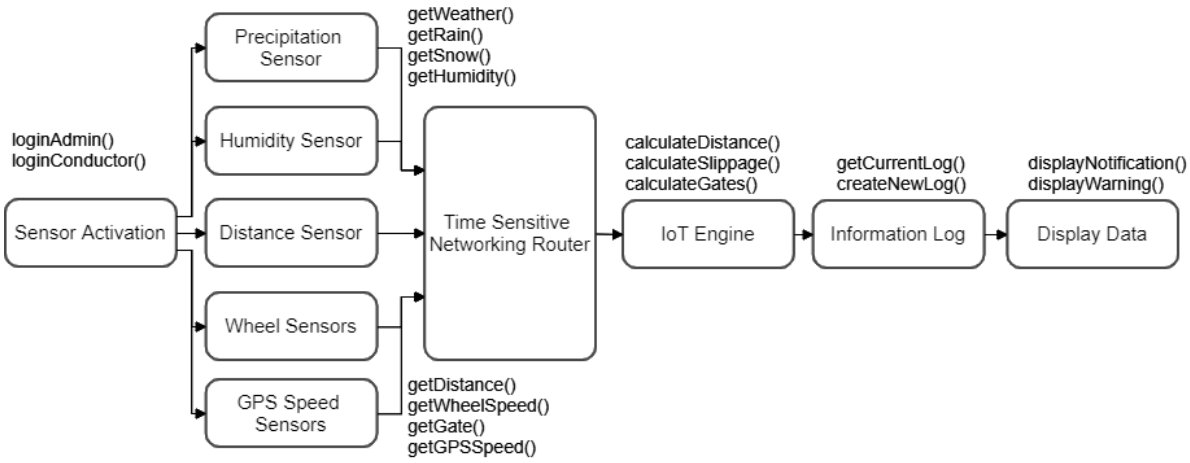


5. Software Architecture:

In this section, we will discuss the pros and cons of different software architectures. The evaluation of pros and cons leads us to choose a first choice and second choice model. Other architectures are also listed, and their cons provide insights into why they were not chosen.

5.1 Software Architecture Model – First Choice

5.1.1 Data Flow Architecture:



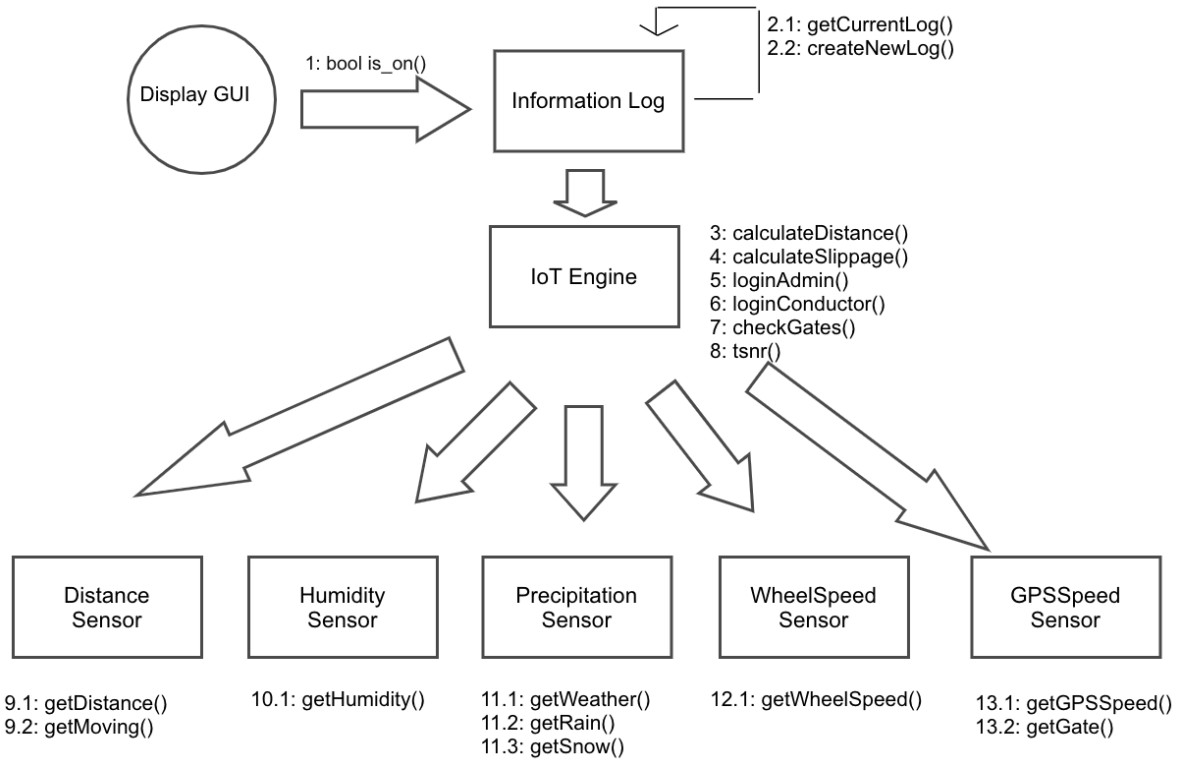
The Data Flow Architecture was the team's first choice as its structure mimics that of the IoT HTR system as data flows from one process to another similar to how data should flow in the IoT HTR system. The pros of this architecture include that data can be easily passed from one process to another and then eventually outputted. Furthermore, another pro is that the data types can be easily manipulated in order to operate with any class. Lastly, the team felt the model was valuable because of its logical and temporal progression of data that makes it easy to follow and understand. Since the process of the system is event-driven, this model shows how real-time data flows through the system.

One of the cons of this architecture is that it is difficult for data to move backwards in the flow. However, the team felt that this would not be an issue as the data in the IoT HTR system will follow a very linear progression. Another drawback of this architecture is that the data is only accessible to components that it has been passed on to and is not available to all components. Another con is that the processes act independently and could not work cooperatively. Despite these minor shortcomings of the data flow architecture, the team felt that it was the best architecture for the IoT HTR system.

The implementation of this model is as follows. The data from the sensors are sent through the Time Sensitive Networking Router to ensure that there are no disparities in time difference between the arrival of data. Next, the data is sent to the IoT Engine for further analysis and calculation. The data is then logged in the information log and displayed on the screen.

5.2 Software Architecture Model – Second Choice

5.2.1 Object-Oriented Architecture:

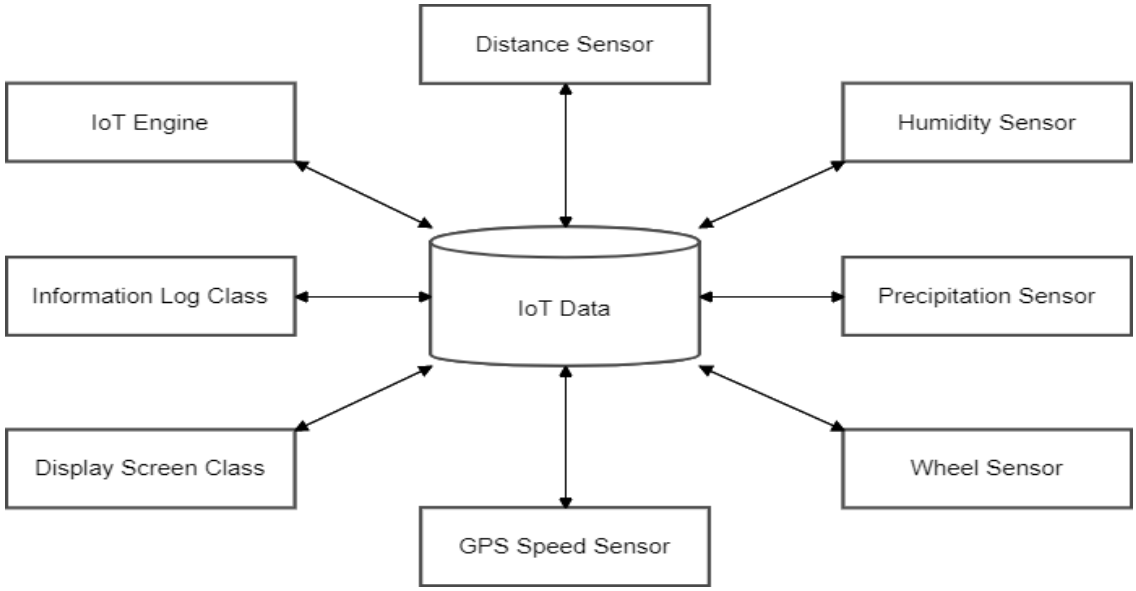


The Object-Oriented Model was the team's second choice. The team recognized that this diagram has many strengths, but ultimately, its strengths did not outweigh the strengths of the Data Flow Architecture Model. The pros for the Object-Oriented Architecture include having the ability to show the relationships among the objects in addition to the messages being sent between them and emphasizing the relationships among the objects and classes for the IoT Engine. Moreover, it maps the application to real world objects for making the code more

understandable and easy to maintain; it improves the quality of the system due to code reuse through polymorphism and abstraction. It also improves testability through encapsulation and reduces the development time and cost. Despite this, Object-Oriented Architecture is not good for event-driven message passing due to its larger and slower program abilities, which is a good method for the IoT Engine.

