Team 1

Hug the Rails IoT Project

Amna Ahmad, Kaiqi Chee, James Lepore, and Jacob Naeher

Table of Contents

Section 1: Introduction

1.1 Problem Statement

Current train systems require wifi connection to receive information about weather conditions; however, with the loss of internet and cellular connection it is difficult for operators or train systems to receive data and adjust accordingly. By using sensors that allow us to access data on precipitation and wind speed, we shall be able to monitor and adjust the speed at which trains travel without having access to wifi. This shall allow passengers to arrive at their destination in a safer manner and shall also be more cost effective as monitoring and adjusting speeds locally in hazardous conditions shall lead to fewer maintenance issues within the trains themselves.

1.2 Stakeholders and Users

The stakeholders for this project shall be the NJ Transit Corporation, who have asked us to add IoT devices and software to their trains to account for inclement weather or other safety hazards. In addition to the NJ Transit Corporation, Reza Peyrovian and Leah Mitelberg are two high priority stakeholders. This shall make the job of the user, or the locomotive operator, easier; they shall have our software, run by inputs of the sensor data, changing the operation of the train automatically for the train to run in the most efficient and safe way possible, while still being able to enter commands and receive the status.

1.3 Importance and Values

Without any manual intervention, data and tasks shall be transmitted and executed. Since IoT reduces human effort, more time shall be saved, costs shall be reduced and safety shall be improved. As a result, increasing opportunities to analyze data in real-time and to further improve operations. For example, IoT significantly improved farmer lives through smart farming. Farmers use IoT enabled tools to monitor soil composition, soil moisture levels, and livestock activity. The tools collect data that can be analyzed to determine the best time to harvest plants.

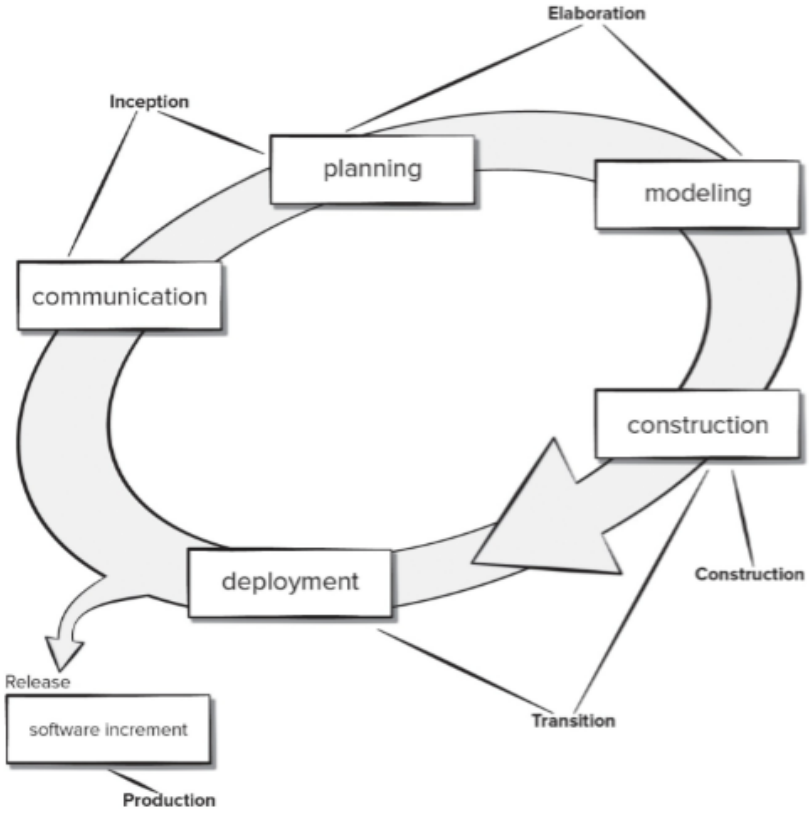
1.4 Expected Delivery

We began this project on February 9th, and hope to have a complete and functional product which shall be ready for deployment by May 1st.

1.5 Approach

Throughout this project, we shall be implementing the unified process model. This shall allow us to have a large amount of quality documentation for the duration of the project. The quality documentation shall ensure that the group has enough information so that the actual construction process shall be well defined and easy to follow. Also, as the group gets feedback from Professor Peyrovian and Leah, the requirements shall be flexible

enough to change or modify based on their requests. The unified process model accommodates requirements changes very well, which is another good reason that the group decided to choose this model.



The chart above outlines the flow of the unified process model. The start date for the group was 2/9, when the group began communicating. The planning phase extends through 3/4, where the group defined the problem and requirements for the solution. The modeling phase shall last from then until 4/5, and construction shall follow until the deployment date of 5/1.

Section 2: Overview

2.1 Define IoT

Iot or Internet of things refers to a system of interconnected, interrelated objects that collect and transfer data over a wireless network without any human intervention. In other words, Iot is an extension of the internet and other networks using sensors and devices.There are several components that come into play regarding Iot: sensors, connection and identification, actuators, Iot gateway, the cloud, and user interface. Sensors are able to measure observable changes in the environment. The type of data collected is primarily dependent on its function. In regards to connection and identification, data must be communicated from the device to the entire Iot system which is accomplished using its IP address. Iot devices should be able to take action based on data collected from sensors and the feedback received from the network. In addition, the Iot gateway acts as a bridge for different devices' data to reach the cloud. Once the cloud has received the data, software can easily reach this data for processing. This is beneficial in the long run as individual devices do not have to reach the cloud separately (less burden). Lastly, user interface allows users to make necessary changes executed by these devices, which adds quality to the overall user experience since there are several types of communication demonstrated (Device-to Device, Device-to-Cloud, Device-to-Gateway, and Back-End-Data-Sharing).

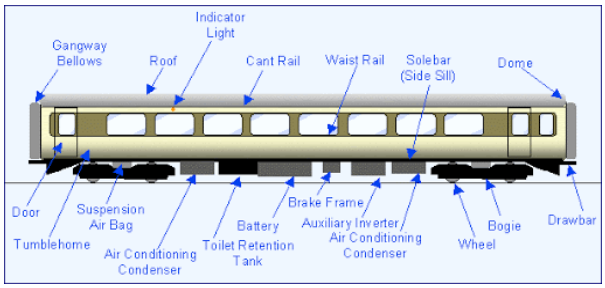
2.2 Define the Problem

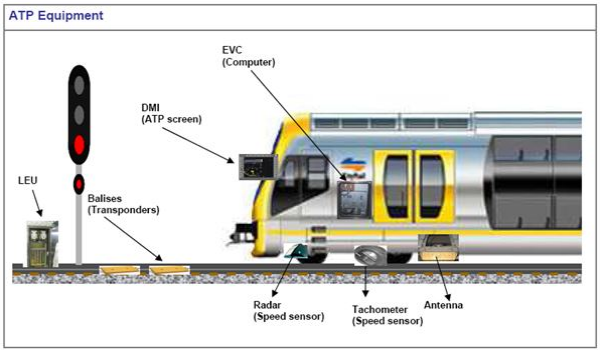
The operation of a train is often dependent on wifi network availability to receive data about environmental, travel, and traffic conditions. Based on those data, the train operator makes decisions about how the train operates. The purpose of this project is to use IoT to make those decisions locally, without being dependent on wifi network availability. This is important because if the train gets disconnected or the network fails, the train is able to continue operating safely.

2.3 Explain how IoT can Solve the Problem

The Internet of Things shall help us solve the problem by allowing us to use two different types of sensors. The first type of sensor shall allow us to detect how much precipitation is occurring in the area where the trains shall be traveling. Based on the information received, the train shall slow down (if there is a lot of rain) or maintain normal speed (if there is little to no rain). Similarly, the second type of sensor shall allow us to monitor wind speeds in the path of the train. Just like the precipitation sensors, the wind sensors shall either maintain the trains current speed (if the wind speed is negligible to low) or slow down the train (in the case where wind speeds could make traveling dangerous). By adjusting the trains speed in hazardous conditions, we hope to achieve a safer method of transportation for customers.

2.4 Overview of Architecture and Components





Components:

- Rain Gauge: Detect amount of rainfall in a given area
    - Hydreon Rain Gauge Model RG-15
- Anemometer: Detect wind speed in a given area
    - Kestrel 1000 Wind Meter
- Sensor to Detect Wheel Slippage
    - Honeywell TARS-IMU Sensors for Wheel Slippage Detection
- Radar Sensor: Detect standing objects on the path with distance; detect a moving object ahead and behind and their speed
    - Infineon´s XENSIV™ 60GHz Radar Sensors
- Separate Display System: Display output and notifications from IoT Engine
    - Advantech 10.4" SVGA / XGA Industrial Panel Mount Monitor
- Camera Sensor: Detect gate crossing open or closed
    - ITS-608 RAIL, 8.6° FOV

Architecture:

- Information from precipitation and wind speed sensors are run through IoT engine
- Metrics are displayed on a separate display system
- IoT engine alerts the operator if changes need to be made

Section 3: Requirements

3.1 Non-Functional Requirements

3.1.1 Reliability Requirements

R-1: Due to the nature of the data being collected by the sensors, the IoT HTR sensors must be able to handle inclement weather. This includes operation in temperatures ranging from -10℉ to 150℉ as well as all types of precipitation and wind speed.

R-2: IoT HTR components shall be able to withstand drops of up to 5 feet.

R-3: IoT HTR sensors shall be powered by the onboard train batteries; this shall allow less maintenance of the individual sensors due to power-failure, as their power-source shall be directly connected to the train.

R-4: IoT HTR system shall have reliability of 0.999. This shall ensure safety of passengers and operators.

3.1.2 Performance Requirements

R-5: The IoT Engine shall have a response time of 0.5 seconds or less, assuming it has been turned on. This shall ensure the operator shall have enough time to react to the sensor readings.

R-6: The IoT Engine shall be able to support up to 1000 sensors, this shall allow sensors to be placed on all relevant parts of the train to detect changes in wind speed and rainfall that may affect the train journey.

3.1.3 Security Requirements

R-7: Any tampering with the hardware of the sensors shall activate security measures to suspend the IoT engine. The data collected affects the alerts and recommended travel speed of the trains, so any tampering that could make the trains travel too fast in hazardous conditions and could have devastating consequences.

R-8: The conductors shall use a User ID and Password to access the data collected by the sensors. This shall also prevent any people who should not have access from retrieving the data.

R-9 : Technician and operator log ins shall be recorded in a read-only file for security tracking purposes.

3.1.4 Operating System Requirements

R-10: 5 TB of data shall be used for storage of data while the trains are traveling. Once trains reach a station, the technician shall have the opportunity to upload the data to a server, freeing the allotted 5 TB for their next trip.

R-11: Windows IoT shall be used as the IoT operating system for HTR

3.2 Functional Requirements

3.2.1 Detect the Wind Speed

R-12: The wind speed sensor shall detect the wind speed in miles per hour in the area in which the train is operating.

R-13: If the speed of the wind is 45+ mph and the train speed itself is greater than 40mph, the IoT Display shall display a warning message in orange and a suggestion to lower speed to 40mph.

R-14: If the speed of the wind is 65+ mph and the train speed itself is greater than 20mph, the IoT Display shall display a warning message in red and a suggestion to lower speed to 20mph.

3.2.2 Detect the Amount of Precipitation

R-15: The precipitation sensor shall detect the amount of precipitation in inches that is in the area in which the train is operating.

R-16: If the amount of rain reaches 0.3+ inches and the train speed itself is greater than 40mph, the IoT Display shall display a warning message in orange and a suggestion to lower speed to 40mph.

R-17: If the speed of the amount of rain is 0.6+ inches and the train speed itself is greater than 20mph, the IoT Display shall display a warning message in red and a suggestion to lower speed to 20mph.

3.2.3 Detect standing objects on the path with distance and suggest speed changes or brake

R-18: The radar sensor shall detect whether there is a stationary object in the path of the train and how far away it is

R-19: For the first 2 positive radar readings, the IoT display shall show a warning message in orange alerting the operator that there is a potential hazard detected ahead.

R-20: If the object ahead is determined by the IoT Engine to be stationary, subsequent positive radar readings shall be displayed as warnings in red and shall tell the operator to stop the train.

3.2.4 Detect a moving object ahead and behind and their speed, direction of move and suggest speed changes or brake

R-21: The radar sensor shall detect any other objects on the rails, and calculate the distance and speed of said object.

R-22: For the first 2 positive radar readings, the IoT display shall show a warning message in orange alerting the operator that there is a potential hazard detected ahead.

R-23: If the object ahead is determined by the IoT Engine to be moving, subsequent positive radar readings shall be displayed as warnings in red and shall tell the operator to stop the train.

3.2.5 Detect gate crossing open or closed, distance and speed (based on GPS), suggest speed changes or brake

R-24: The camera sensor shall be used to detect if the gate is open or closed, whether the red lights are flashing, the distance the train is away from the gate, and the speed.

R-25: If the gate is in the open position, the IoT Display shall display a red warning and suggest the train to stop.