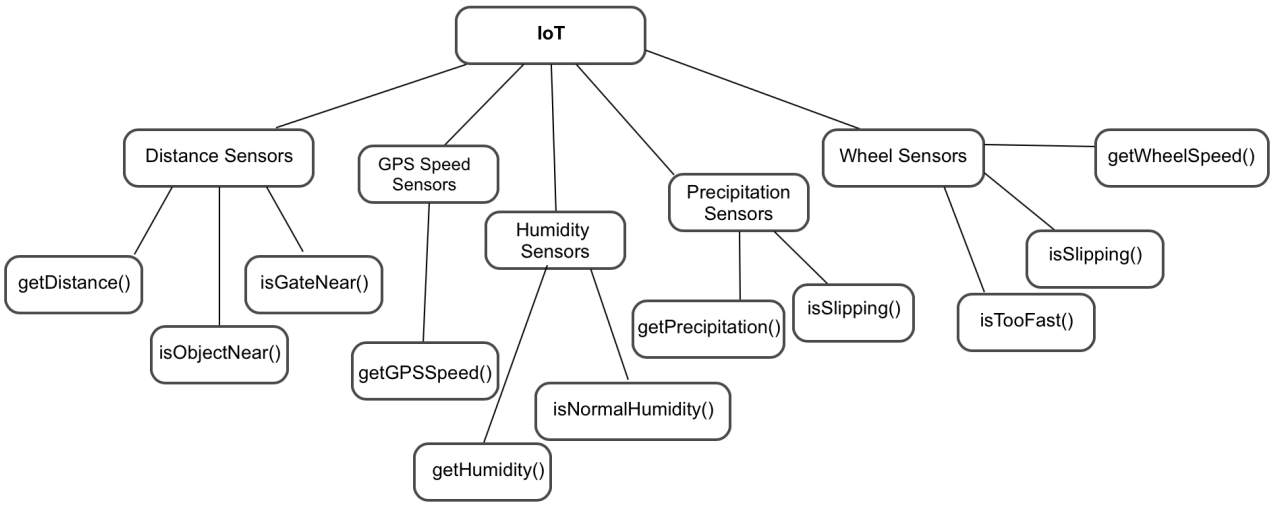
5.3 Other Architecture Models

5.3.1 Data Centered Architecture:



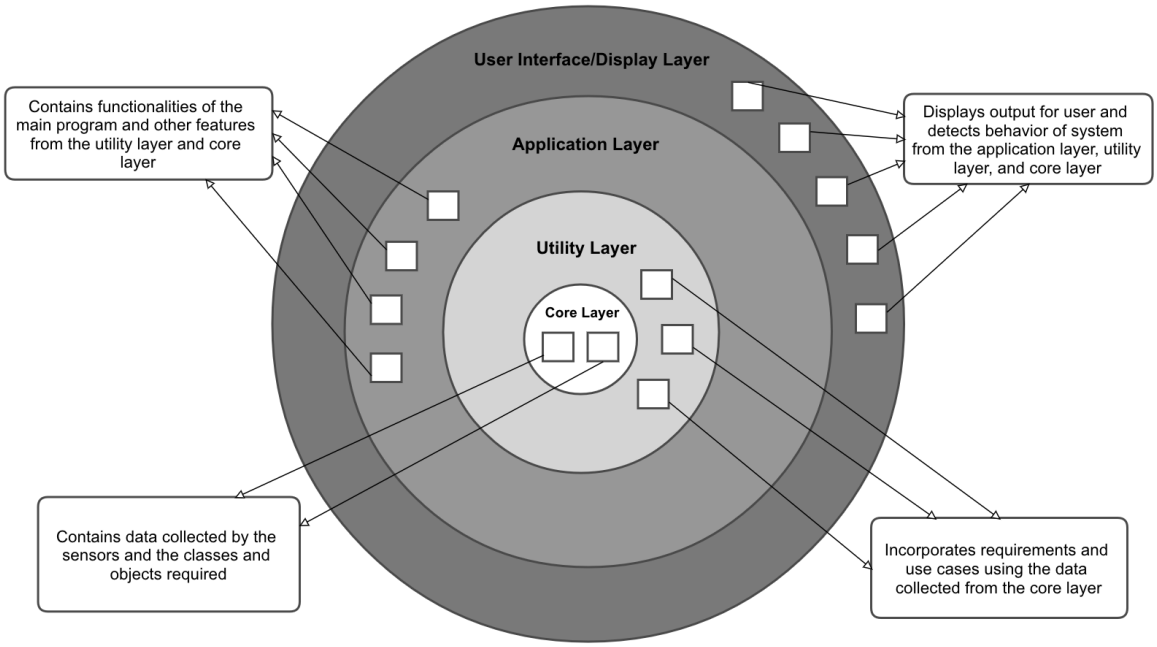
The pros of the data centered architecture include that all the components utilize the same data so the processes can pull the most updated data from the data repository, which is derived from sensor data. Another feature of the data centered architecture is that the components do not rely on each other and can therefore act independently. This system is also highly modular and allows for components to be interchanged very easily if changes need to be made to the IoT Engine. Some of the cons of this architecture is that the data needs to be held and maintained in a central data structure and all of the components rely on the accuracy of this data. If one sensor provides bad data, that could affect the entire system. The centralized data may also be difficult to work with as it is difficult to manage the data.

5.3.2 Call Return Architecture:



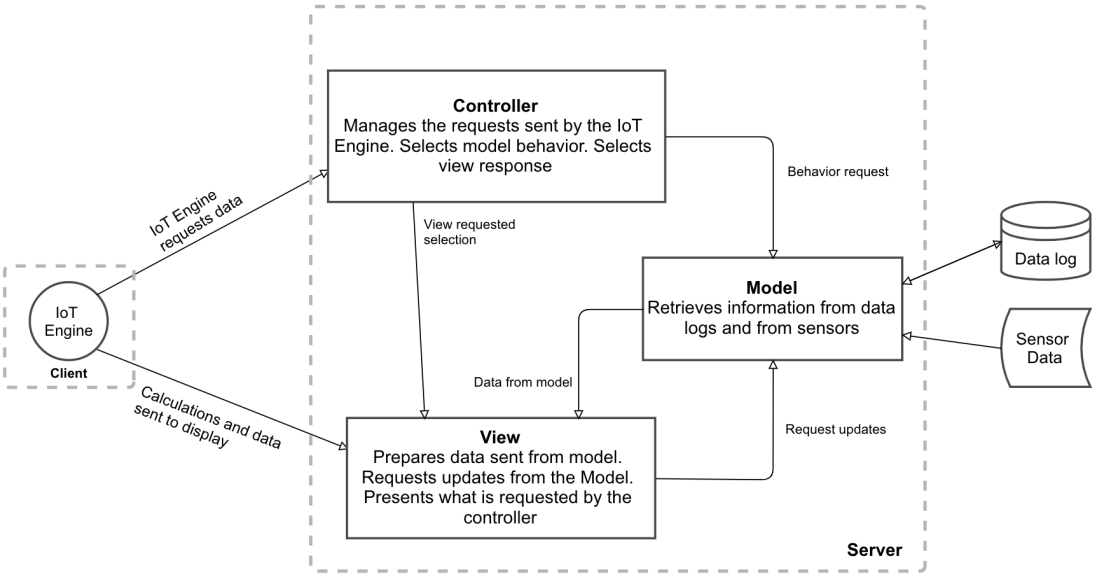
The pros of this architecture include: the fact that it can be very easily modified if changes need to be made to the IoT Engine; the architecture is straightforward and therefore simple to analyze for this project; the functionality can be grown simply by adding more boxes to the bottom; and as you go down in the hierarchy, your program gets more specific. The cons of this architecture include: multiple processes may be hard to depict in this diagram because of the complexity of the IoT Engine; there is a lack of detail within this diagram; and if this architecture is not implemented well, it can be difficult to add on to and create subprograms.

5.3.3 Layered Architecture:



The pros of this architecture include: each layer is separate, meaning the team would only need to consider each layer in its own scope; the problems of each layer are easy to manage for IoT; a change in one layer is independent of the others (in an ideal scenario) which allows for reworking if necessary; every layer can be tested on its own, so testing is optimized and simplified. The cons include: the model does not show the detailed interaction between each layer in the IoT system, nor does it show the temporal flow of data; the cost of management will increase if there are too many layers for IoT; the performance decreases as more layers get added; faults in abstraction in lower layers will disturb the flow of outer layers. The team saw that the temporal flow of data was the best way to show our software architecture due to the systematic interactions between the sensors and the IoT Engine. The Layered Architecture Model is unable to show the process of how real-time data is transferred, and was therefore not chosen.

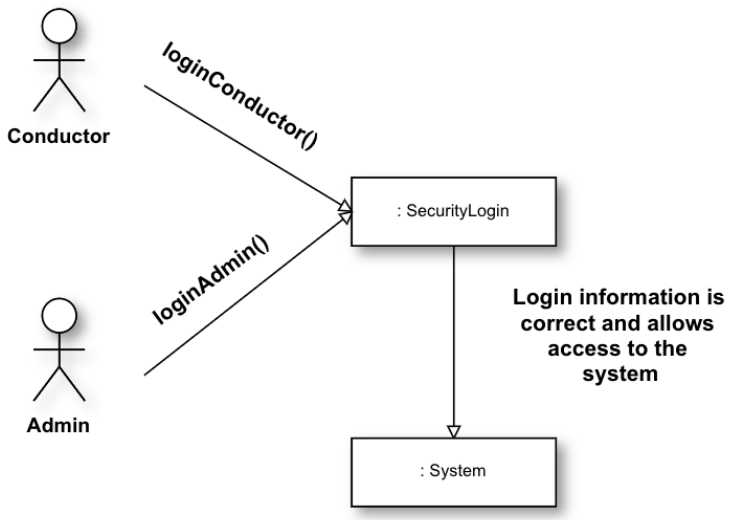
5.3.4 Model View Controller Architecture:



The pros of this architecture include: shows a logical flow of requests from the IoT Engine and proper data flow from data logs and sensor data; has an organized approach to how data flows in the software architecture; development of the application is quick and it is easy to update the application; the application is easy to debug because there are multiple levels in the application. The cons include: the model is better suited for the creation of web applications because it has more relevance to the flow of data from the databases to the user; the model is better suited for applications with extremely strict method rules.

5.4 Login Architecture Models

5.4.1 Object-Oriented Architecture:



This is a model of how an administrator or conductor will be able to log into the system, which is separate from the software architecture for this project. While we chose the data-flow architecture for the software architecture, we chose the object-oriented architecture for the login system because it allows us to display the system, the methods, and the actor involved in the project.

6. Project Code:

In this section, we will display the code used to complete this project. The team believed it would be most appropriate to code this project in python because of its robust nature and ability to easily create multiple classes.

6.1 IoT Engine

6.1.1 Variables and Constructors:

```
188  class IotEngine():
189
190      ##################################################################################################################
191      ##  IotEngine Class: Performs calculations with data collected from the sensors to display reccomendations to the conductor.
192      ##################################################################################################################
193
194      is_on = False
195      admin = False
196      conductor = False
197      is_logged_on = False
198
199      conductor_username = "conductor"
200      conductor_password = "password"
201      admin_username = "admin"
202      admin_password = "password"
203
204      ## Changable variables for Warnings
205      slippingtexts = "No Warnings"
206      distancetexts = "No Warnings"
207      gatestexts = "No Gates"
208
209      SwarningColor = "green"
210      DwanrningColor = "green"
211      GwarningColor = "green"
212
213      values = 0
214      iter_val = 0
```

6.1.2 Login Methods:

```
216     def loginAdmin(self):
217         """
218             Logs the administrator into the IoT Engine
219         """
220         print("************** ADMIN LOGGED IN **************")
221         time.sleep(2)
222         print("IoT Engine calibrating...\n")
223         time.sleep(1)
224         self.admin = True
225         self.is_logged_on = True
226         self.is_on = True
227 
228 
229     def loginConductor(self):
230         """
231             Logs the conductor into the IoT Engine
232         """
233         print("************** CONDUCTOR LOGGED IN **************")
234         time.sleep(2)
235         print("IoT Engine calibrating...\n")
236         time.sleep(1)
237         self.conductor = True
238         self.is_logged_on = True
239         self.is_on = True
240 
```

6.1.3 Calculation Methods:

```
242     def calculateDistance(self, dis_sensor):
243         """
244             Calculates the distance away from the closest object and sends
245             a notification or warning to the conductor
246         """
247         distance = dis_sensor.getDistance()
248         moving = dis_sensor.getMoving()
249 
250         if (distance > 1000):
251             self.distancetexts = "Object more than 1000 feet away. Do not change speed."
252             self.DwarningColor = "green"
253         elif (distance <= 1000 and moving == True):
254             self.distancetexts = "Moving object is " + str(distance) + " feet away. Brake to slow down."
255             self.DwarningColor = "red"
256         else:
257             self.distancetexts = "Stationary object is " + str(distance) + " feet away. Brake to slow down."
258             self.DwarningColor = "red"
259 
```

```
261     def calculateSlippage(self, humid_sensor, precip_sensor, wheel_sensor, gps_sensor):
262         """
263             Calculates the if there is slippage and sends a notification
264             or warning to the conductor
265         """
266         rain = precip_sensor.getRain()
267         snow = precip_sensor.getSnow()
268         wheelspeed = wheel_sensor.getWheelSpeed()
269         gpsspeed = gps_sensor.getGPSSpeed()
270         humidity = humid_sensor.getHumdidity()
271 
272         slipping = False
273         speed = wheelspeed * 33 * 3.14 * 60 / 63360
274 
275         if (abs(speed - gpsspeed) > 5):
276             slipping = True
277         if (humidity > 50):
278             slipping = True
279         if (rain == True):
280             slipping = True
281         if (snow == True):
282             slipping = True
283         if (slipping):
284             self.slippingtexts = "Slippage is occurring. Brake to slow down."
285             self.SwarningColor = "red"
286         else:
287             self.slippingtexts = "No slippage. Continue speed."
288             self.SwarningColor = "green"
```

```
276     def checkGates(self, dis_sensor, gps_sensor):
277         """
278             Checks if a gate is nearby and sends a notification or warning
279             to the conductor
280         """
281         distance = dis_sensor.getDistance()
282         gate = gps_sensor.getGate()
283 
284         horn = False
285         near = False
286         arrived = False
287 
288         if (1500 < distance < 1700):
289             horn = True
290         if (distance < 100):
291             arrived = True
292         if (100 < distance < 1500):
293             near = True
294         if (gate and horn):
295             self.gatestexts = "Gate is " + str(distance) + " meters away. Blow horn for 15 seconds."
296             self.GwarningColor = "orange"
297         elif (gate and arrived):
298             self.gatestexts = "Gate is " + str(distance) + " meters away. Blow horn for 5 seconds."
299             self.GwarningColor = "red"
300         elif (gate and near):
301             self.gatestexts = "Gate is " + str(distance) + " meters away."
302             self.GwarningColor = "orange"
303         else:
304             self.gatestexts = "No gates nearby. Continue speed."
305             self.GwarningColor = "green"
306 
```

6.1.4 Main Method:

```
311     def tsnr(self):
312         """
313             Grabs data from sensors (csv file) and initializes lists to hold data.
314             Allows admin to view logs
315             Allows conductor to start a new trip
316         """
317 
318         print("************** WELCOME **************")
319         print("Loading all processes...\n")
320         time.sleep(2)
321         print("Connecting to sensors...\n")
322         time.sleep(2)
323 
324         try:
325             cwd = os.getcwd() + '/sensordata.csv'
326             df = pd.read_csv(cwd)
327             df = df.dropna()
328 
329         except:
330             self.is_on = False
331             print ("Unable to connect to sensor data")
332             sys.exit()
333 
334         dist_list = []
335         moving_list = []
336         wheel_list = []
337         humid_list = []
338         precip_list = []
339         temp_list = []
340         gps_list = []
341         gate_list = []
342         connect_list = []
343 
```

```
344         for i in range(len(df)):
345             dist_list.append(df["Object Distance"][i])
346             moving_list.append(df["Moving"][i])
347             wheel_list.append(df["Wheel Speed"][i])
348             humid_list.append(df["Humidity"][i])
349             precip_list.append(df["Precipitation Level"][i])
350             temp_list.append(df["Temperature"][i])
351             gps_list.append(df["GPS Speed"][i])
352             gate_list.append(df["Gate"][i])
353             connect_list.append(df["Connectivity"][i])
354 
355         invalid_login = True
356         login_count = 0
357 
358         while (invalid_login and login_count < 8):
359             username = input("Enter a username: ")
360             password = input("Enter a password: ")
361             if (username == self.conductor_username and password == self.conductor_password):
362                 self.loginConductor()
363                 invalid_login = False
364             if (username == self.admin_username and password == self.admin_password):
365                 self.loginAdmin()
366                 invalid_login = False
367             if (invalid_login == True):
368                 login_count = login_count + 1
369                 print("Invalid Login. Try Again.")
370 
371         if (login_count >= 8):
372             print("Too many attempts to login, please contact LCS for manual override.")
373 
374         if (self.admin):
375             old_logs = {}
376             now = datetime.datetime.now()
377             infolog = InformationLog(now, **old_logs)
378             infolog.getCurrentLog()
379 
```

```
    ## GUI Items
    window = tk.Tk()

    window.title("IoT HTR")
    window.geometry('900x380')
    window.config(bg="gray")

    titlefont = fonts.Font(family="Univers", size=36, weight="bold")
    standardfont = fonts.Font(family="Univers", size=24)
    labelfont = fonts.Font(family="Univers", size=20)

    title = Label(text="IoT Display", fg="white", bg="gray", font=titlefont)

    slippingtext = Label(text="Slippage", fg="white", bg="gray", font=labelfont)
    slippingcontainer = LabelFrame(labelwidget=slippingtext, labelanchor='n', fg="gray", bg="gray", padx=20, pady=10)

    distancetext = Label(text="Distance", fg="white", bg="gray", font=labelfont)
    distancecontainer = LabelFrame(labelwidget=distancetext, labelanchor='n', fg="gray", bg="gray", padx=20, pady=10)

    gatetext = Label(text="Gates", fg="white", bg="gray", font=labelfont)
    gatecontainer = LabelFrame(labelwidget=gatetext, labelanchor='n', fg="gray", bg="gray", padx=20, pady=10)

    slippinglabel = Label(slippingcontainer, text=self.slippingtexts, fg="white", bg=self.SwarningColor, font=standardfont, justify=tk.LEFT)
    objectlabel = Label(distancecontainer, text=self.distancetexts, fg="white", bg=self.DwanrningColor, font=standardfont, justify=tk.LEFT)
    gatelabel = Label(gatecontainer, text=self.gatestexts, fg="white", bg=self.GwarningColor, font=standardfont, justify=tk.LEFT)

    title.pack()
    slippingcontainer.pack()
    slippinglabel.pack()
    distancecontainer.pack()
    objectlabel.pack()
    gatecontainer.pack()
    gatelabel.pack()
```

```
    if (self.conductor):
        self.values = len(df)
        self.iter_val = 0
        old_logs = {}
        def loop():

            if (self.values == 0):
                window.destroy()
                return

            precipsensor = PrecipitationSensor(precip_list[self.iter_val],
                                               temp_list[self.iter_val],
                                               False,
                                               False)
            dissensor = DistanceSensor(dist_list[self.iter_val],
                                       moving_list[self.iter_val])
            humidsensor = HumiditySensor(humid_list[self.iter_val])
            wheelsensor = WheelSensor(wheel_list[self.iter_val])
            gpssensor = GPSSensor(gps_list[self.iter_val], True, gate_list[self.iter_val])
            connect = connect_list[self.iter_val]

            if(connect == 1):
                print("Connection has been restored to WiFi/Cellular.")
                print("IoT Engine no longer needed.")
                print("Redirecting to normal system...\n")
                sys.exit()


            self.calculateDistance(dissensor)
            self.calculateSlippage(humidsensor, precipsensor, wheelsensor, gpssensor)
            self.checkGates(dissensor, gpssensor)

            slippinglabel.config(text=self.slippingtexts, bg=self.SwarningColor)
            objectlabel.config(text=self.distancetexts, bg=self.DwarningColor)
            gatelabel.config(text=self.gatestexts, bg=self.GwarningColor)
            window.after(5000, loop)
```

```
451
452                     now = datetime.datetime.now()
453                     infolog = InformationLog(now, **old_logs)
454                     infolog.createNewLog(now, humidsensor, precipsensor, wheelsensor, gpssensor, dissensor)
455
456                     self.values = self.values - 1
457                     self.iter_val = self.iter_val + 1
458
459                 loop()
460                 window.mainloop()
461                 print("Thank you for riding\n")
462                 time.sleep(3)
463
464             self.is_on = False
465
466     iot = IotEngine()
467     iot.tsnr()
```

6.2 Information Log

6.2.1 Methods, Variables, Constructors:

```
10  class InformationLog:
11      ##############################################################################################################
12      ##  Information Log Class: Creates and stores information logs with information about the train trip
13      ##############################################################################################################
14
15
16      def __init__(self, entry, **saved):
17          """
18              InformationLog constructor
19                  entry:      list entry to be added to the log
20                  saved:      dictionary where logs are saved
21          """
22          self.entry = entry
23          self.saved = saved
24
25      def getCurrentLog(self):
26          """
27              Gets the current log for the trip
28          """
29          print("Todays log: ", self.saved)
30          return self.saved.setdefault(datetime.datetime.now())
31
32
33      def createNewLog(self, d, humid_sensor, precip_sensor, wheel_sensor, gps_sensor, dis_sensor):
34          """
35              Creates a new log for the trip by getting current data from sensors
36          """
37          rain = precip_sensor.getRain()
38          snow = precip_sensor.getSnow()
39          distance = dis_sensor.getDistance()
40          humidity = humid_sensor.getHumdidity()
41          wheelspeed = wheel_sensor.getWheelSpeed()
42          speed = gps_sensor.getGPSSpeed()
43
44          self.saved.update({d: [ "Rain Data: " + str(rain),
45                                  "Snow Data: " + str(snow),
46                                  "Distance Data: " + str(distance),
47                                  "Humidity Data: " + str(humidity),
48                                  "Wheel Speed Data: " + str(wheelspeed),
49                                  "GPS Speed Data: " + str(speed) ]
50                             })
```

6.3 Distance Sensor

6.3.1 Methods, Variables, Constructors:

```
168  class DistanceSensor:
169      ##########################################################################################################
170      ##  Distance Sensor Class: Collects information from the distance sensor
171      ##########################################################################################################
172
173      def __init__(self, distance, moving):
174          """
175              DistanceSensor constructor
176                  distance:   int distance away from closest object detected by sensor
177                  moving:     boolean to check whether an object is moving or stationary
178          """
179          self.distance = distance
180          self.moving = moving
181
182      def getDistance(self):
183          return self.distance
184
185      def getMoving(self):
186          return self.moving
187
```

6.4 GPS Sensor

6.4.1 Methods, Variables, Constructors:

```
72   class GPSSensor:
73       ##########################################################################################################
74       ##  GPS Sensor Class: Collects information from the GPS sensor
75       ##########################################################################################################
76
77       def __init__(self, GPS_speed, GPS_connection, gate):
78           """
79               GPSSensor constructor
80                   GPS_speed:      int GPS speed level detected by sensor
81                   GPS_connection: bool connection detected by sensor
82                   gate:           bool gate checker that detects locations of gates
83           """
84           self.GPS_speed = GPS_speed
85           self.gate = gate
86           self.GPS_connection = GPS_connection
87
88       def getGate(self):
89           """
90               Checks whether there's a gate in the path of the train
91           """
92           return self.gate
93
94       def getGPSSpeed(self):
95           """
96               Gets the GPS speed from the GPS speed sensor
97           """
98           if (self.GPS_connection == False):
99               return False
100          if (self.GPS_connection == True):
101              return self.GPS_speed
102
```