3.2.6 Detect wheel slippage, and its amount using GPS data and the wheel RPM, suggest speed changes, if any, or brake

R-26: The wheel slippage sensor shall be used to detect the amount of slippage occurring in the wheels, based on the wheels RPM and its projected RPM, and based on this data the conductor shall be informed of any speed changes that should be made based on the calculated RPM difference.

R-27: If the difference between the projected speed and calculated wheel speed is between 20-50 RPM and the train speed itself is above 40mph, the IoT Engine shall inform the operator of wheel slippage using the IoT interface and display a caution message in orange and recommend the speed be decreased to 40mph.

R-28: If the difference between the projected speed and calculated wheel speed is greater than 50 and the train speed itself is above 20mph, the IoT Engine shall inform the operator of wheel slippage using the IoT interface and display an extreme caution message and recommend the speed be decreased to 20mph in red.

3.2.7 Sensors shall store their readings in a Log File

R-29: The Log File shall store the current speed, sensor readings, and IoT recommendations every 0.5 seconds during the span of the trip.

R-30: The Log File shall be uploaded to the Cloud Log by the Technician when wifi connection is available, then cleared from the local database.

R-31: The Log File shall only be accessible with valid Technician login information.

3.2.8 IoT Software shall be updated by the Technician

R-32: When wifi connection is present, technician shall have option to update IoT Engine system software.

Section 4: Requirements Modeling

4.1 Use Cases

Use Case: Activation of IoT Engine, Sensors, and IoT Display
Use Case No. : 4.1.1
Primary Actor: Train
Secondary Actor: IoT Engine, Sensors, IoT Display
Goal: Ensure that IoT engine and sensors are working properly
Preconditions:
Trigger: Train turns on
Scenario:

1. Train turning on shall make the IoT Engine and Sensors turn on
2. IoT Display shall turn on and display the login page with the current Sensor data

Exceptions:

1. IoT Display fails to turn on
2. Sensors fails to connect to IoT Engine
3. IoT Engine fails to connect to IoT Display

Use Case: Username & Password Entry
Use Case No. : 4.1.2
Primary Actor: Operator
Secondary Actor: IoT Engine, IoT Display
Goal: Authenticate the operator's username and password
Preconditions: IoT Display is activated
Trigger: Operator attempts to login through IoT Display
Scenario:

1. Operator enters username and password into IoT Display login page
2. IoT accepts username and password and continues to display homepage with sensor data

Exceptions:

1. Username or password is invalid and IoT Engine rejects the login attempt, then prompts Operator to try again.

Use Case: Rain Gauge Detection
Use Case No. : 4.1.3
Primary Actor: Rain Gauge
Secondary Actor: Train Operator, GPS, IoT Display, IoT Engine, Log File
Goal: Notify train operator on the IoT interface in the case of hazardous rain conditions
Preconditions: IoT Display is activated
Trigger: Rain Gauge Sensor is running

Scenario:

1. Rain gauge sensor sends data to IoT Engine.
2. IoT displays “No Hazard” in green if rain level is less than 0.3 inches.
3. IoT displays “Hazardous Condition Detected!” in red or orange if rain level is greater than or equal to 0.3 inches and the train is moving.
    a. If the rain level is greater than 0.3 inches and the train speed is greater than 40 mph, a warning message in orange is displayed and a suggestion to lower speed to 40 mph is displayed.
    b. If the rain level is greater than 0.6 inches and the train speed is greater than 20 mph, a warning message in red is displayed and a suggestion to lower speed to 20 mph is displayed.
2. Train Operator acknowledges the warning on the IoT Display by pressing “Accept Suggestion” which will change the speed of the train, or he ignores it.
3. Log File records appropriate information with timestamp.

Exceptions:

1. The IoT detects rain but fails to read the amount of rain. In this case, inform the operator of potential rain hazard.

Use Case: Anemometer Detection
Use Case No. : 4.1.4
Primary Actor: Anemometer
Secondary Actor: Train Operator, IoT Engine, IoT Display, Log File
Goal: Notify train operator on the IoT interface in the case of hazardous wind conditions
Preconditions: IoT Display is activated
Trigger: Anemometer sensor is running
Scenario:

2. Anemometer sends data to IoT Engine.
3. IoT displays “No Hazard” in green if wind level is less than 55mph.
4. IoT displays “Hazardous Condition Detected!” in red or orange if wind level is greater than or equal to 55mph and the train is moving.
    a. IoT Engine displays a warning message on the IoT interface with recommended speed changes:
        i. If the speed of the wind is 45+ mph and the train speed itself is greater than 40mph, the IoT Display shall display a warning message in orange and a suggestion to lower speed to 40mph.
        ii. If the speed of the wind is 65+ mph and the train speed itself is greater than 20mph, the IoT Display shall display a warning message in red and a suggestion to lower speed to 20mph.
    b. Train Operator acknowledges the message on the interface by pressing “okay”.

c. Log File records operator response with timestamp.

Exceptions:

1. The IoT detects wind but fails to read the windspeed. In this case, inform the operator of potential wind hazard.

Use Case: Wheel Slippage Sensor Detection
Use Case No. : 4.1.5
Primary Actor: Wheel Slippage Sensor
Secondary Actor: Train Operator, IoT Engine, IoT Display, Log File
Goal: Notify train operator on the IoT interface in the case of hazardous wheel slippage
Preconditions: IoT Display is activated
Trigger: Wheel slippage sensor is running
Scenario:

1. Wheel Slippage Sensor sends data to IoT Engine.
2. IoT displays “No Hazard” in green if wheel slippage is less than 20 RPM.
3. IoT displays “Hazardous Condition Detected!” in red or orange if wind level is greater than or equal to 20 RPM and the train is moving.
    a. IoT Engine displays a warning message on the IoT interface with recommended speed changes:
        i. If the difference between the projected speed and calculated wheel speed is between 20-50 RPM and the train speed itself is above 40mph, the IoT Engine shall inform the operator of wheel slippage using the IoT interface and display a caution message in orange and recommend the speed be decreased to 40mph.
        ii. If the difference between the projected speed and calculated wheel speed is greater than 50 and the train speed itself is above 20mph, the IoT Engine shall inform the operator of wheel slippage using the IoT interface and display an extreme caution message and recommend the speed be decreased to 20mph in red.
    b. Train Operator acknowledges the message on the interface by pressing “okay”.
    c. Log File records operator response with timestamp.

Exceptions:

1. The IoT detects wheel slippage but fails to read how much. In this case, inform the operator of potential wheel slippage.

Use Case: Moving Object Detection
Use Case No. : 4.1.6
Primary Actor: Radar Sensor
Secondary Actor: Train Operator, IoT Engine, IoT Display, Log File

Goal: Notify train operator on the IoT interface in the case of moving object ahead
Preconditions: IoT Display is activated
Trigger: Radar sensor is running
Scenario:

1. Radar Sensor sends data to IoT Engine.
2. IoT displays “No Hazard” in green if no moving object detected ahead
3. IoT displays “Hazardous Condition Detected!” in red or orange if moving object is detected ahead
    a. For the first 2 positive radar readings, the IoT Engine displays a warning message in orange on the IoT interface notifying the operator of a hazard ahead
    b. Any following positive radar readings will be displayed in red and recommend that the operator stop the train
    c. Train Operator acknowledges the message on the interface by pressing “okay”.
    d. Log File records operator response with timestamp.

Exceptions:

1. The sensor detects an object but fails to detect if it is moving. In this case, inform the operator of potentially moving objects ahead.

Use Case: Stationary Object Detection
Use Case No. : 4.1.7
Primary Actor: Radar Sensor
Secondary Actor: Train Operator, IoT Engine, IoT Display, Log File
Goal: Notify train operator on the IoT interface in the case of stationary object ahead
Preconditions: IoT Display is activated
Trigger: Radar sensor is running
Scenario:

1. Radar Sensor sends data to IoT Engine.
2. IoT displays “No Hazard” in green if no stationary object detected ahead
3. IoT displays “Hazardous Condition Detected!” in red or orange if stationary object is detected ahead
    a. For the first 2 positive radar readings, the IoT Engine displays a warning message in orange on the IoT interface notifying the operator of a hazard ahead
    b. Any following positive radar readings will be displayed in red and recommend that the operator stop the train
    c. Train Operator acknowledges the message on the interface by pressing “okay”.
    d. Log File records operator response with timestamp.

Exceptions:

1. The sensor detects an object but fails to detect if it is moving. In this case, inform the operator of potentially moving object ahead.

Use Case: Open Gate Detection
Use Case No. : 4.1.8
Primary Actor: Camera Sensor
Secondary Actor: Train Operator, IoT Engine, IoT Display, Log File
Goal: Notify train operator on the IoT interface in the case of an open gate ahead
Preconditions: IoT Display is activated
Trigger: Camera sensor is running
Scenario:

1. Camera Sensor sends data to IoT Engine.
2. IoT displays “No Hazard” in green if closed gate detected ahead
3. IoT displays “Hazardous Condition Detected!” in red if open gate detected ahead
    e. If the gate is in the open position, the IoT Display shall display a red warning and suggest the train to stop.
    f. Train Operator acknowledges the message on the IoT Display by pressing “okay”.
    g. Log File records operator response with timestamp.

Exceptions:

1. The sensor detects a gate but fails to detect if it is open or closed. In this case, inform the operator of a potentially open gate ahead.

Use Case: Technician uploads Log File to Cloud Log
Use Case No. : 4.1.9
Primary Actor: Technician
Secondary Actor: Log File, IoT Display, Cloud Log
Goal: Technician is able to log in and access onboard Log File
Preconditions:

1. Train is on
2. Technician has log in clearance and username/password

Trigger: Technician attempts to log in to IoT.
Scenario:

1. Technician enters username/password into IoT Engine
    a. Technician’s login is valid and access is granted to the logs
2. Technician shall upload Log File to Cloud Log if there is wifi connection; else, an error message displays

Exceptions:

1. Technician's login is invalid, an "incorrect username or password" message is displayed and technician is denied access to logs

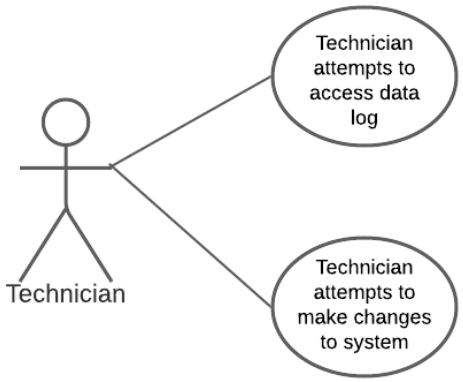
Use Case: Technician makes software update to the Iot Engine system software
Use Case No. : 4.1.10
Primary Actor: Technician
Secondary Actor: IoT Engine, IoT Display
Goal: Technician makes successful changes to the IoT Engine
Preconditions: Train is on, technician has valid clearance, and there is a wifi connection
Trigger: Technician attempts to make changes to the IoT Engine
Scenario:

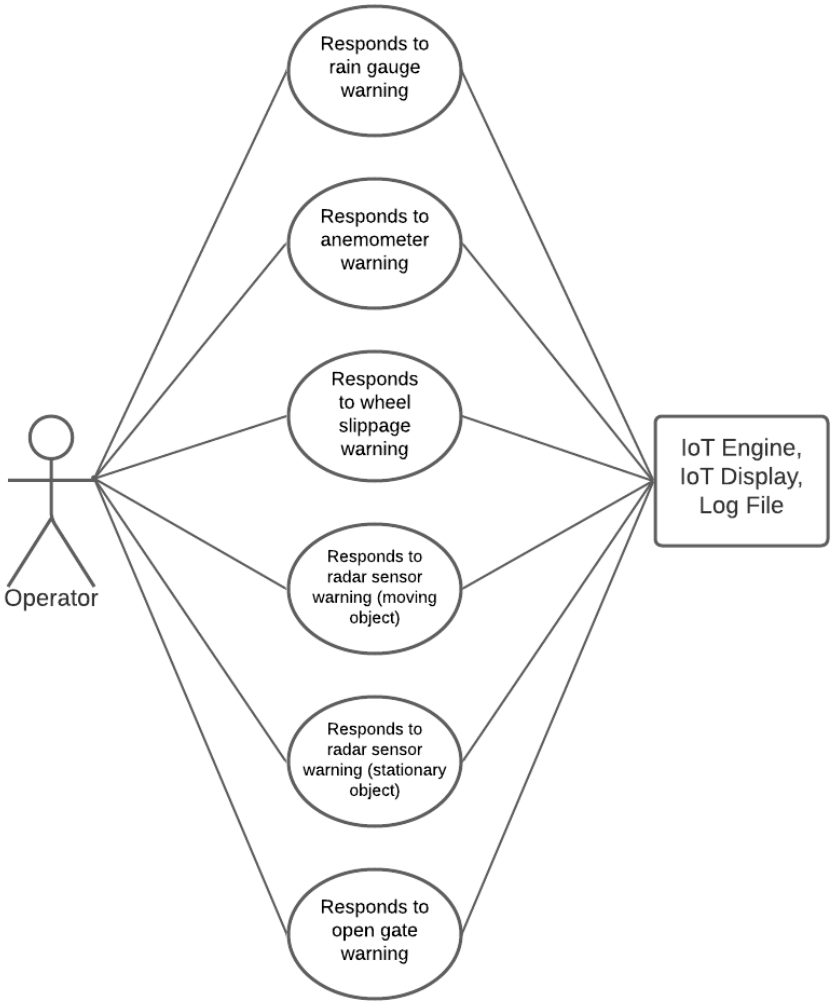
1. Technician logs into IoT Engine either onboard or remotely and selects "update system"
    a. IoT shall display "failed to connect" message if there is no wifi connection on board
2. Technicians shall upload changes to the system.