6.5 Humidity Sensor

6.5.1 Methods, Variables, Constructors:

```
53  class HumiditySensor:
54      ##############################################################################################################
55      ##  Humidity Sensor Class: Collects information from the humidity sensor
56      ##############################################################################################################
57
58      def __init__(self, humidty):
59          """
60              HumiditySensor constructor
61                  humidty:    int humidity level detected by sensor
62          """
63          self.humidty = humidty
64
65      def getHumdidity(self):
66          """
67              Gets humidity level from the humidity sensor
68          """
69          return self.humidty
70
```

6.6 Precipitation Sensor

6.6.1 Methods, Variables, Constructors:

```
122 class PrecipitationSensor:
123     ##############################################################################################################
124     ##  Precipitation Sensor Class: Collects information from the precipitation sensor
125     ##############################################################################################################
126
127     def __init__(self, precip_level, temperature, rain, snow):
128         """
129             PrecipitationSensor constructor
130                 precip_level:   int precipitation level detected by sensor
131                 temperature:    int temperature detected by sensor
132                 rain:           bool rain checker
133                 snow:           bool snow checker
134         """
135         self.precip_level = precip_level
136         self.temperature = temperature
137         self.rain = False
138         self.snow = False
139
140     def getWeather(self):
141         """
142             Checks whether there is rain or snow when precipitation is detected
143         """
144         if (self.precip_level > 4 and self.temperature < 32):
145             self.snow = True
146             self.rain = False
147         elif (self.precip_level > 4 and self.temperature >= 32):
148             self.snow = False
149             self.rain = True
150         else:
151             self.snow = False
152             self.rain = False
153
154     def getRain(self):
155         """
156             Checks if there is rain from the precipitation sensor
157         """
158         PrecipitationSensor.getWeather(self)
159         return self.rain
160
161     def getSnow(self):
162         """
163             Checks if there is snow from the precipitation sensor
164         """
165         PrecipitationSensor.getWeather(self)
166         return self.snow
167
```

6.7 Wheel Sensor

6.7.1 Methods, Variables, Constructors:

```
104  class WheelSensor:
105      ##########################################################################################################
106      ##   Wheel Sensor Class: Collects information from the wheel sensor
107      ##########################################################################################################
108
109      def __init__(self, rotations_per_min):
110          """
111              WheelSensor constructor
112                  rotations_per_min:      int rotations per minute detected by sensor
113          """
114          self.rotations_per_min = rotations_per_min
115
116      def getWheelSpeed(self):
117          """
118              Gets the wheel speed from the wheel sensor
119          """
120          return self.rotations_per_min
```

6.8 Import

6.8.1 Import Statements:

```
1    import time
2    import sys
3    import tkinter as tk
4    import datetime
5    import pandas as pd
6    import os
7    import tkinter.font as fonts
8    from tkinter import *
```

7. Testing:

In this section, we will list all the test cases per requirement and have them numbered. We will ensure that all our requirements using their number are met (passed).

7.1 Validation Testing

Requirement	Description	Test
R1	The sensors purchased must be waterproof	The company supplying the sensors for this project have specified that all sensors are waterproof.
R2	The sensors must be able to last 1000 hours and operate 99.99% of the time to be approved	The company supplying the sensors for this project have specified that all sensors can last at least 1000 hours and will be operational 99.99% of the time.
R3	The LoRaWan protocol is the network that will support this project.	The LoRaWan protocol is the network that is used to support the IoT Engine when the train does not have access to WiFi/Cellular.
R4	The sensor will be able to process an event on time and give a note to the conductor within 0.1 seconds.	The sensor purchased is able to process an event and give note to the conductor within 0.1 seconds. Despite this capability, the team has reevaluated and seen that it is fair for the sensors to send data to the TSNR to be processed every 5 seconds.
R5	The train and IoT engine will process the event within 0.5 seconds of the occurrence to enable proper speed of transmission of a message from the distance sensor to the train.	The train and IoT engine have the capability to process an event within 0.5 seconds, or any other amount of time the conductor wishes. High speed of transmission between the distance sensor and the IoT Engine exist.
R6	The hardware must be failure free for 1000 hours,	The company supplying the sensors for this project have specified that all sensors can last

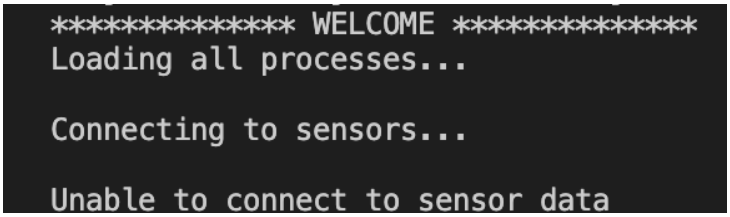
	and have a high probability of lasting this long.	at least 1000 hours and have a high probability of lasting this long.
R7	The reliability of hardware should be no less than 99.99%.	The company supplying the sensors for this project have specified that all sensors have a high reliability of 99.99%.
R8	The system will require a secure username and password to access or operate it from both employees and administrators.	A secure username and password is required to access the IoT Engine. The IoT Engine supplies a different password for the admin and the conductor. If the username or password is incorrect, the IoT Engine prompts the user to try logging in again.
R9	The system will feature a manual override option in the event of tampering or faulty sensor data.	If the user tries to login more than 8 times, then the program exits and an LCS manager will need to be contacted for further assistance and a manual override.
R10	The data transfer from the sensors to the IoT engine will occur locally to prevent outside attacks.	The data transfer from sensors to the IoT Engine occurs locally without the use of a third-party network or server.
R11	Three distance sensors placed in the front of the train.	The train contains three distance sensors placed in the front of the train, which all collect distance data and report back to the IoT Engine.
R12	The distance sensors will send information to IoT, which will recommend to the operator: braking, increasing speed, decreasing speed, or stopping the train.	The conductor gets recommendations from the IoT Engine depending on how far away an object is from the train.

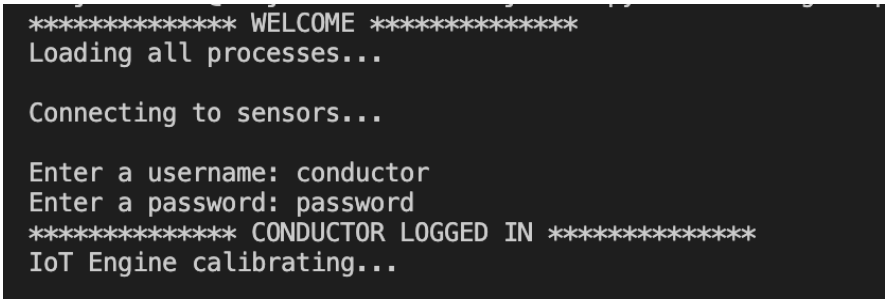
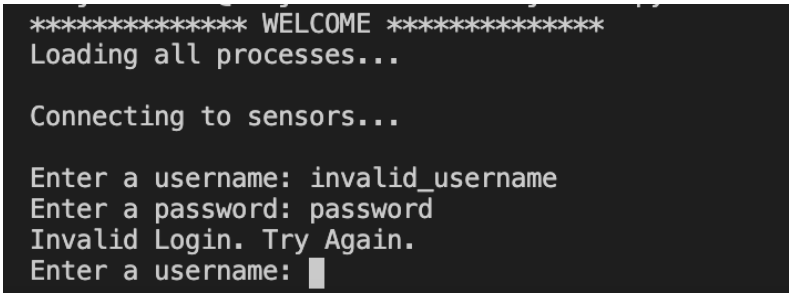
R13	The distance sensors will provide information on how close an object is in five-second increments.	The distance sensors report back to the IoT Engine in 5 second increments.
R14	The distance sensors will gather data about the speed of the train from the GPS.	The GPS sensor provides the distance sensor with information about the speed of the train.
R15	The distance sensors will use both the speed and distance data to make decisions about what to tell the conductor.	The distance sensor uses data from itself and GPS sensor to make a decision and recommend a course of action for the conductor.
R16	If the sensor detects that action is required, it will display the information on the console.	When action is required, the IoT Engine displays the data to the conductor through a GUI display screen
R17	Three speed sensors each on the front and the back of the train.	The train contains three distance sensors placed in the back of the train, which all collect distance data and report back to the IoT Engine.
R18	The speed sensors will provide information about how fast moving objects are going, which will be resolved into a whole number.	The speed sensor calculates the speed of the train itself and the speed of other moving objects around the train.
R19	The distance sensors should alert the speed sensors and give one of the following commands to the operator: brake, increase	The distance sensor uses data from itself and GPS sensor to make a decision and recommend a course of action for the conductor.

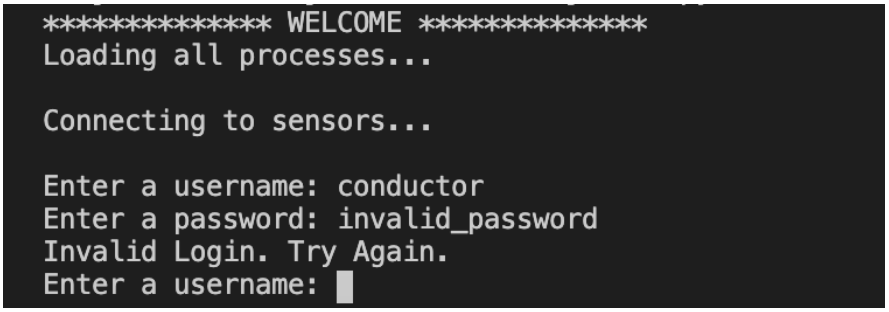
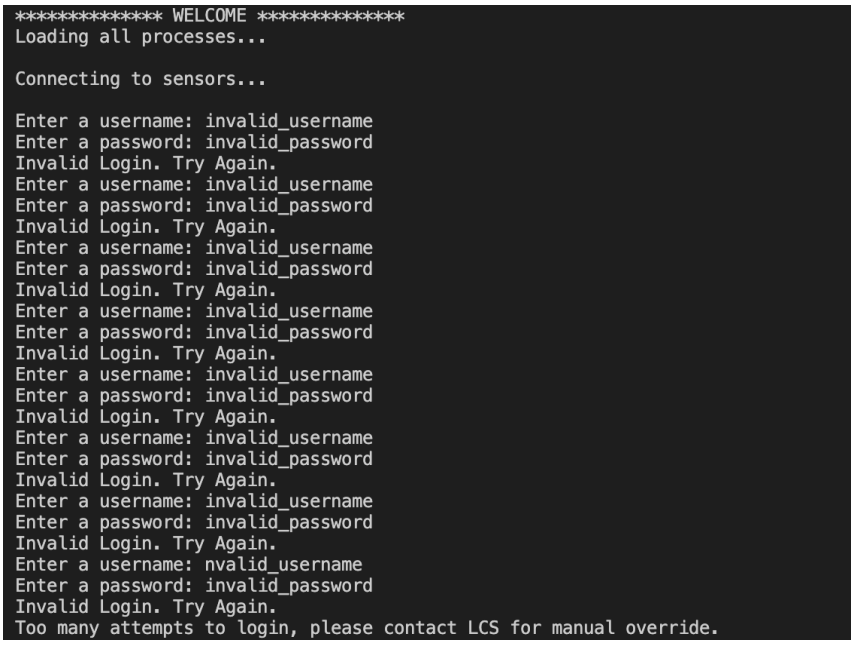
	speed,decrease speed, or stop the train.	
R20	The console will display recommended action if it is required.	When action is required, the IoT Engine displays the data to the conductor through a GUI display screen.
R21	A GPS sensor will detect the visual of a gate crossing coming soon on the tracks.	The GPS sensor is programmed to know where the gates are. When the train approaches a gate, it will alert the conductor.
R22	A camera will show the visibility of the gate crossing from 1000 miles away.	A camera is attached to the outside of the train to show the conductor the gate crossing when it is 1000 miles away.
R23	A distance sensor will recognize the signal and bars from 1000 miles away and notify the operator of what action to take: braking, increasing speed, decreasing speed, or stopping the train.	The distance sensor is also used to recognize that there is a gate and it alerts the conductor if the gate is less than 1000 miles away. The IoT Engine will recommend a course of action for the conductor.
R24	The gate crossing will send a signal to the IoT engine 60 seconds before the train arrives.	The GPS sensor detects the signal from the gate crossing and alerts the IoT engine 60 seconds before the train arrives at the location of the crossing.
R25	The conductor will be alerted to sound the horn for 15 seconds when the gate is 1500 to 1700 meters away. The conductor will then blow the horn for 5 seconds when the gate is less than 100 meters away.	The conductor is alerted of gates and is alerted to sound their horn for a given amount of time when the train approaches the gate.

R26	The wheel sensors will record the RPM of the train.	The wheel sensors retrieve the rotations per minute of the train to calculate the speed of the train
R27	The IoT will store the theoretical RPM based on the wheel size and the speed received from GPS data.	The IoT will use the wheel sensor RPM and compare it to the train speed from GPS data and sensors.
R28	The IoT will make a recommendation to the operator based on any discrepancies between the RPM from the sensor and the RPM from the GPS data.	If there is a large discrepancy between the RPM speed and the GPS sensor speed, the IoT Engine will assume there is slippage occurring and it will alert the conductor to take action as well as recommend a course of action.
R29	The console will display the recommendation to the operator on the screen so the operator can act on the recommendation.	When action is required, the IoT Engine displays the data to the conductor through a GUI display screen.

7.2 Scenario-Based Testing

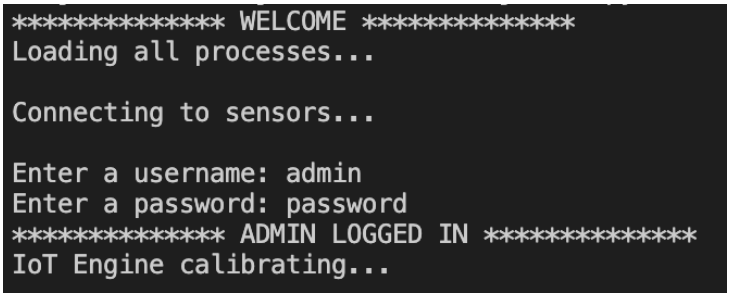
Use Case	Use Case Description	Tests
1	Activate the IoT Engine	**Test Description:** The function *tsnr()* is activated when the IoT Engine is turned on. When the program is started, the conductor or administrator is greeted with a welcome screen. The IoT Engine takes time to load the processes and connect to the sensors. **Test 1: Connection is established** If the IoT Engine is able to establish a connection to the sensors, the user will see the following message:  The IoT Engine was able to establish a connection to the sensors, so the user is prompted to enter their username. A password prompt is shown after the user enters their username. In this case, the connection is simulated by a csv file with information. **Test 2: Connection is not established** If the IoT Engine is unable to establish a connection to the sensors, the user will see the following message:  The IoT Engine tells the conductor that a connection was unable to be made and the program is exited. The user can reactivate the program if they wish to do so. In this case, the connection is simulated by a csv file without information that is empty.

2	Conductor logs into IoT engine	**<u>Test Description:</u>** The function *tsnr()* is activated when the IoT Engine is turned on. When the program is started, the conductor is prompted to enter their login information. **<u>Test 1: Conductor successfully logs in</u>** The conductor will attempt to login using the correct credentials  Since the username and password are correct, the conductor will be logged into the system. The IoT Display will begin to show warnings/notifications to the conductor. **<u>Test 2: Conductor supplies incorrect username</u>** The conductor will attempt to login using the wrong username.  Since the username is incorrect, the conductor will be told that their login is invalid, and will be given another chance to login to the system. **<u>Test 3: Conductor supplies incorrect password</u>** The conductor will attempt to login using the wrong password.

		 Since the password is incorrect, the conductor will be told that their login is invalid, and will be given another chance to login to the system. **Test 4: Conductor supplies incorrect credentials many times** The conductor will attempt to login using the wrong username/password multiple times.  Since the username/password is incorrect, the conductor will be told that their login is invalid. After seven attempts, they will be locked out of the system and be asked to contact LCS for further instruction.
3	Administrator logs into IoT engine	**Test Description:** The function *tsnr()* is activated when the IoT Engine is turned on. When the program is started, the administrator is prompted to enter their login information.

Test 1: Administrator successfully logs in

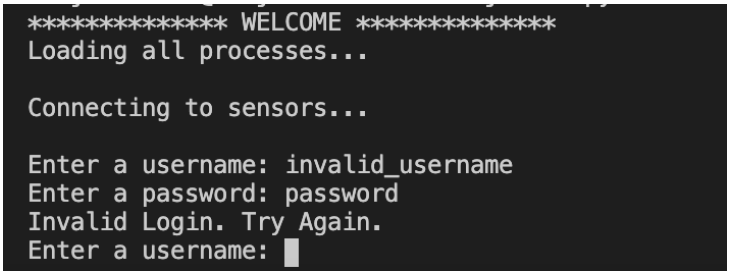
The administrator will attempt to login using the correct credentials



Since the username and password are correct, the administrator will be logged into the system.

Test 2: Administrator supplies incorrect username

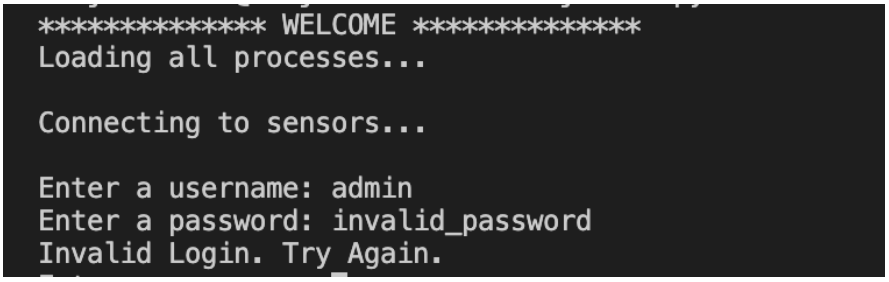
The administrator will attempt to login using the wrong username.



Since the username is incorrect, the administrator will be told that their login is invalid, and will be given another chance to login to the system.

Test 3: Administrator supplies incorrect password

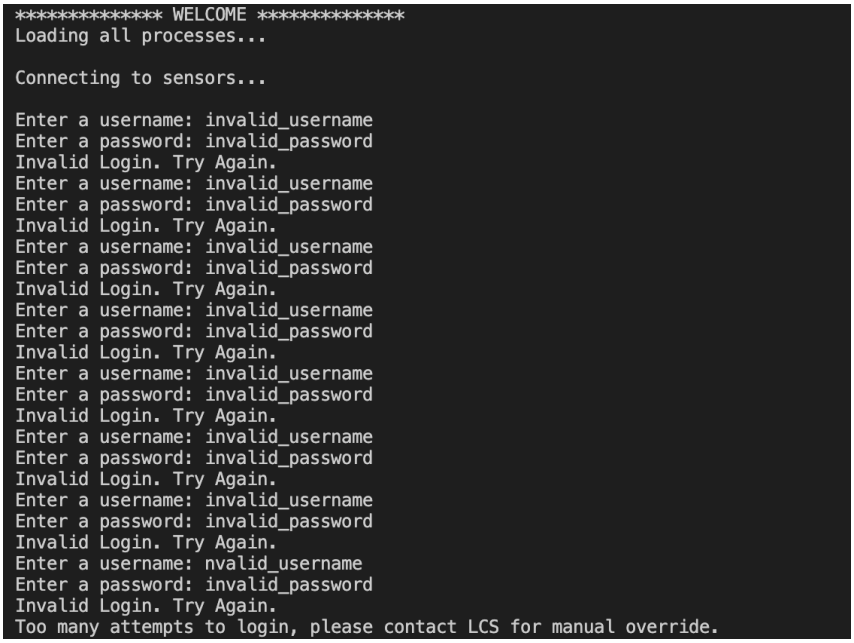
The administrator will attempt to login using the wrong password.



Since the password is incorrect, the administrator will be told that their login is invalid, and will be given another chance to login to the system.

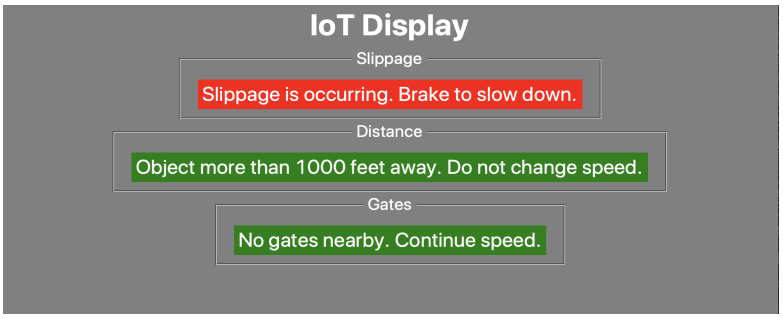
Test 4: Administrator supplies incorrect credentials many times

The administrator will attempt to login using the wrong

		username/password multiple times.  Since the username/password is incorrect, the administrator will be told that their login is invalid. After seven attempts, they will be locked out of the system and be asked to contact LCS for further instruction.
4	Detection of Hazardous Weather Conditions	**<u>Test Description:</u>** The function *checkslippage()* is used to detect slippage on the tracks. If there is slippage from the Humidity or Precipitation Sensor, then the conductor will be alerted. **<u>Test 1: Humidity Sensors detect slippage</u>** Humidity sensors detect that the humidity level is greater than 50 through *checkslippage()*, meaning there is high amounts of precipitation on the tracks. All other hazardous conditions are not present. The state of the data shows that only humidity is detecting an error because the humidity levels are greater than 50.