Exceptions:

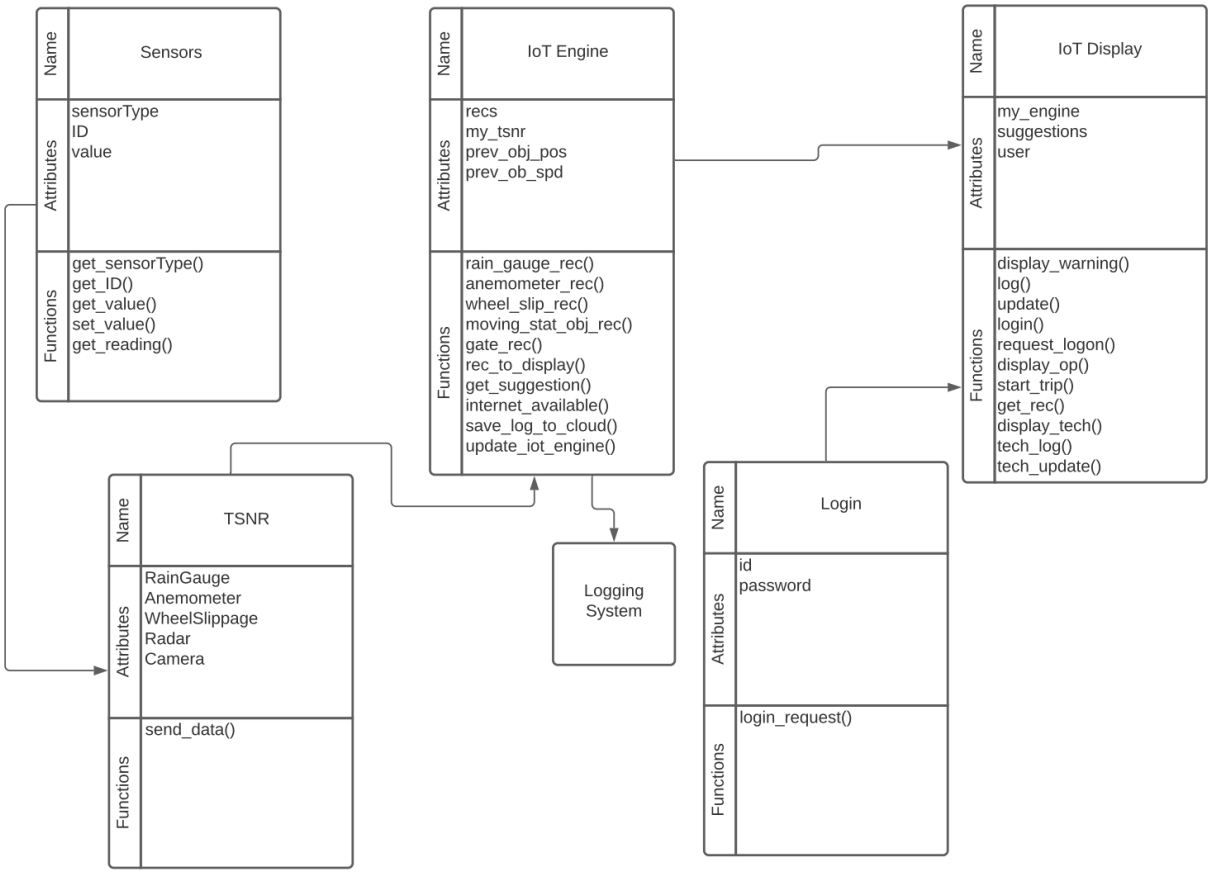
1. If wifi connection is lost part way through upload, all changes are reverted
2. If technician does not have valid clearance or passes an invalid username/password combo they are not allowed access to system
3. If there is no wifi connection, updates are not possible

4.2 Use Case Diagram




Responds to rain gauge warning
Responds to anemometer warning
Responds to wheel slippage warning
Responds to radar sensor warning (moving object)
Responds to radar sensor warning (stationary object)
Responds to open gate warning
Operator
IoT Engine, IoT Display, Log File

4.3 Class-Based Modeling



Each of the sensors shall have 3 attributes: type, ID, and value. The type shall store what type of sensor it is, such as the rain sensors, wind sensors, etc. Each sensor shall have a unique ID, which shall help in specifying which sensor is which. The value shall be the data that is sent directly to the IoT Engine.

The TSNR receives the data from the sensors to send to the IoT Engine. It is there because the data from the different types of sensors shall be sent at different times, so the TSNR ensures that the data that is sent to the IoT Engine is in equal increments.

The IoT Engine shall process all of the data received from the TSNR and turn it into a suggestion or warning to be displayed on the IoT Engine. It shall factor in data from all of the sensors when generating the suggestions. At the end of each trip, the IoT Engine shall send all of the data collected to the Log File, which shall be accessible to all those with a recognized username and the matching password.

The IoT Display shall allow the conductor to see all the data from the sensors. The display shall have separate sections for each of the sensors, which allows the conductor to access data from every hazardous condition that can be detected with the sensors. Each section shall also turn green, orange, or red; depending on the level to which the conditions are hazardous.

The Logging System shall store all of the data collected after each trip, specifically once the train reaches the destination station. This shall only be accessible to users with an authorized username and their matching password, which is verified in the Login.

4.4 CRC Modeling/Cards

Sensors
Description: Capture data on various external factors that may affect the train.
Responsibilities: ● Obtain data on external factors such as rain, wind, obstacles, wheel slippage, and gate openings ● Send the data to the IoT engine

IoT Engine
Description: Process the data received from the sensors and give suggestions, warnings, and recommendations to the train conductor.
Responsibilities: ● Process the data received from the different types of sensors ● Based on the data given, send a suggest and/or warning to the train conductor via the IoT Display ● Send the data at the end of the trip to the Logging System

Logging System
Description: Store the data for the trip and allow those with access to the system to view stored data.
Responsibilities: ● Store data on the different trips that the train takes ● Allows conductors with the correct username and password to view the data on previous trips

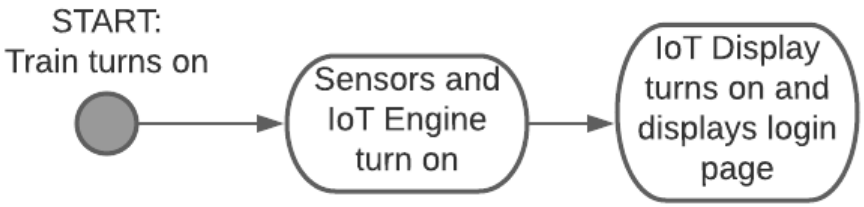
IoT Display
Description: A tablet-like screen next to the conductor which displays information.
Responsibilities: ● Inform conductor of hazardous conditions ● Display warnings and speed suggestions ● Display all data collected from the sensors

Login
Description: Authenticates operator or technician login information.
Responsibilities: ● Authenticate user ID and password ● Display correct display for operator or technician

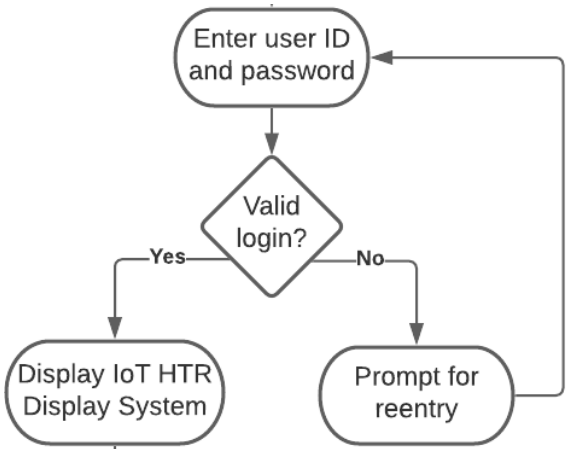
TSNR
Description: Receives sensor readings when they are sent collected and sent.
Responsibilities: ● Collect sensor data when it is sent ● Send collected data to IoTEngine