Object Distance	Wheel Speed	Humidity	Precipitation Level	Temperature	GPS Speed	Gate
2000	999	60	0	40	999	FALSE
2000	999	60	0	40	999	FALSE
2000	999	60	0	40	999	FALSE
2000	999	60	0	40	999	FALSE
2000	999	60	0	40	999	FALSE
2000	999	60	0	40	999	FALSE
2000	999	60	0	40	999	FALSE
2000	999	60	0	40	999	FALSE
2000	999	60	0	40	999	FALSE
2000	999	60	0	40	999	FALSE
2000	999	60	0	40	999	FALSE
2000	999	60	0	40	999	FALSE
2000	999	60	0	40	999	FALSE
2000	999	60	0	40	999	FALSE
2000	999	60	0	40	999	FALSE
2000	999	60	0	40	999	FALSE
2000	999	60	0	40	999	FALSE
2000	999	60	0	40	999	FALSE
2000	999	60	0	40	999	FALSE
2000	999	60	0	40	999	FALSE
2000	999	60	0	40	999	FALSE

Therefore, an error is displayed on the display screen.



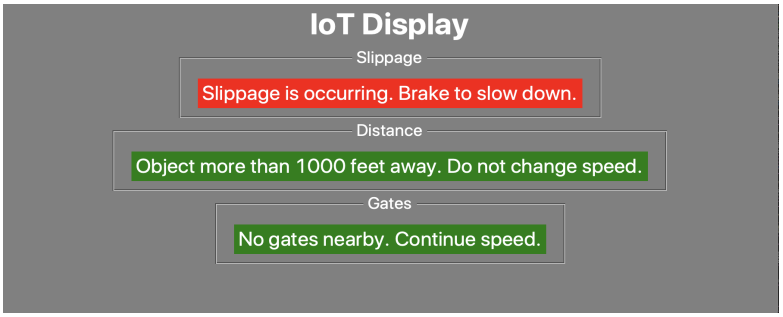
Since the IoT Engine has detected a humidity level greater than 50, the IoT Engine is displaying a slippage error and prompting the conductor to brake to slow the train down.

<u>Test 2: Precipitation Sensors detect slippage due to rain</u>

Precipitation sensors detect that the precipitation level is greater than 4 through *getWeather()*, meaning there is high amounts of precipitation on the tracks. Precipitation sensors also detect that the temperature is greater than 32 degrees, meaning that there is rain on the tracks. All other hazardous conditions are not present. The state of the data shows that only humidity is detecting an error because the precipitation levels are greater than 4 and temperature is greater than 32 degrees.

Object Distance	Wheel Speed	Humidity	Precipitation Level	Temperature	GPS Speed	Gate
2000	999	40	5	40	999	FALSE
2000	999	40	5	40	999	FALSE
2000	999	40	5	40	999	FALSE
2000	999	40	5	40	999	FALSE
2000	999	40	5	40	999	FALSE
2000	999	40	5	40	999	FALSE
2000	999	40	5	40	999	FALSE
2000	999	40	5	40	999	FALSE
2000	999	40	5	40	999	FALSE
2000	999	40	5	40	999	FALSE
2000	999	40	5	40	999	FALSE
2000	999	40	5	40	999	FALSE
2000	999	40	5	40	999	FALSE
2000	999	40	5	40	999	FALSE
2000	999	40	5	40	999	FALSE
2000	999	40	5	40	999	FALSE
2000	999	40	5	40	999	FALSE
2000	999	40	5	40	999	FALSE
2000	999	40	5	40	999	FALSE
2000	999	40	5	40	999	FALSE
2000	999	40	5	40	999	FALSE

Therefore, an error is displayed on the display screen.



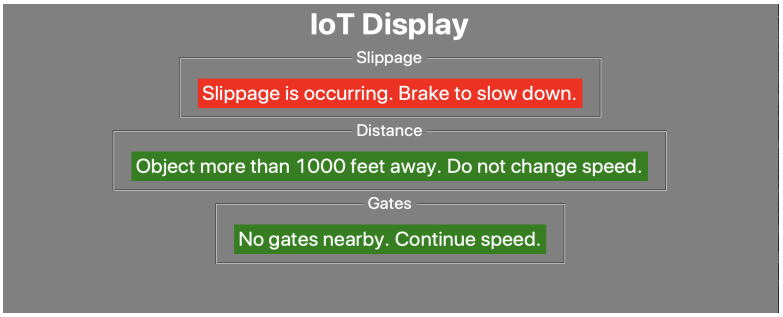
Since the IoT Engine has detected a precipitation level greater than 4 and a temperature greater than 32 degrees, the IoT Engine is displaying a slippage error and prompting the conductor to brake to slow the train down.

Test 3: Precipitation Sensors detect slippage due to snow

Precipitation sensors detect that the precipitation level is greater than 4 through *getWeather()*, meaning there is high amounts of precipitation on the tracks. Precipitation sensors also detect that the temperature is less than 32 degrees, meaning that there is snow on the tracks. All other hazardous conditions are not present. The state of the data shows that only humidity is detecting an error because the precipitation levels are greater than 4 and temperature is less than 32 degrees.

Object Distance	Wheel Speed	Humidity	Precipitation Level	Temperature	GPS Speed	Gate
2000	999	40	5	20	999	FALSE
2000	999	40	5	20	999	FALSE
2000	999	40	5	20	999	FALSE
2000	999	40	5	20	999	FALSE
2000	999	40	5	20	999	FALSE
2000	999	40	5	20	999	FALSE
2000	999	40	5	20	999	FALSE
2000	999	40	5	20	999	FALSE
2000	999	40	5	20	999	FALSE
2000	999	40	5	20	999	FALSE
2000	999	40	5	20	999	FALSE
2000	999	40	5	20	999	FALSE
2000	999	40	5	20	999	FALSE
2000	999	40	5	20	999	FALSE
2000	999	40	5	20	999	FALSE
2000	999	40	5	20	999	FALSE
2000	999	40	5	20	999	FALSE
2000	999	40	5	20	999	FALSE
2000	999	40	5	20	999	FALSE
2000	999	40	5	20	999	FALSE

Therefore, an error is displayed on the display screen.



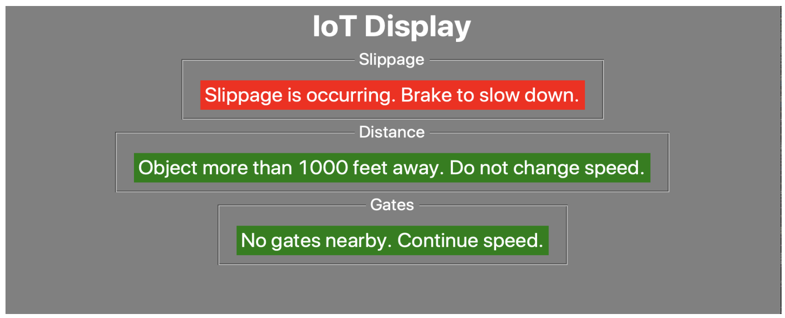
Since the IoT Engine has detected a precipitation level greater than 4 and a temperature less than 32 degrees, the IoT Engine is displaying a slippage error and prompting the conductor to brake to slow the train down.

Test 4: Humidity Sensors detect slippage and Precipitation Sensors detect slippage

Humidity sensors detect that the humidity level is greater than 50 through *checkslippage()*, meaning there is high amounts of precipitation on the tracks. Precipitation sensors detect that the precipitation level is greater than 4 through *getWeather()*, meaning there is high amounts of precipitation on the tracks. Precipitation sensors also detect that the temperature is less than 32 degrees in some cases and greater than 32 degrees in some cases, meaning that there is rain or snow on the tracks. The state of the data shows slippage from humidity sensors and precipitation sensors.

Object Distance	Wheel Speed	Humidity	Precipitation Level	Temperature	GPS Speed	Gate
2000	999	60	5	20	999	FALSE
2000	999	60	5	40	999	FALSE
2000	999	60	5	20	999	FALSE
2000	999	60	5	20	999	FALSE
2000	999	60	5	20	999	FALSE
2000	999	60	5	20	999	FALSE
2000	999	60	5	20	999	FALSE
2000	999	60	5	20	999	FALSE
2000	999	60	5	20	999	FALSE
2000	999	60	5	20	999	FALSE
2000	999	60	5	20	999	FALSE
2000	999	60	5	20	999	FALSE
2000	999	60	5	20	999	FALSE
2000	999	60	5	20	999	FALSE
2000	999	60	5	20	999	FALSE
2000	999	60	5	20	999	FALSE
2000	999	60	5	20	999	FALSE
2000	999	60	5	20	999	FALSE
2000	999	60	5	20	999	FALSE
2000	999	60	5	20	999	FALSE

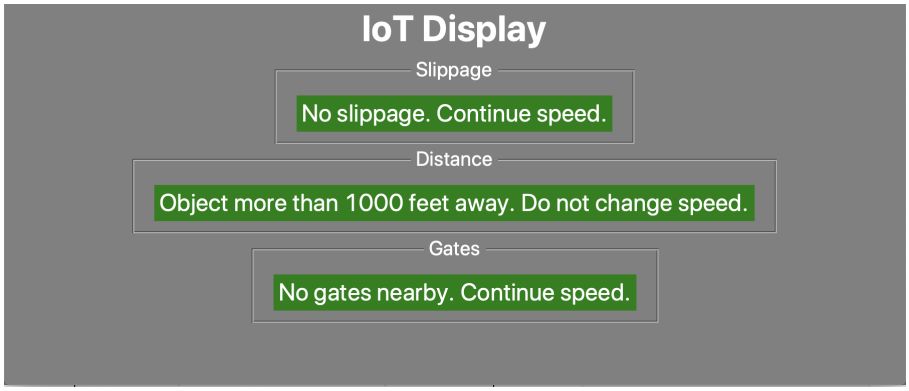
Therefore, an error is displayed on the display screen.



Since the IoT Engine has detected a humidity level greater than 50, and rain/snow on the tracks, the IoT Engine is displaying a slippage error and prompting the conductor to brake to slow the train down.

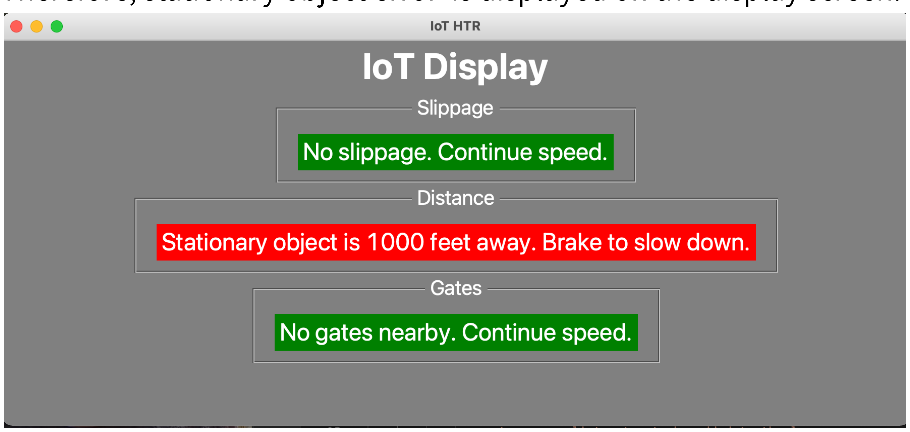
Test 5: No Slippage is detected from Humidity and Precipitation Sensors

Humidity sensors detect that the humidity level is less than 50 through *checkslippage()*, meaning there are low amounts of precipitation on the tracks. Precipitation sensors detect that the precipitation level is less than 4 through *getWeather()*, meaning there is low amounts of precipitation on the tracks. Precipitation sensors also detect that the temperature is greater than 32 degrees, meaning that there is no snow on the tracks. All other hazardous conditions are not present. The state of the data shows no slippage

		<table><tr><th>Object Distance</th><th>Wheel Speed</th><th>Humidity</th><th>Precipitation Level</th><th>Temperature</th><th>GPS Speed</th><th>Gate</th></tr><tr><td>2000</td><td>100</td><td>40</td><td>0</td><td>40</td><td>10</td><td>FALSE</td></tr><tr><td>2000</td><td>100</td><td>40</td><td>0</td><td>40</td><td>10</td><td>FALSE</td></tr><tr><td>2000</td><td>100</td><td>40</td><td>0</td><td>40</td><td>10</td><td>FALSE</td></tr><tr><td>2000</td><td>100</td><td>40</td><td>0</td><td>40</td><td>10</td><td>FALSE</td></tr><tr><td>2000</td><td>100</td><td>40</td><td>0</td><td>40</td><td>10</td><td>FALSE</td></tr><tr><td>2000</td><td>100</td><td>40</td><td>0</td><td>40</td><td>10</td><td>FALSE</td></tr><tr><td>2000</td><td>100</td><td>40</td><td>0</td><td>40</td><td>10</td><td>FALSE</td></tr><tr><td>2000</td><td>100</td><td>40</td><td>0</td><td>40</td><td>10</td><td>FALSE</td></tr><tr><td>2000</td><td>100</td><td>40</td><td>0</td><td>40</td><td>10</td><td>FALSE</td></tr><tr><td>2000</td><td>100</td><td>40</td><td>0</td><td>40</td><td>10</td><td>FALSE</td></tr><tr><td>2000</td><td>100</td><td>40</td><td>0</td><td>40</td><td>10</td><td>FALSE</td></tr><tr><td>2000</td><td>100</td><td>40</td><td>0</td><td>40</td><td>10</td><td>FALSE</td></tr><tr><td>2000</td><td>100</td><td>40</td><td>0</td><td>40</td><td>10</td><td>FALSE</td></tr><tr><td>2000</td><td>100</td><td>40</td><td>0</td><td>40</td><td>10</td><td>FALSE</td></tr><tr><td>2000</td><td>100</td><td>40</td><td>0</td><td>40</td><td>10</td><td>FALSE</td></tr><tr><td>2000</td><td>100</td><td>40</td><td>0</td><td>40</td><td>10</td><td>FALSE</td></tr><tr><td>2000</td><td>100</td><td>40</td><td>0</td><td>40</td><td>10</td><td>FALSE</td></tr><tr><td>2000</td><td>100</td><td>40</td><td>0</td><td>40</td><td>10</td><td>FALSE</td></tr><tr><td>2000</td><td>100</td><td>40</td><td>0</td><td>40</td><td>10</td><td>FALSE</td></tr><tr><td>2000</td><td>100</td><td>40</td><td>0</td><td>40</td><td>10</td><td>FALSE</td></tr><tr><td>2000</td><td>100</td><td>40</td><td>0</td><td>40</td><td>10</td><td>FALSE</td></tr><tr><td>2000</td><td>100</td><td>40</td><td>0</td><td>40</td><td>10</td><td>FALSE</td></tr></table> Therefore, no slippage error is displayed on the display screen.  Since the IoT Engine has detected a humidity level less than 50, and no rain/snow on the tracks, the IoT Engine is not displaying an error.
5	Detection of a stationary obstacle on the tracks	**<u>Test Description:</u>** The function *getMoving*() is used to detect whether the object in front of the train is stationary or moving. If the object is moving, returns True, else returns False. **<u>Test 1: Distance Sensors detect stationary object</u>** Distance sensors detect an object on the tracks that is not moving (false in the moving column).

Object Distance	Wheel Speed	Moving	Humidity	Precipitation Level	Temperature	GPS Speed	Gate	Connectivity
1000	100	FALSE	40	0	40	10	FALSE	0
1000	100	FALSE	40	0	40	10	FALSE	1
1000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0

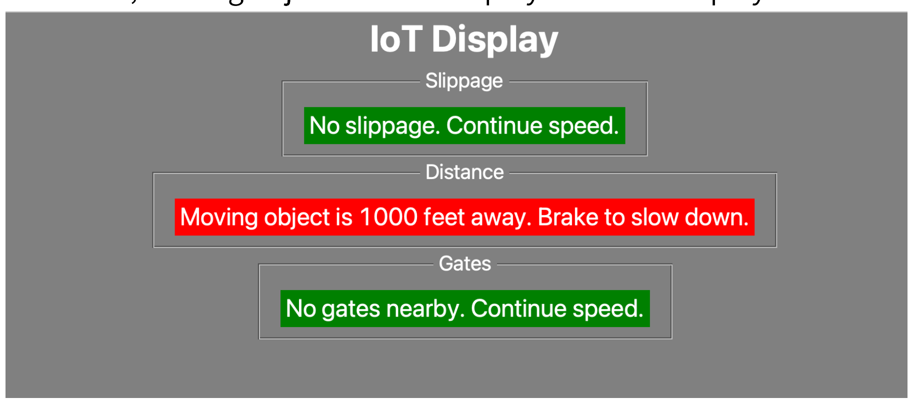
Therefore, stationary object error is displayed on the display screen.



6	Detection of a moving obstacle on the tracks	**<u>Test Description:</u>** The function *getMoving*() is used to detect whether the object in front of the train is stationary or moving. If the object is moving, returns True, else returns False. **<u>Test 1: Distance Sensors detect moving object</u>** Distance sensors detect an object that is less than 1000 feet away and on the tracks that is moving (true in the moving column).

Object Distance	Wheel Speed	Moving	Humidity	Precipitation Level	Temperature	GPS Speed	Gate	Connectivity
1000	100	TRUE	40	0	40	10	FALSE	0
1000	100	TRUE	40	0	40	10	FALSE	1
1000	100	TRUE	40	0	40	10	FALSE	0
2000	100	TRUE	40	0	40	10	FALSE	0
2000	100	TRUE	40	0	40	10	FALSE	0
2000	100	TRUE	40	0	40	10	FALSE	0
2000	100	TRUE	40	0	40	10	FALSE	0
2000	100	TRUE	40	0	40	10	FALSE	0
2000	100	TRUE	40	0	40	10	FALSE	0
2000	100	TRUE	40	0	40	10	FALSE	0
2000	100	TRUE	40	0	40	10	FALSE	0
2000	100	TRUE	40	0	40	10	FALSE	0
2000	100	TRUE	40	0	40	10	FALSE	0
2000	100	TRUE	40	0	40	10	FALSE	0
2000	100	TRUE	40	0	40	10	FALSE	0
2000	100	TRUE	40	0	40	10	FALSE	0
2000	100	TRUE	40	0	40	10	FALSE	0
2000	100	TRUE	40	0	40	10	FALSE	0
2000	100	TRUE	40	0	40	10	FALSE	0
2000	100	TRUE	40	0	40	10	FALSE	0

Therefore, moving object error is displayed on the display screen.



7 Detection of wheel slippage

Test Description:

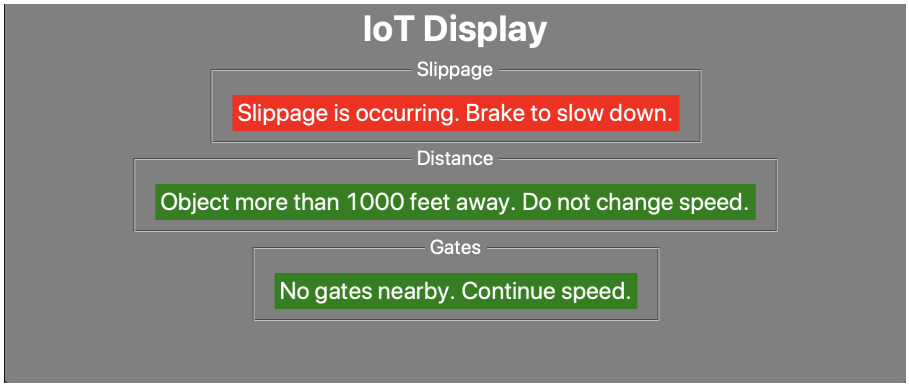
The function *checkslippage()* is used to detect slippage on the tracks. If there is slippage from the Wheel Sensor, then the conductor will be alerted.

Test 1: Wheel Sensors detect slippage

Wheel sensors detect that the wheel speed has a discrepancy greater than 5 miles per hour through *checkslippage()*, meaning there is detected slippage. All other hazardous conditions are not present. The state of the data shows that the RPM (wheel speed) is 100, so the true speed of the wheel is 9.8125 miles per hour and the GPS speed is 20 miles/hour, meaning that there is slippage.

Object Distance	Wheel Speed	Humidity	Precipitation Level	Temperature	GPS Speed	Gate
2000	100	40	0	40	20	FALSE
2000	100	40	0	40	20	FALSE
2000	100	40	0	40	20	FALSE
2000	100	40	0	40	20	FALSE
2000	100	40	0	40	20	FALSE
2000	100	40	0	40	20	FALSE
2000	100	40	0	40	20	FALSE
2000	100	40	0	40	20	FALSE
2000	100	40	0	40	20	FALSE
2000	100	40	0	40	20	FALSE
2000	100	40	0	40	20	FALSE
2000	100	40	0	40	20	FALSE
2000	100	40	0	40	20	FALSE
2000	100	40	0	40	20	FALSE
2000	100	40	0	40	20	FALSE
2000	100	40	0	40	20	FALSE
2000	100	40	0	40	20	FALSE
2000	100	40	0	40	20	FALSE
2000	100	40	0	40	20	FALSE
2000	100	40	0	40	20	FALSE
2000	100	40	0	40	20	FALSE

Therefore, an error is displayed on the display screen.



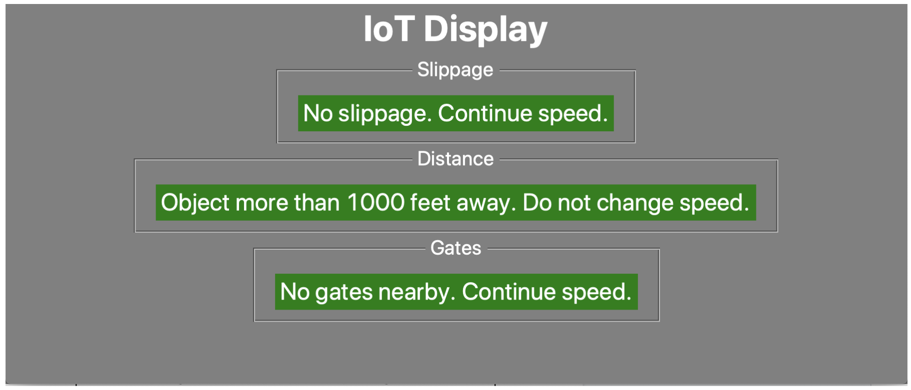
Since the IoT Engine has detected a humidity level greater than 50, the IoT Engine is displaying a slippage error and prompting the conductor to brake to slow the train down.