4.5 Activity Diagram

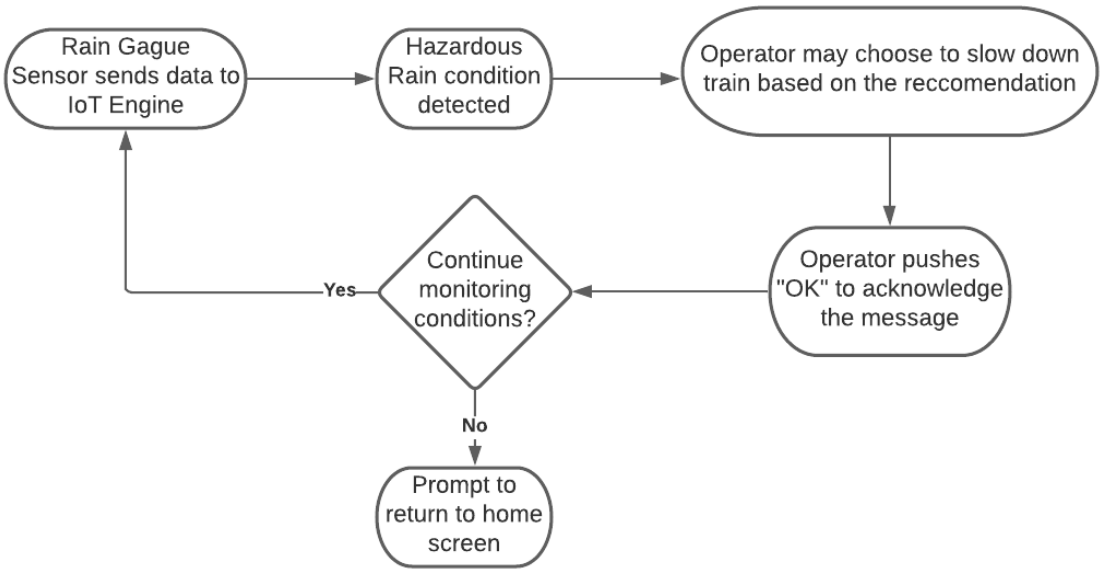
Use Case 4.1.1



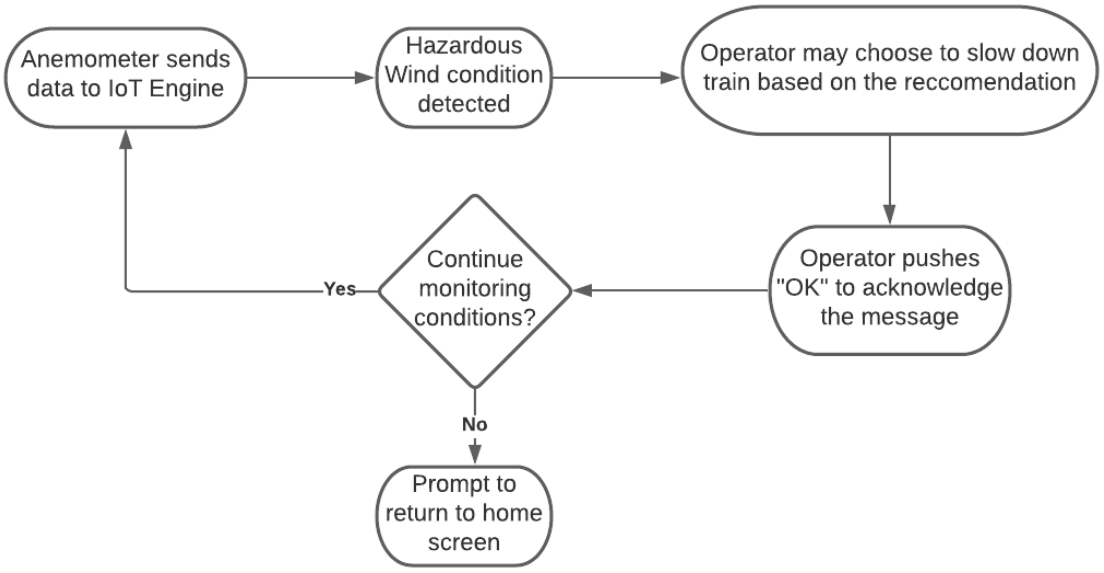
Use Case 4.1.2



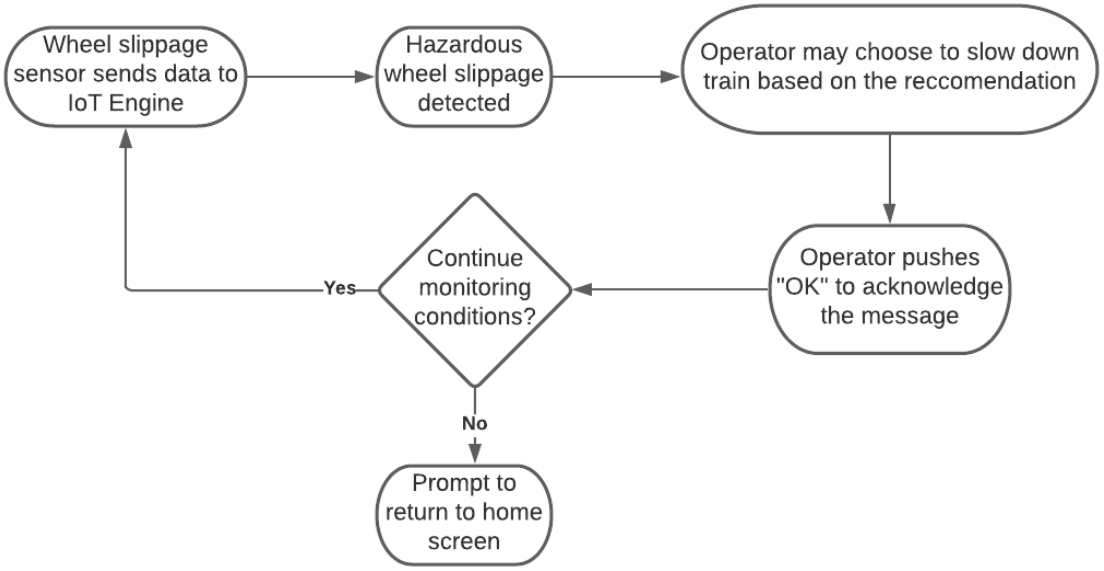
Use Case 4.1.3



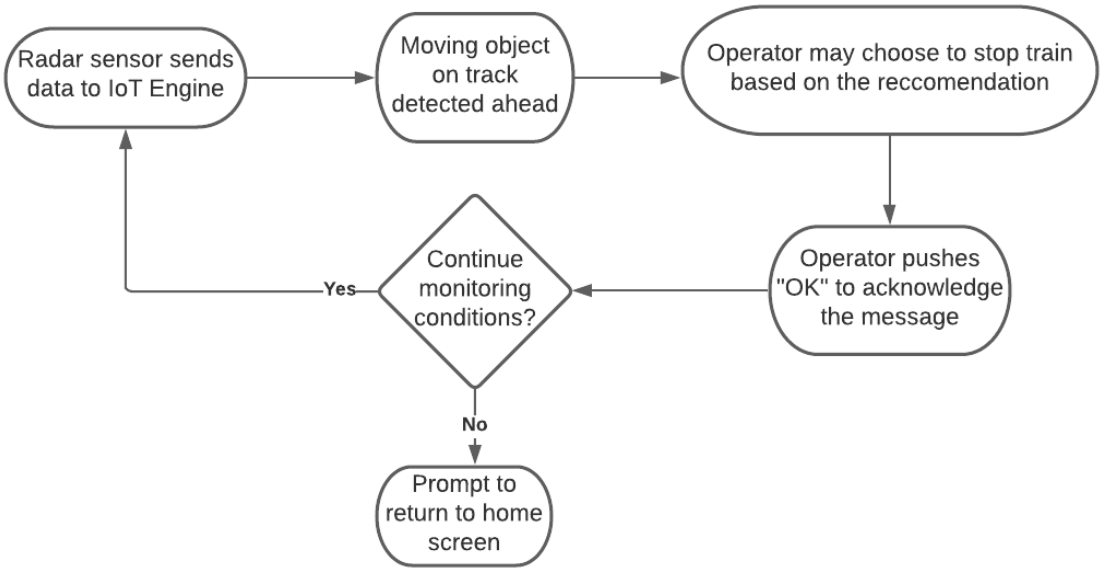
Use Case 4.1.4



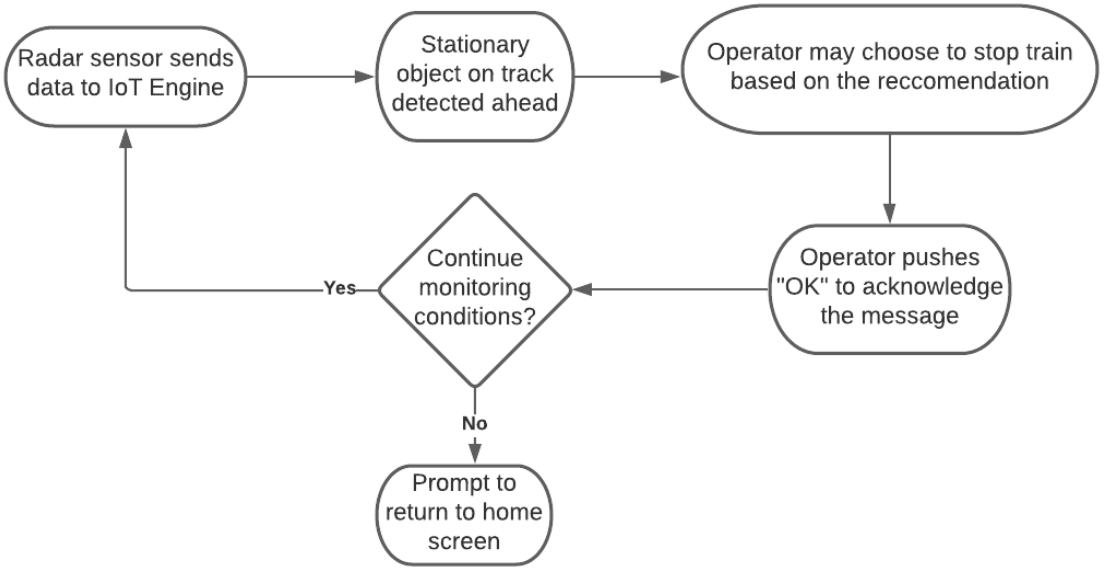
Use Case 4.1.5



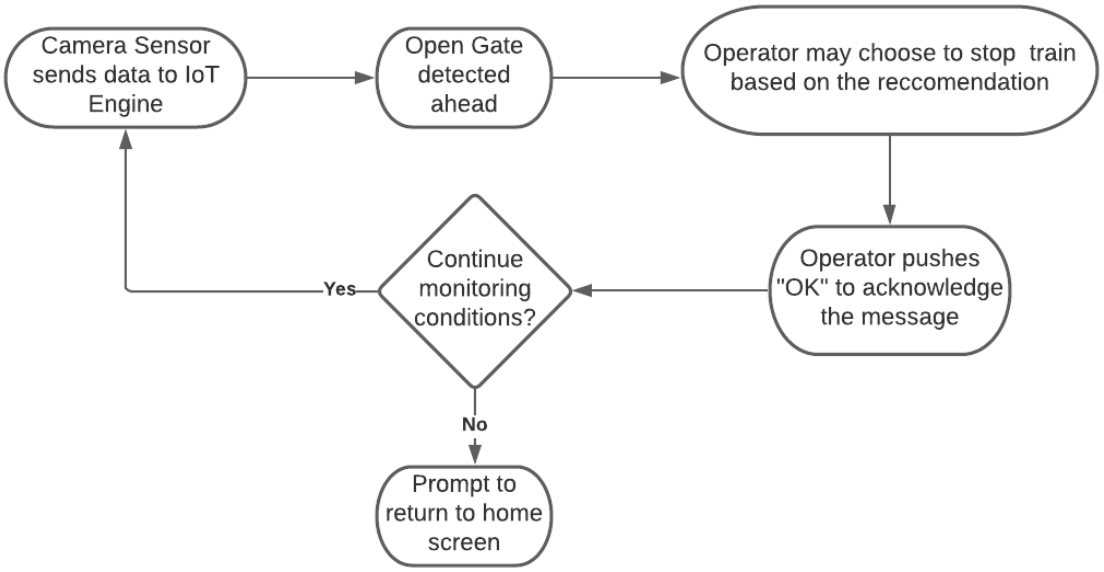
Use Case 4.1.6



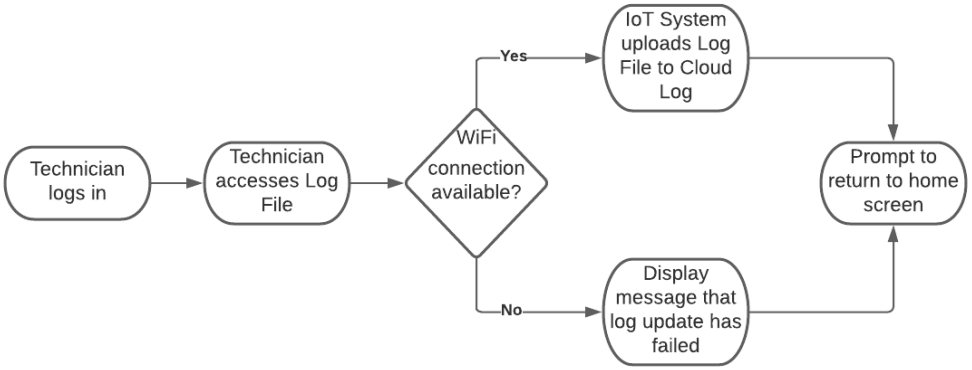
Use Case 4.1.7



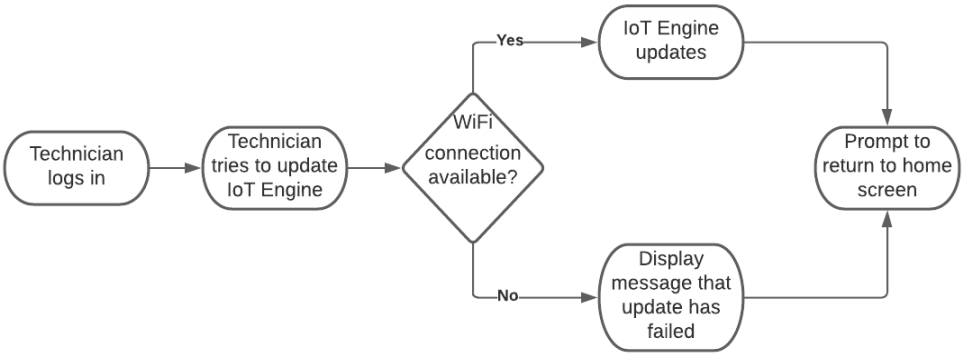
Use Case 4.1.8



Use Case 4.1.9

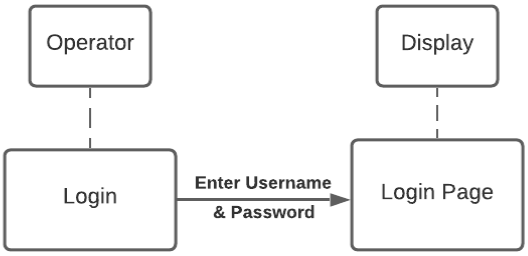


Use Case 4.1.10

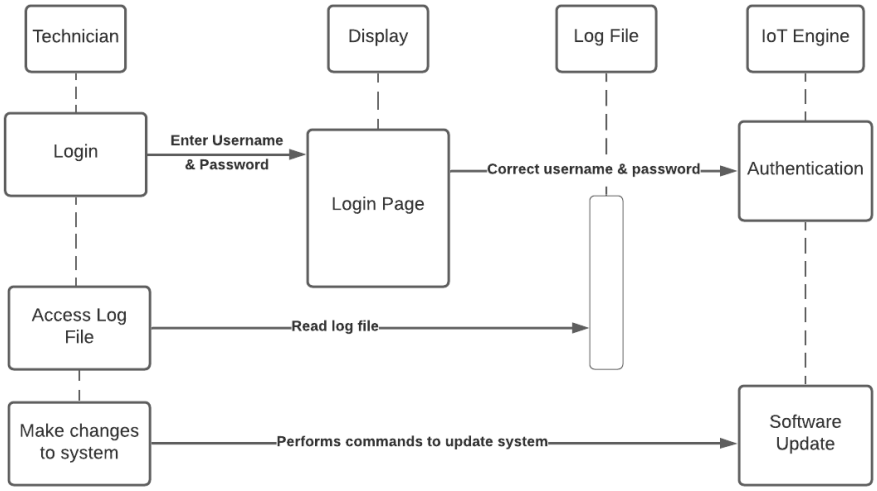


4.6 Sequence Diagrams

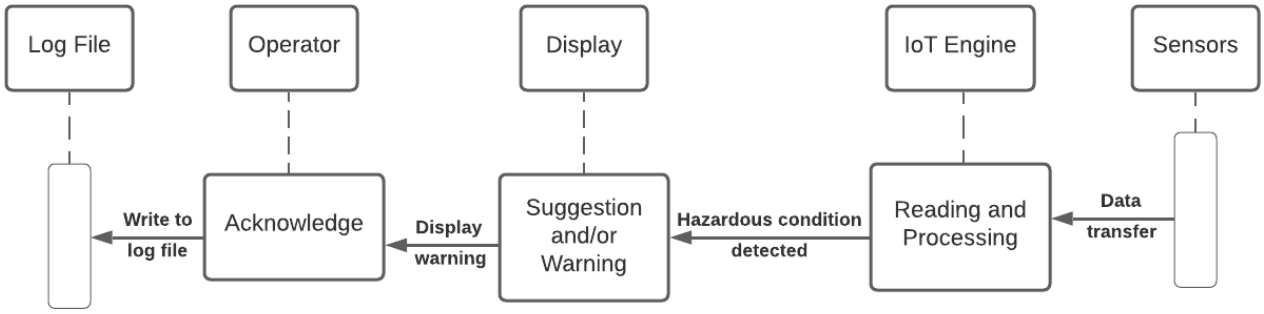
Operator Sequence Diagram:



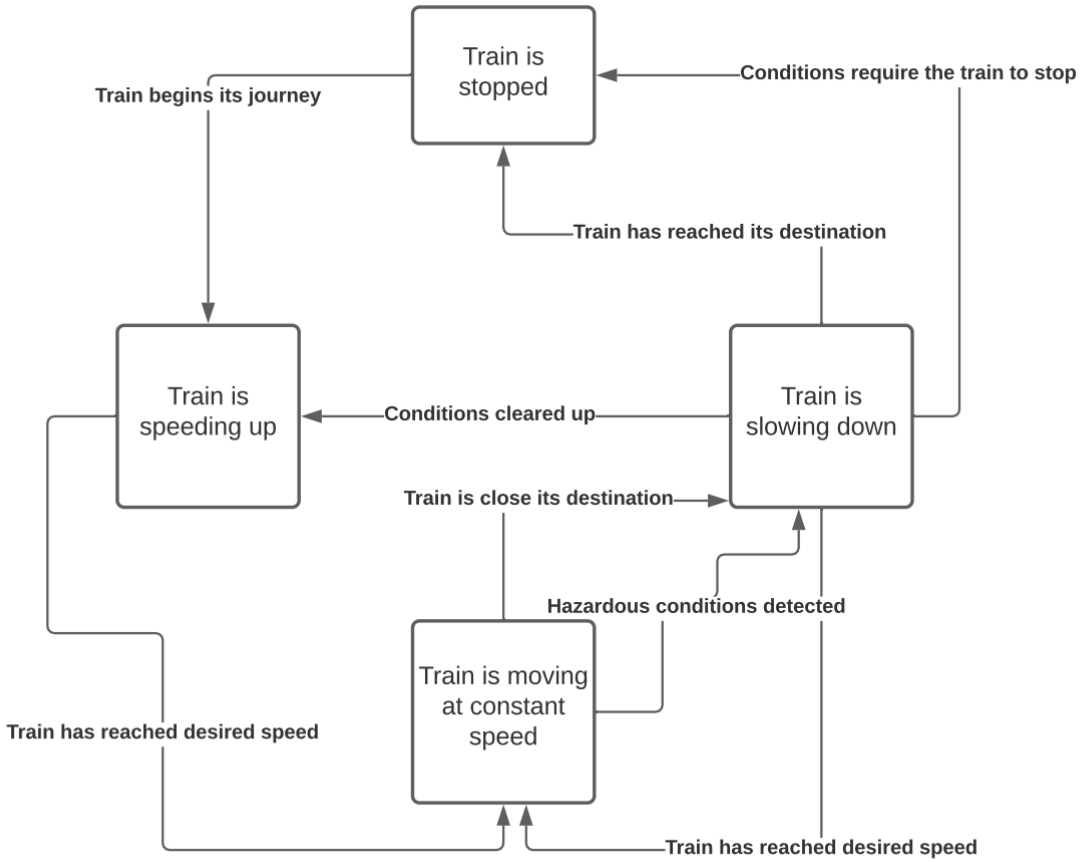
Technician Sequence Diagram:



Sensor Sequence Diagram:



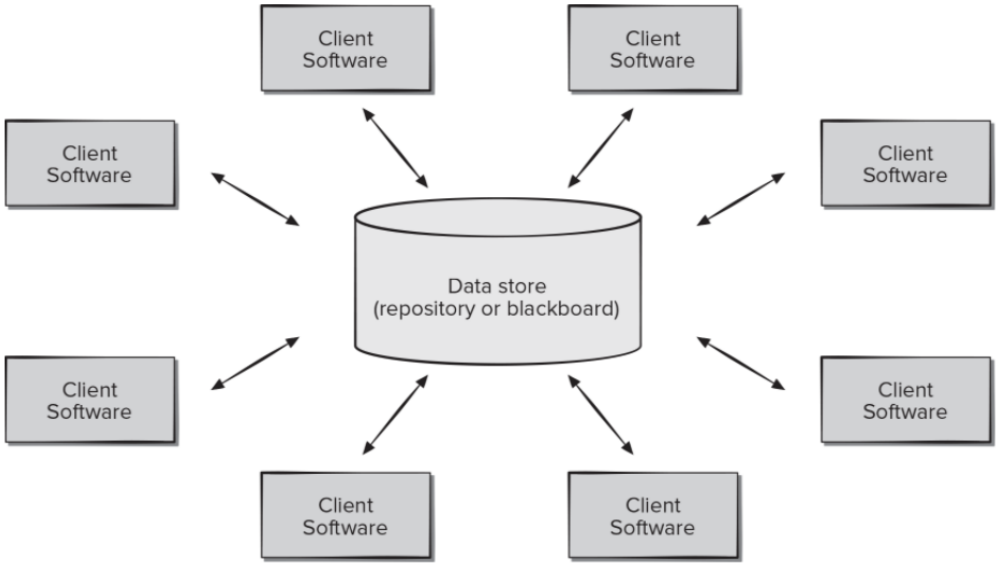
4.7 State Diagram



Section 5: Software Architecture

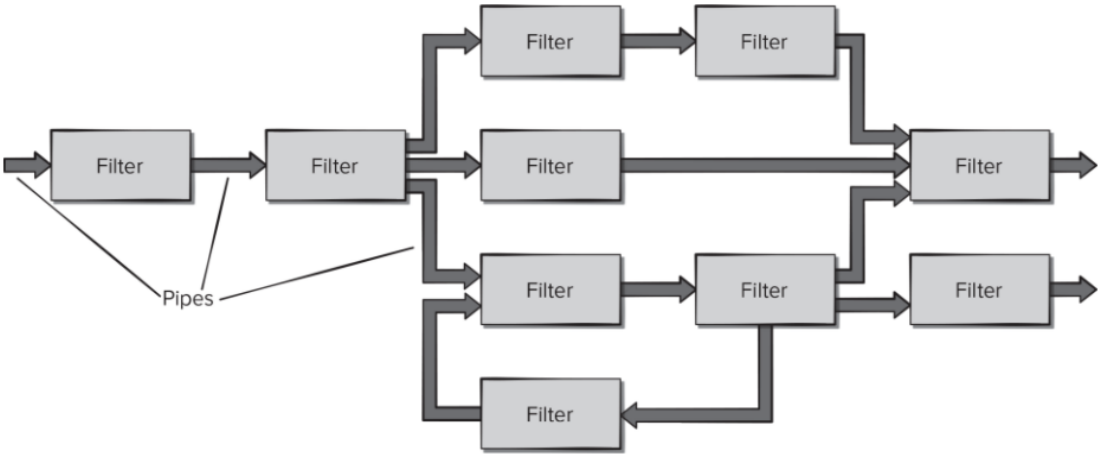
5.1 Software Architecture Models

5.1.1 Data-Centered Architecture



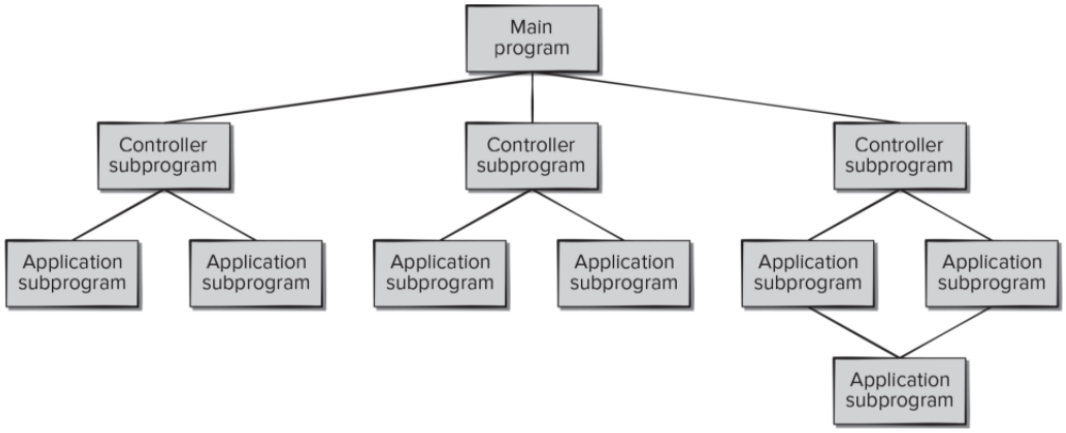
This architectural style is centered around a data store with independent components accessing the data store to update or modify the data. This is not a feasible architecture for IoT Hug the Rails because each sensor makes the system slower and there could be real time delay issues. Also, the data store is not meant to process any logic, it is only meant to hold data. However, we need our IoT engine to process the data, so this model is not feasible. can act as one of the independent components and the central data store can be our IoT Engine.

5.1.2 Data-Flow Architecture



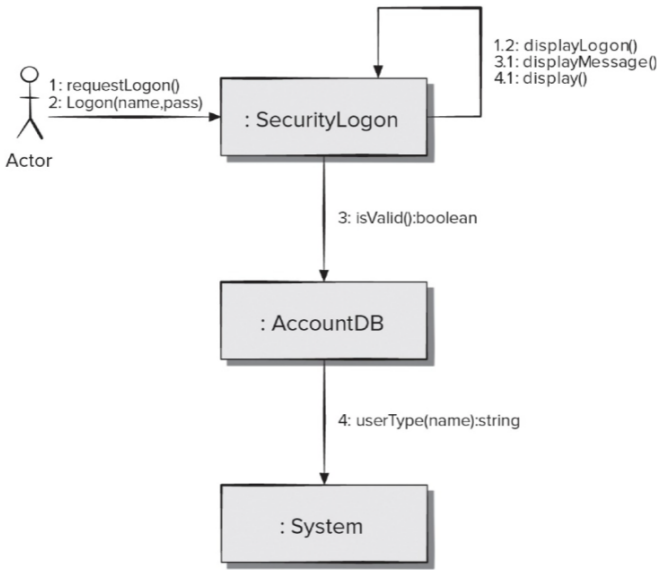
Data-Flow architecture is used when input data is to be manipulated and changed through computations. Each filter acts independently but expects some form of data as an input. The very first filter on the left hand side would be the Time Sensitive Network Router, in order to detect information from the sensors at different times. The second filter would then read the data and pass it on to different filters, where each filter would be either wheel slippage, rain, wind, etc. The very last filter on the right hand side would be the IoT Display, and the process would loop again to keep reading data and/or files. This is the most feasible option for the IoT Engine in the Hug the Rails project as the data would keep getting passed through the filters and the cycle would continue while the train is moving. Once the train stops and the cycle ends, the trip's data would be sent to the Log File.

5.1.3 Call-and-Return Architecture



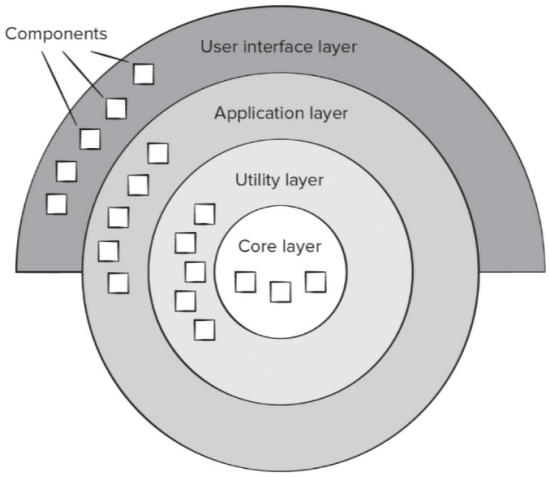
There are two categories of call-and-return architecture: main program/subprogram and remote procedure call. Both of these are used for a structure that can be easy to adjust and scale. This could be a feasible option for IoT Hug the Rails because the main program could call other functions to determine suggestions for the different types of hazards that could disrupt the trains journey, such as rain, wind, etc. This main program shall be in charge of reading data from the sensor and deciding where to send the data, which can be the IoT Engine for analysis and/or the IoT Display to show the operator.

5.1.4 Object-Oriented Architecture



Each component of the system is represented by its own class, which communicates with other classes to transmit, receive, and analyze data. More specifically, the Time Sensitive Network Router would be one object, and the IoT engine would be another object. This is a feasible option for the user login process and would work for this project by having each of our actors displayed on the diagram.