<u>Test 2: Wheel Sensors do not detect slippage</u>

Wheel sensors detect that the wheel speed has a discrepancy less than 5 miles per hour through *checkslippage()*, meaning there is no detected slippage. All other hazardous conditions are not present. The state of the data shows no slippage.

Object Distance	Wheel Speed	Humidity	Precipitation Level	Temperature	GPS Speed	Gate
2000	100	40	0	40	10	FALSE
2000	100	40	0	40	10	FALSE
2000	100	40	0	40	10	FALSE
2000	100	40	0	40	10	FALSE
2000	100	40	0	40	10	FALSE
2000	100	40	0	40	10	FALSE
2000	100	40	0	40	10	FALSE
2000	100	40	0	40	10	FALSE
2000	100	40	0	40	10	FALSE
2000	100	40	0	40	10	FALSE
2000	100	40	0	40	10	FALSE
2000	100	40	0	40	10	FALSE
2000	100	40	0	40	10	FALSE
2000	100	40	0	40	10	FALSE
2000	100	40	0	40	10	FALSE
2000	100	40	0	40	10	FALSE
2000	100	40	0	40	10	FALSE
2000	100	40	0	40	10	FALSE
2000	100	40	0	40	10	FALSE
2000	100	40	0	40	10	FALSE
2000	100	40	0	40	10	FALSE

Therefore, no slippage error is displayed on the display screen.



Since the IoT Engine has detected no discrepancies in speed, the IoT Engine is not displaying an error.

8 | Detection of gate crossing ahead

<u>Test Description:</u>

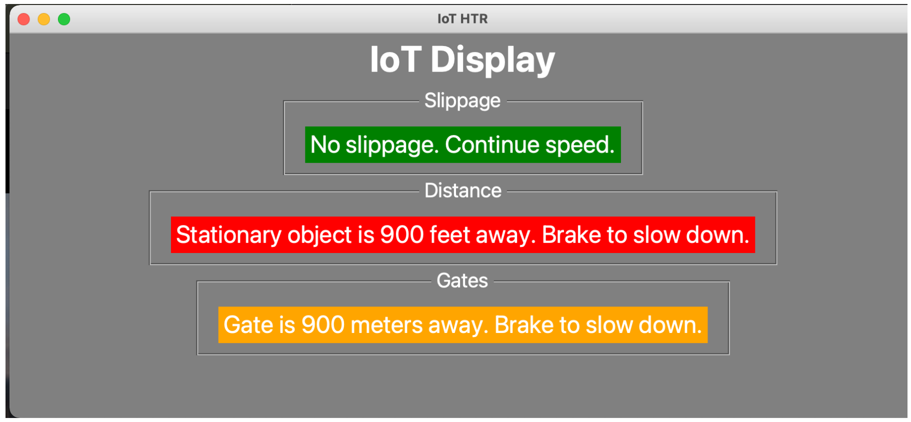
The function *checkGates()* is used to detect if a gate is nearby. If there is a gate detected by the GPS Sensor, then the conductor will be alerted.

<u>Test 1: GPS Sensors detect a gate</u>

Distance sensors detect that there is an object less than a 1000 feet away. GPS sensors detect that there is a gate through *getGate()*, which returns True, meaning there is a gate.

Object Distance	Wheel Speed	Moving	Humidity	Precipitation Level	Temperature	GPS Speed	Gate	Connectivity
900	100	FALSE	40	0	40	10	TRUE	0
900	100	FALSE	40	0	40	10	TRUE	1
900	100	FALSE	40	0	40	10	TRUE	0
2000	100	FALSE	40	0	40	10	TRUE	0
2000	100	FALSE	40	0	40	10	TRUE	0
2000	100	FALSE	40	0	40	10	TRUE	0
2000	100	FALSE	40	0	40	10	TRUE	0
2000	100	FALSE	40	0	40	10	TRUE	0
2000	100	FALSE	40	0	40	10	TRUE	0
2000	100	FALSE	40	0	40	10	TRUE	0
2000	100	FALSE	40	0	40	10	TRUE	0
2000	100	FALSE	40	0	40	10	TRUE	0
2000	100	FALSE	40	0	40	10	TRUE	0
2000	100	FALSE	40	0	40	10	TRUE	0
2000	100	FALSE	40	0	40	10	TRUE	0
2000	100	FALSE	40	0	40	10	TRUE	0
2000	100	FALSE	40	0	40	10	TRUE	0
2000	100	FALSE	40	0	40	10	TRUE	0
2000	100	FALSE	40	0	40	10	TRUE	0
2000	100	FALSE	40	0	40	10	TRUE	0

Therefore, the gate ahead error is displayed on the display screen.



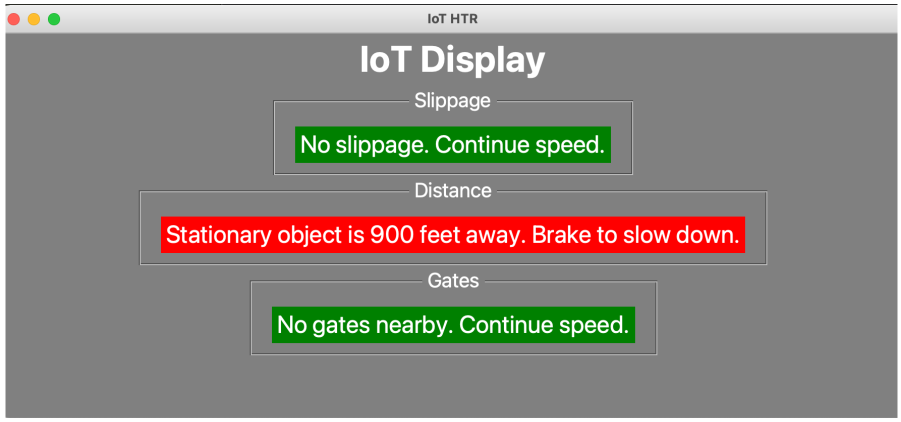
Since the IoT Engine has detected a gate, the IoT Engine is displaying an error.

<u>Test 2: GPS Sensors do not detect a gate</u>

GPS sensors detect that there is no gate through *getGate()*, which returns False, meaning there is no a gate.

Object Distance	Wheel Speed	Moving	Humidity	Precipitation Level	Temperature	GPS Speed	Gate	Connectivity
900	100	FALSE	40	0	40	10	FALSE	0
900	100	FALSE	40	0	40	10	FALSE	1
900	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0

Therefore, the no gate ahead notification is displayed on the display screen.



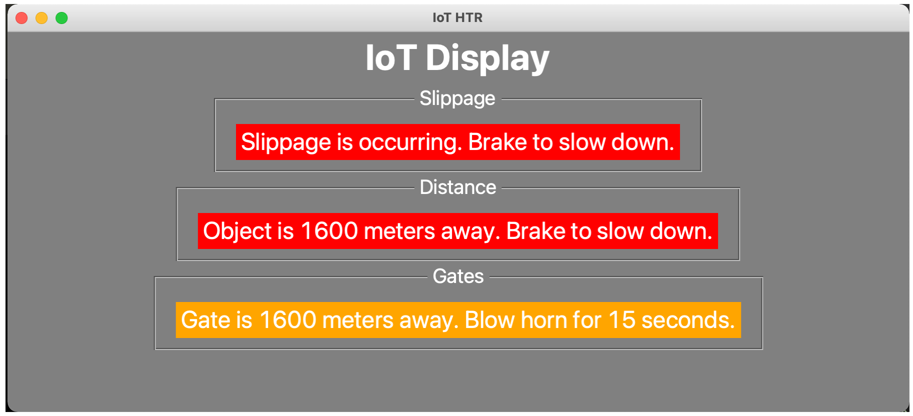
Since the IoT Engine has not detected a gate, the IoT Engine is not displaying an error.

<u>Test 3: GPS Sensors detect a gate 1500-1700 feet away</u>

Distance sensors detect that there is an object between 1500 and 1700 feet away. GPS sensors detect that there is a gate through *getGate()*, which returns True, meaning there is a gate.

Object Distance	Wheel Speed	Moving	Humidity	Precipitation Level	Temperature	GPS Speed	Gate	Connectivity
1600	100	FALSE	40	0	40	10	TRUE	0
50	100	FALSE	40	0	40	10	TRUE	1
900	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0

Therefore, the display shows a notification to blow the horn for 15 seconds.

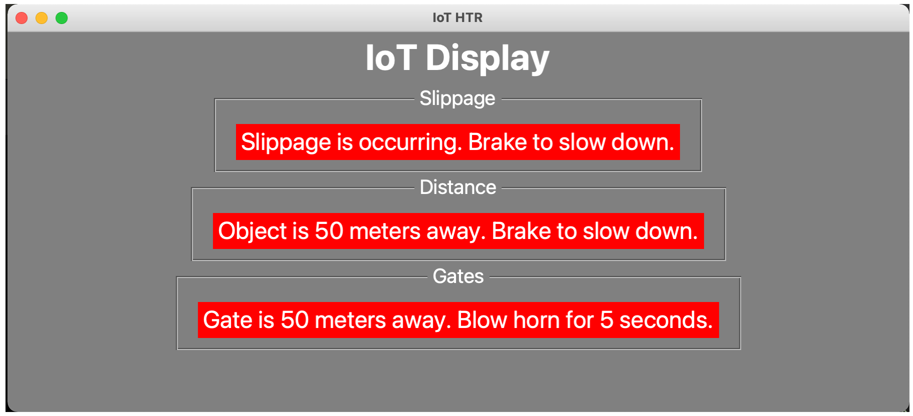


<u>Test 4: GPS Sensors detect a gate less than 100 feet away</u>

Distance sensors detect that there is an object less than 100 feet away. GPS sensors detect that there is a gate through *getGate()*, which returns True, meaning there is a gate.

Object Distance	Wheel Speed	Moving	Humidity	Precipitation Level	Temperature	GPS Speed	Gate	Connectivity
1600	100	FALSE	40	0	40	10	TRUE	0
50	100	FALSE	40	0	40	10	TRUE	1
900	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0

Therefore, the display shows a notification to blow the horn for 5 seconds.



9

Detection of a disconnection from network

Test Description:
The function *tnsr*() takes in the information from the sensordata CSV, and keeps the information in each column in a separate list. When a 1 is detected in the list, the Display stops, and the conductor must login again, meaning a disconnection from the network.

Test 1: Disconnection from the network
The connectivity table shows a 1 to indicate that the connection has been restored and IoT Engine is no longer needed.

Object Distance	Wheel Speed	Moving	Humidity	Precipitation Level	Temperature	GPS Speed	Gate	Connectivity
900	100	FALSE	40	0	40	10	FALSE	0
900	100	FALSE	40	0	40	10	FALSE	1
900	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0
2000	100	FALSE	40	0	40	10	FALSE	0

Therefore, the terminal shows these statements.

		``` ********** WELCOME ********** Loading all processes... Connecting to sensors... Enter a username: conductor Enter a password: password ********** CONDUCTOR LOGGED IN ********** IoT Engine calibrating... Connection has been restored to WiFi/Cellular. IoT Engine no longer needed. Redirecting to normal system... ```