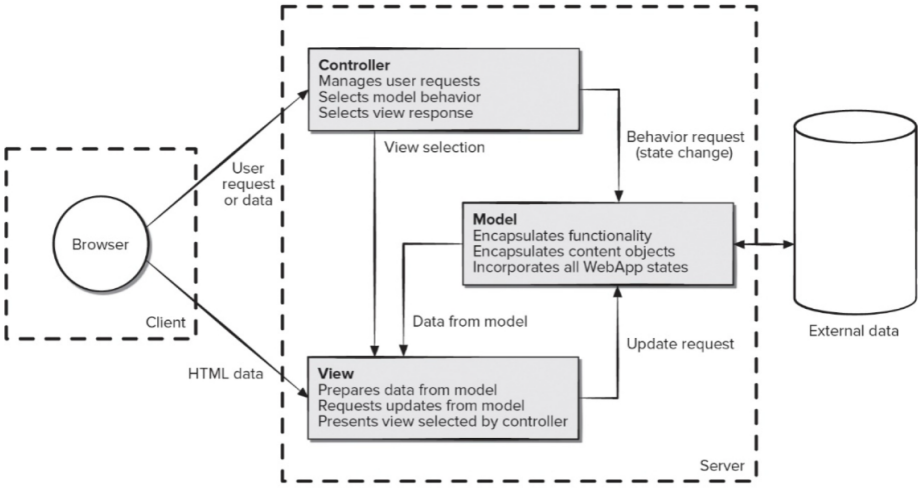
5.1.5 Layered Architecture



The layered architecture model is based on different levels, beginning from the user interface, which is what the conductor would see, all the way down to the core layer, where the sensors send data to the IoT Engine. This is a less feasible option for the IoT Hug the Rails project, because while theoretically each of the different actors could be a different layer in the model, splitting the functionality

into different layers adds time. Since our IoT HTR is a mission-critical system, a split-second delay in calculation could be devastating. This sort of model was meant more for operating systems, and it is hard to implement.

5.1.6 Model-View-Controller Architecture

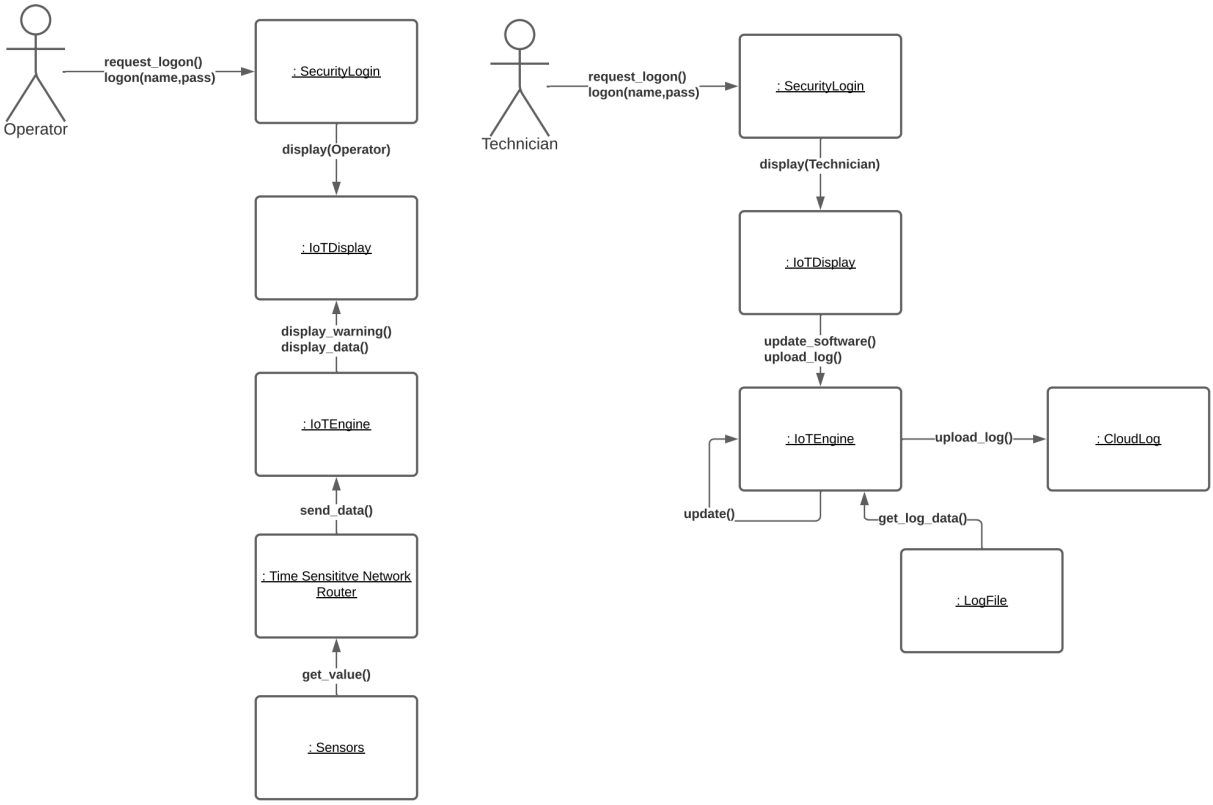


The MVC architecture model is a mobile infrastructure model, mainly used in web development. It has three components: the model, the view, and the controller. If we process data from right to left, the external data takes in the sensor data that has gone through the Time Sensitive Network Router.
This data would be passed to the model, which would represent the IoT engine. The output of the IoT engine would be passed to the view, which is constantly being updated, which would then be taken by the controller and displayed on the browser. The browser shall represent our IoT display and shall display things like data and login pages. The controller decides which view to display based on who logged in and what they need. This could be feasible for the IoT Hug the Rails project, however, as this is really meant to be a web-based model, other models would be better suited.

5.2 Pros and Cons

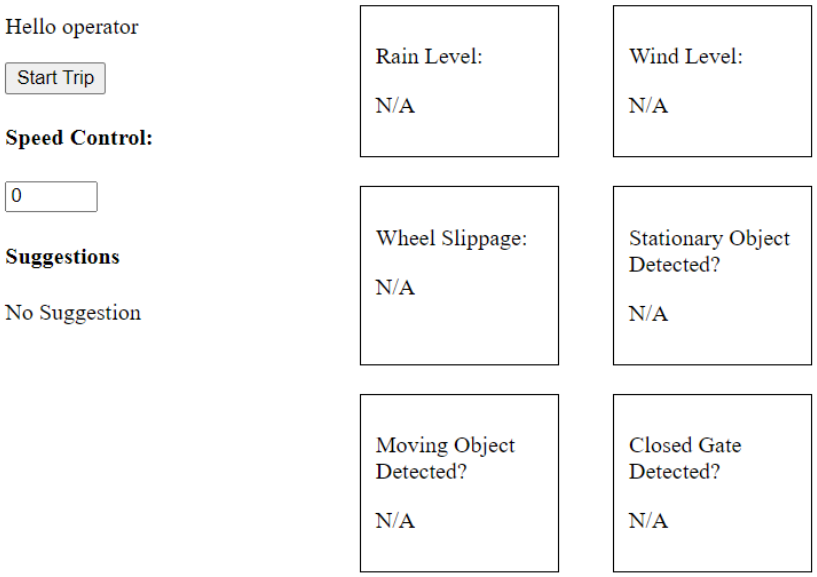
Architecture	Pros	Cons
Data-Centered Architecture	● Provides concurrency that allows all sensors to work in parallel ● Allows us to closely monitor sensor readings through data log ● Allows us to closely monitor operator and technician actions through data log	● A change in the data store may affect every other client software ● May be difficult to test system subcomponents ● Data store shouldn't contain logic ● Real time delay
Data-Flow Architecture	● Allows us to filter the information and check for dangerous conditions	● Filter manipulates data which is not what we want the IoT Engine to do
Call-and-Return Architecture	● Easily scalable program for future sensor additions ● Allows the entire system to be broken into subsections ● Good for subsection debugging	● Subsections may be more difficult to keep track of and maintain through different iterations
Object-Oriented Architecture	● Allows the entire system to be broken into subsections ● Good for subsection debugging	● Forces division of sections into classes, structure is more rigid
Layered Architecture	● Allows the entire system to be broken into subsections ● Good for subsection debugging	● Performance may slow down with more layers being added
Model-View-Controller Architecture	● Predefined sections, allows structure	● Has set sections, less flexibility ● Web based model

5.3 Choice of Architecture



For the Internet of Things Hug the Rails project, we chose to go with an Object-Oriented Architecture. The pros of this architecture are that we can split each component into a different class and this would allow for modular development. The first class would be the security login, the next class would be the IoT Display, IoT Engine, Time Sensitive Network Router, and Sensors. The model allows us to have a clear flow of data and clearly demonstrates how we need to break up our system when we begin development.

The IoT display shall contain a “Start Trip” button as well as a speed control to simulate speed changes. The display shall also have sections to display the current Rain Level, Wind Level, Wheel Slippage, Stationary Object Detected?, Moving Object Detected?, Closed Gate Detected?, as well as a suggestions column to the left which displays suggestions due to hazardous conditions (in corresponding colors).



Section 6: Code

6.1: Sensor Class

```
import csv
import re

class Sensor:
  #constructor to declare attributes of Sensors
  def __init__(self, sensorType, ID):
    self.sensorType = sensorType
    self.ID = ID
    self.value = self.set_value()

  #return sensor type
  def get_sensorType(self):
    return self.sensorType

  #return sensor ID
  def get_ID(self):
    return self.ID

  #returns value from sensor
  def get_value(self):
    return self.value

  #collects sensor readings
  def set_value(self):
    queue = []
    with open('data.csv') as File:
      readFile = csv.reader(File)
      for row in readFile:
        if(self.get_sensorType() == re.sub(r'[^A-Za-z0-9 ]+', '', row[0])):
          queue=row[1:]
    return queue

  #returns most recent sensor reading
  def get_reading(self):
    return self.value.pop(0)


```

6.2: TSNR Class

```
import Sensors as sensors

class TSNR:
  #constructor to declare TSNR attributes
  def __init__(self):
    self.RainGauge = sensors.Sensor("raingauge",4)
    self.Anemometer = sensors.Sensor("anemometer",3)
    self.WheelSlippage = sensors.Sensor("wheelslippage",8)
    self.Radar = sensors.Sensor("radar",9)
    self.Camera = sensors.Sensor("camera",2)

  #sends data from sensors
  def send_data(self):
    return [
        self.RainGauge.get_reading(),
        self.Anemometer.get_reading(),
        self.WheelSlippage.get_reading(),
        self.Radar.get_reading(),
        self.Camera.get_reading()
    ]
```

6.3: Login Class

```
# import display as display

class Login:
    def __init__(self, id, password):
        self.id = id
        self.password = password

    #accepts a login request with user id and password and validates or rejects user
    def login_request(self):
      if(self.id=="operator" and self.password=="cs347"):
        return "operator"
      elif (self.id=="technician" and self.password=="cs347"):
        return "technician"
      else:
        return "Login Failed. Please Try Again."
```

6.4: IoT Engine Class

```
import TSNR as tsnr
from urllib.request import urlopen, URLError
import time

class IoTEngine:
  #constructor to declare attributes of IoTEngine
  def __init__(self):
    self.recs = []
    self.my_tsnr = tsnr.TSNR()
    self.prev_obj_pos = 0
    self.prev_obj_spd = -1

  #Logic that determines display message for rain gauge
  def rain_gauge_rec(self, reading, speed):
    speed = int(speed)
    reading = float(reading)
    if(speed <= 20):
      return ['ok', 'no hazard detected']
    elif(speed > 20 and speed <= 40):
      if(reading < 0.6):
        return ['ok', 'no hazard detected']
      else:
        return ['warning', 'reduce speed to 20 MPH']
    else:
      if(reading < 0.3):
        return  ['ok', 'no hazard detected']
      elif(reading < 0.6):
        return ['hazard', 'reduce speed to 40 MPH']
      else:
        return ['warning', 'reduce speed to 20 MPH']

  #Logic that determines display message for anemometer readings
  def anemometer_rec(self, reading, speed):
    speed = int(speed)
    reading = float(reading)
    if(speed <= 20):
      return ['ok', 'no hazard detected']
    elif(speed > 20 and speed <= 40):
      if(reading < 65):
        return ['ok', 'no hazard detected']
      else:
        return ['warning', 'reduce speed to 20 MPH']
    else:
      if(reading < 45):
        return ['ok', 'no hazard detected']
      elif(reading < 65):
        return ['hazard', 'reduce speed to 40 MPH']
      else:
        return ['warning', 'reduce speed to 20 MPH']

```

```
  #Logic that determines display message for wheel slippage
  def wheel_slip_rec(self, reading, speed):
    speed = int(speed)
    reading = float(reading)
    if(speed <= 20):
      return ['ok', 'no hazard detected']
    elif(speed > 20 and speed <= 40):
      if(reading < 50):
        return ['ok', 'no hazard detected']
      else:
        return ['warning', 'reduce speed to 20 MPH']
    else:
      if(reading < 20):
        return ['ok', 'no hazard detected']
      elif(reading < 50):
        return ['hazard', 'reduce speed to 40 MPH']
      else:
        return ['warning', 'reduce speed to 20 MPH']

  #Logic that determines display message for detection of a moving/stat object
  def moving_stat_obj_rec(self, reading, speed):
    speed = int(speed)
    reading = float(reading)
    if(reading <= 0):
      self.prev_obj_spd = -1
      self.prev_obj_pos = 0
      return ['ok', 'no hazard detected', 'ok', 'no hazard detected']
    if(self.prev_obj_pos == 0):
      self.prev_obj_pos = reading
      return ['hazard', 'potential hazard detected', 'hazard', 'potential hazard detected']
    if(self.prev_obj_spd == 0):
      self.prev_obj_spd = self.prev_obj_pos - reading
      self.prev_obj_pos = reading
      return ['hazard', 'potential hazard detected', 'hazard', 'potential hazard detected']
    if(speed != 0):
      if(self.prev_obj_spd == self.prev_obj_pos - reading):
        self.prev_obj_spd = self.prev_obj_pos - reading
        self.prev_obj_pos = reading
        return ['warning', 'moving object detected', 'ok', 'no hazard detected']
      self.prev_obj_spd = self.prev_obj_pos - reading
      self.prev_obj_pos = reading
      return ['ok', 'no hazard detected', 'warning', 'stationary object detected']
    if(reading != self.prev_obj_pos):
      self.prev_obj_spd = self.prev_obj_pos - reading
      self.prev_obj_pos = reading
      return ['warning', 'moving object detected', 'ok', 'no hazard detected']
    self.prev_obj_spd = self.prev_obj_pos - reading
    self.prev_obj_pos = reading
    return ['ok', 'no hazard detected', 'warning', 'stationary object detected']

  #Logic that determines display message for an open/closed gate
  def gate_rec(self, reading, speed):
    speed = int(speed)
    reading = float(reading)
    if(reading == 1):
      return['warning', 'Stop the train! Reduce speed to 0 MPH']
    return ['ok', 'no hazard detected']

```

```
  #Logic that determines display message
  def rec_to_display(self, speed):
    speed = int(speed)
    if(speed > 0 and (self.recs[7] == "warning" or self.recs[9] == "warning" or self.recs[11] == "warning")):
      return ["0", "Stop the train"]
    if(speed > 20 and (self.recs[1] == "warning" or self.recs[3] == "warning" or self.recs[5] == "waring")):
      return ["20", "Slow the train to 20 MPH or less"]
    if(speed > 40 and (self.recs[1] == "hazard" or self.recs[3] == "hazard" or self.recs[5] == "hazard")):
      return ["40", "Slow the train to 40 MPH or less"]

    return ["0", "---No Suggestion---"]

  #Collects suggestions from prior functions
  def get_suggestion(self, speed):
    data = self.my_tsnr.send_data()
    # data = values for the following sensors [rain,wind,wheel,radar,camera]
    self.recs = [speed] + \
      self.rain_gauge_rec(data[0], speed) + \
      self.anemometer_rec(data[1], speed) + \
      self.wheel_slip_rec(data[2], speed) + \
      self.moving_stat_obj_rec(data[3], speed) + \
      self.gate_rec(data[4], speed) + \
      data
    self.recs += self.rec_to_display(speed)
    file1 = open("LogFile.txt", "a")  # append mode
    file1.write(str(self.recs)+ "\n")
    file1.close()
    return self.recs

  #check if internet connection is available
  def internet_available(self):
    try:
        urlopen('http://216.58.192.142', timeout=1)
        return True
    except URLError as err:
        return False

  #sends sensor data to log file
  def save_log_to_cloud(self):
    if(self.internet_available()):
      f = open('LogFile.txt', 'r+')
      g = open('CloudLog.txt', 'a')
      g.write(f.read())
      f.truncate(0)
      f.close()
      g.close()
      return "Successfully uploaded to Cloud Log."
    return "No Internet Connection. Cannot upload to Cloud Log."

  #updates IoT engine
  def update_iot_engine(self):
    if(self.internet_available()):
      time.sleep(3)
      return "IoT Engine up to date."
    return "No Internet Connection. Cannot update IoT Engine."

```

6.5: IoT Display Class

```python
from flask import Flask,request,render_template,jsonify
import Login as Login
import IoTEngine as IoTEngine
from datetime import datetime

app = Flask(__name__)
app.config['SEND_FILE_MAX_AGE_DEFAULT'] = 0

class Display:
    def __init__(self, user):
      self.my_engine = IoTEngine.IoTEngine()
      self.suggestions = []
      self.user = user

    #displays warning in corresponing color for each sensor
    def display_warning(self, speed):
      self.suggestions = self.my_engine.get_suggestion(speed)
      print("Suggestions:", self.suggestions)
      return ",".join(self.suggestions)

    def log(self):
        return self.my_engine.save_log_to_cloud()

    def update(self):
        return self.my_engine.update_iot_engine()

    #renders the login form
    @app.route('/')
    def login():
        return render_template("login.html")

    #sends username and password, validates user
    @app.route("/login", methods=["POST"])
    def request_logon():
        usr = request.form['usr']
        pwd = request.form['pwd']
        my_login = Login.Login(usr,pwd)
        return my_login.login_request()

    #renders operator display view
    @app.route("/operator", methods=["GET"])
    def display_op():
        return render_template("operator.html")

    #allows the operator to start trip
    @app.route("/operator", methods=["POST"])
    def start_trip():
        op_disp.my_engine = IoTEngine.IoTEngine()
        file1 = open("LogFile.txt", "a")  # append mode
        file1.write("New Trip: " + datetime.now().strftime("%d/%m/%Y %H:%M:%S") + "\n")
        file1.close()
        return str(len(op_disp.my_engine.my_tsnr.RainGauge.value))

```

```python
#recieves a recomendation from the IoTEngine and displays on IoTDisplay
@app.route("/operator/trip", methods=["POST"])
def get_rec():
    speed = request.form['speed']
    return op_disp.display_warning(speed)

#renders technician display view
@app.route("/technician", methods=["GET"])
def display_tech():
    return render_template("technician.html")

#renders technician display log
@app.route("/technician/log", methods=["POST"])
def tech_log():
    return tech_disp.log()

#renders technician option to update software
@app.route("/technician/update", methods=["POST"])
def tech_update():
    return tech_disp.update()
```

6.6 Additional Display and Formatting Files

6.6.1 getData.js

```javascript
var suggestionVal = 0;

//prepares IoTEngine to receive login data
$(document).ready(function() {
        try {
                var u = document.getElementById("uname");
                var p = document.getElementById("psw");

                u.addEventListener("keyup", function(event) {
                        if (event.key == "Enter") {
                                event.preventDefault();
                                document.getElementById("login_button").click();
                        }
                });

                p.addEventListener("keyup", function(event) {
                        if (event.key == "Enter") {
                                event.preventDefault();
                                document.getElementById("login_button").click();
                        }
                });
        } catch {}
});

//validates user login
function login() {
        $.ajax({
                data : {
                  usr : document.getElementById("uname").value,
                  pwd : document.getElementById("psw").value
                },
                type : 'POST',
                url : '/login'
        }).done(function(data) {
                if(data == "Login Failed. Please Try Again."){
                        alert(data);
                } else {
                        window.location.href = '/'+data
                }
        });
}

//allows user to logoff
function logoff() {
        window.location.href = "/"
}

```

```
//begins trip and sensor reading
function start_trip() {
        document.getElementById("start_trip").style.pointerEvents = "none";
        document.getElementById("start_trip").style.opacity = "0.6";
        $.ajax({
                type : 'POST',
                url : '/operator'
        }).done(function(duration) {
                dur = parseInt(duration, 10);
                myLoop(dur);
        });
}

//Continuously runs the IoTEngine until trip is over
function myLoop(dur) {
        setTimeout(function() {
                $.ajax({
                        data : {
                          speed : document.getElementById("speed_val").value,
                        },
                        type : 'POST',
                        url : '/operator/trip'
                }).done(function(data) {
                        if(dur == 0){
                                reset();
                                return;
                        }
                        data = data.split(",");

                        console.log(dur, data);

                        set(data);
                });
                dur--;
                if (dur > 0) {
                  myLoop(dur);
                } else {
                        alert("trip over");
                }
          }, 500);
}

```

```
//sets warning for each sensor
function set(data){
        //Rain
        document.getElementById("rain_val").innerHTML = data[13] + " IN";
        if(data[1] == 'ok') {
                document.getElementById("rain_ok").style.display = "block";
                document.getElementById("rain_hazard").style.display = "none";
                document.getElementById("rain_warning").style.display = "none";
        } else if (data[1] == 'hazard'){
                document.getElementById("rain_ok").style.display = "none";
                document.getElementById("rain_hazard").style.display = "block";
                document.getElementById("rain_warning").style.display = "none";
        } else {
                document.getElementById("rain_ok").style.display = "none";
                document.getElementById("rain_hazard").style.display = "none";
                document.getElementById("rain_warning").style.display = "block";
        }

        //Wind
        document.getElementById("wind_val").innerHTML = data[14] + " MPH"
        if(data[3] == 'ok') {
                document.getElementById("wind_ok").style.display = "block";
                document.getElementById("wind_hazard").style.display = "none";
                document.getElementById("wind_warning").style.display = "none";
        } else if (data[3] == 'hazard'){
                document.getElementById("wind_ok").style.display = "none";
                document.getElementById("wind_hazard").style.display = "block";
                document.getElementById("wind_warning").style.display = "none";
        } else {
                document.getElementById("wind_ok").style.display = "none";
                document.getElementById("wind_hazard").style.display = "none";
                document.getElementById("wind_warning").style.display = "block";
        }

        //Slip
        document.getElementById("slip_val").innerHTML = data[15] + " RPM difference"
        if(data[5] == 'ok') {
                document.getElementById("slip_ok").style.display = "block";
                document.getElementById("slip_hazard").style.display = "none";
                document.getElementById("slip_warning").style.display = "none";
        } else if (data[5] == 'hazard'){
                document.getElementById("slip_ok").style.display = "none";
                document.getElementById("slip_hazard").style.display = "block";
                document.getElementById("slip_warning").style.display = "none";
        } else {
                document.getElementById("slip_ok").style.display = "none";
                document.getElementById("slip_hazard").style.display = "none";
                document.getElementById("slip_warning").style.display = "block";
        }

        //Stationary Object Detection
        document.getElementById("stat_val").innerHTML = (data[9] == "ok" ? 'No' : (data[9] == "hazard" ? 'Calculating...' : data[16] + "M away"))
        if(data[9] == 'ok') {
                document.getElementById("stat_ok").style.display = "block";
                document.getElementById("stat_hazard").style.display = "none";
                document.getElementById("stat_warning").style.display = "none";
        } else if (data[9] == 'hazard'){
                document.getElementById("stat_ok").style.display = "none";
                document.getElementById("stat_hazard").style.display = "block";
                document.getElementById("stat_warning").style.display = "none";
        } else {
                document.getElementById("stat_ok").style.display = "none";
                document.getElementById("stat_hazard").style.display = "none";
                document.getElementById("stat_warning").style.display = "block";
        }

```

```
        //Moving Object Detection
        document.getElementById("mov_val").innerHTML = (data[7] == "ok" ? 'No' : (data[7] == "hazard"? 'Calculating...' : data[16] + "M away"))
        if(data[7] == 'ok') {
                document.getElementById("mov_ok").style.display = "block";
                document.getElementById("mov_hazard").style.display = "none";
                document.getElementById("mov_warning").style.display = "none";
        } else if (data[7] == 'hazard'){
                document.getElementById("mov_ok").style.display = "none";
                document.getElementById("mov_hazard").style.display = "block";
                document.getElementById("mov_warning").style.display = "none";
        } else {
                document.getElementById("mov_ok").style.display = "none";
                document.getElementById("mov_hazard").style.display = "none";
                document.getElementById("mov_warning").style.display = "block";
        }

        //Closed Gate Detection
        document.getElementById("gate_val").innerHTML = (data[17] == "0" ? 'No' : 'Potential')
        if(data[11] == 'ok') {
                document.getElementById("gate_ok").style.display = "block";
                document.getElementById("gate_warning").style.display = "none";
        } else {
                document.getElementById("gate_ok").style.display = "none";
                document.getElementById("gate_warning").style.display = "block";
        }

        // Suggestion Value
        document.getElementById("suggestion").innerHTML = data[19];
        suggestionVal = parseInt(data[18], 10);
        if(data[19] == "---No Suggestion---"){
                document.getElementById("acc_sug").style.display = "none";
        } else {
                document.getElementById("acc_sug").style.display = "block";
        }
}

//registers when operator accepts IoTEngine suggestion
function accept_suggestion() {
        document.getElementById("speed_val").value = suggestionVal;
        document.getElementById("acc_sug").style.display = "none";
}
```

```
//resets IoTEngine
function reset() {
        document.getElementById("start_trip").style.pointerEvents = "auto";
        document.getElementById("start_trip").style.opacity = "1";
        document.getElementById("speed_val").value = 0;
        document.getElementById("suggestion").innerHTML = "---No Suggestion---";
        document.getElementById("acc_sug").style.display = "none";
        document.getElementById("rain_val").innerHTML = "N/A";
        document.getElementById("rain_ok").style.display = "none";
        document.getElementById("rain_hazard").style.display = "none";
        document.getElementById("rain_warning").style.display = "none";
        document.getElementById("wind_val").innerHTML = "N/A";
        document.getElementById("wind_ok").style.display = "none";
        document.getElementById("wind_hazard").style.display = "none";
        document.getElementById("wind_warning").style.display = "none";
        document.getElementById("slip_val").innerHTML = "N/A";
        document.getElementById("slip_ok").style.display = "none";
        document.getElementById("slip_hazard").style.display = "none";
        document.getElementById("slip_warning").style.display = "none";
        document.getElementById("stat_val").innerHTML = "N/A";
        document.getElementById("stat_ok").style.display = "none";
        document.getElementById("stat_hazard").style.display = "block";
        document.getElementById("stat_warning").style.display = "none";
        document.getElementById("mov_val").innerHTML = "N/A";
        document.getElementById("mov_ok").style.display = "none";
        document.getElementById("stat_hazard").style.display = "none";
        document.getElementById("mov_warning").style.display = "none";
        document.getElementById("gate_val").innerHTML = "N/A";
        document.getElementById("gate_ok").style.display = "none";
        document.getElementById("gate_warning").style.display = "none";
}

//Displays technician log
function tech_log() {
        document.getElementById("log_button").style.opacity = "0.6";
        document.getElementById("update_button").style.opacity = "0.6";
        document.getElementById("log_button").style.pointerEvents = "none";
        document.getElementById("update_button").style.pointerEvents = "none";
        $.ajax({
                type : 'POST',
                url : '/technician/log'
        }).done(function(data) {
                alert(data)
                document.getElementById("log_button").style.opacity = "1";
                document.getElementById("update_button").style.opacity = "1";
                document.getElementById("log_button").style.pointerEvents = "auto";
                document.getElementById("update_button").style.pointerEvents = "auto";
        });
}

//Gives technician option to update software
function tech_update() {
        document.getElementById("log_button").style.opacity = "0.6";
        document.getElementById("update_button").style.opacity = "0.6";
        document.getElementById("log_button").style.pointerEvents = "none";
        document.getElementById("update_button").style.pointerEvents = "none";
        $.ajax({
                type : 'POST',
                url : '/technician/update'
        }).done(function(data) {
                alert(data)
                document.getElementById("log_button").style.opacity = "1";
                document.getElementById("update_button").style.opacity = "1";
                document.getElementById("log_button").style.pointerEvents = "auto";
                document.getElementById("update_button").style.pointerEvents = "auto";
        });
}
```

6.6.2 styles.css

```css
.login_container {
        display: flex;
        flex-direction: column;
        justify-content: space-around;
        align-items: center;
        height:200px;
}

.hidden {
        display: none;
}

.ok {
        color: #4CAF50;
}

.hazard {
        color:orange;
}

.warning {
        color:#f4511e;
}

#operator_content {
        display: flex;
        justify-content: space-around;
}

#technician_content div{
        display: flex;
        flex-direction: column;
        justify-content: space-around;
        align-items: center;
}

#settings_container {
        display: flex;
        flex-direction: column;
        align-items: flex-start;
        justify-content: space-between;
        width: 300px;
}

#suggestion_container {
        height: 150px;
}

#acc_sug, #log_button, #update_button {
        border-radius: 4px;
        background-color: #008CBA;
        border: none;
        color: #FFFFFF;
        text-align: center;
        font-size: 14px;
        padding: 20px;
        width: 150px;
        transition: all 0.5s;
        cursor: pointer;
        margin: 5px;
}

```

```
#vals_container {
        display: flex;
        justify-content: space-around;
        flex-wrap: wrap;
        width: 540px;
}

#vals_container div {
        margin: 10px;
        padding: 10px;
        width: 210px;
        height: 125px;
        border: 1px solid black;
}

#start_trip {
        display: inline-block;
        border-radius: 4px;
        background-color: #4CAF50;
        border: none;
        color: #FFFFFF;
        text-align: center;
        font-size: 14px;
        padding: 20px;
        width: 150px;
        transition: all 0.5s;
        cursor: pointer;
        margin: 5px;
}

.cool_button span {
        cursor: pointer;
        display: inline-block;
        position: relative;
        transition: 0.5s;
}

.cool_button span:after {
        content: '\00bb';
        position: absolute;
        opacity: 0;
        top: 0;
        right: -20px;
        transition: 0.5s;
}

.cool_button:hover span {
        padding-right: 25px;
}

.cool_button:hover span:after {
        opacity: 1;
        right: 0;
}

#logoff_button {
        display: inline-block;
        border-radius: 4px;
        background-color: #f4511e;
        border: none;
        color: #FFFFFF;
        text-align: center;
        font-size: 14px;
        padding: 20px;
        width: 150px;
        transition: all 0.5s;
        cursor: pointer;
        margin: 5px;
}
```

6.6.3 HTML Template Files

6.6.3.1 login.html

```
<!DOCTYPE html>
<html>
    <head>
        <title>Hugs the Rail</title>
        <link rel = 'stylesheet' href = {{url_for('static', filename = 'css/style.css')}}></link>
    </head>
    <body>
        <div class="login_container">
            <div>
                <h1>
                    Welcome to IoT HTR System
                </h1>
            </div>

            <div>
                <label for="uname"><b>Username</b></label>
                <input type="text" placeholder="Enter Username" name="uname" id="uname" required>
            </div>

            <div>
                <label for="psw"><b>Password</b></label>
                <input type="password" placeholder="Enter Password" name="psw" id="psw" required>
            </div>

            <button type="submit" id="login_button" onclick="login()">Login</button>
        </div>
        <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
        <script src={{ url_for('static', filename='js/getData.js') }}></script>
    </body>
</html>
```

6.6.3.2 operator.html

```
<!DOCTYPE html>
<html>
    <head>
        <title>Hugs the Rail</title>
        <link rel = 'stylesheet' href = {{url_for('static', filename = 'css/style.css')}}></link>
    </head>
    <body>
        <main id="operator_content">
            <div id='settings_container'>
                <div class="container">
                    <h2>Welcome, operator</h2>
                    <button type="submit" id="start_trip" class="cool_button" onclick="start_trip()"><span>Start Trip</span></button>
                </div>
                <br/>
                <div id="speed_container">
                    <h4>Speed Control:</h4>
                    <input type="number" min="0" max="60" value="0" step="5" onkeydown="return false" id="speed_val" /> MPH
                </div>
                <br/>
                <div id="suggestion_container">
                    <h4>Suggestions</h4>
                    <p id="suggestion">---No Suggestion---</p>
                    <button type="submit" id = "acc_sug" class='cool_button hidden' onclick="accept_suggestion()"><span>Accept Suggestion</span></butto
                </div>
                <br/>
                <div>
                    <button type="submit" id='logoff_button' class='cool_button' onclick="logoff()"><span>Log Off</span></button>
                </div>
            </div>

            <div id="vals_container">
                <div>
                    <p>Rain Level:</p>
                    <p id="rain_val">N/A</p>
                    <p id="rain_ok" class='hidden ok'>No Hazard</p>
                    <p id="rain_hazard" class='hidden hazard'>Hazardous Condition Detected!</p>
                    <p id="rain_warning" class='hidden warning'>Hazardous Condition Detected!</p>
                </div>
                <div>
                    <p>Wind Level:</p>
                    <p id="wind_val">N/A</p>
                    <p id="wind_ok" class='hidden ok'>No Hazard</p>
                    <p id="wind_hazard" class='hidden hazard'>Hazardous Condition Detected!</p>
                    <p id="wind_warning" class='hidden warning'>Hazardous Condition Detected!</p>
                </div>
                <div>
                    <p>Wheel Slippage:</p>
                    <p id="slip_val">N/A</p>
                    <p id="slip_ok" class='hidden ok'>No Hazard</p>
                    <p id="slip_hazard" class='hidden hazard'>Hazardous Condition Detected!</p>
                    <p id="slip_warning" class='hidden warning'>Hazardous Condition Detected!</p>
                </div>
                <div>
                    <p>Stationary Object Detected?</p>
                    <p id="stat_val">N/A</p>
                    <p id="stat_ok" class='hidden ok'>No Hazard</p>
                    <p id="stat_hazard" class='hidden hazard'>Potentially Hazardous Condition Detected.</p>
                    <p id="stat_warning" class='hidden warning'>Hazardous Condition Detected!</p>
                </div>
                <div>
                    <p>Moving Object Detected?</p>
                    <p id="mov_val">N/A</p>
                    <p id="mov_ok" class='hidden ok'>No Hazard</p>
                    <p id="mov_hazard" class='hidden hazard'>Potentially Hazardous Condition Detected.</p>
                    <p id="mov_warning" class='hidden warning'>Hazardous Condition Detected!</p>
                </div>
                <div>
                    <p>Closed Gate Detected?</p>
                    <p id="gate_val">N/A</p>
                    <p id="gate_ok" class='hidden ok'>No Hazard</p>
                    <p id="gate_warning" class='hidden warning'>Hazardous Condition Detected!</p>
                </div>
            </div>
        </main>
        <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
        <script src={{ url_for('static', filename='js/getData.js') }}></script>
    </body>
</html>
```

6.6.3.3 technician.html

```
<!DOCTYPE html>
<html>
    <head>
        <title>Hugs the Rail</title>
        <link rel = 'stylesheet' href = {{url_for('static', filename = 'css/style.css')}}></link>
    </head>
    <body>
        <main id="technician_content">
            <div class="settings_container">
                <h2>Welcome, Technician</h2>
                <br/>
                <div>
                    <button type="submit" id='log_button' class='cool_button' onclick="tech_log()"><span>Upload Log to Cloud</span></button>
                </div>
                <br/>
                <div>
                    <button type="submit" id='update_button' class='cool_button' onclick="tech_update()"><span>Update IoT Engine</span></button>
                </div>
                <br/>
                <div>
                    <button type="submit" id='logoff_button' class='cool_button' onclick="logoff()"><span>Log Off</span></button>
                </div>
            </div>
        </main>
        <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
        <script src={{ url_for('static', filename='js/getData.js') }}></script>
    </body>
</html>
```

Section 7: Testing

7.1: Scenario-Based Testing

The Scenario-Based tests have been developed to document the use cases outlined in Section 4.1.

7.1.1: Activation of IoT Engine, Sensors, and IoT Display

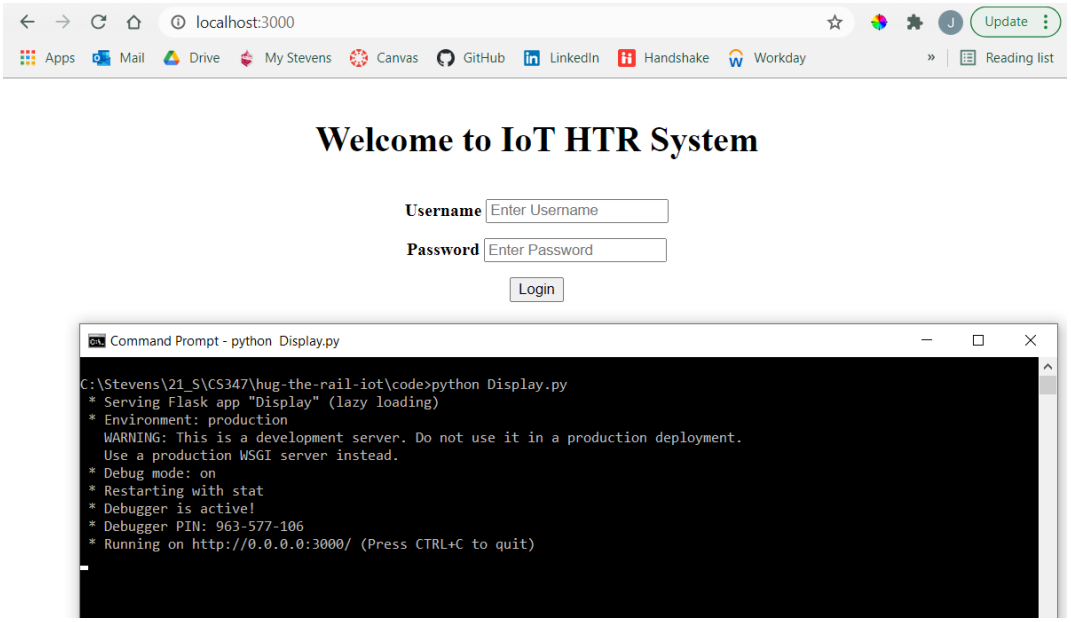
Test Description: This test is to verify the activation of the Engine, Sensors, and Display when the train turns on.

Methodology:

- Begin running python file for IoT Display with command “python Display.py” to simulate the train turning on and activating the Display

Expected Outcome: Flask server will begin to locally host the IoT Display for the user to view

Actual Outcome: Matches Expected Outcome ✓



7.1.2 Username & Password Entry

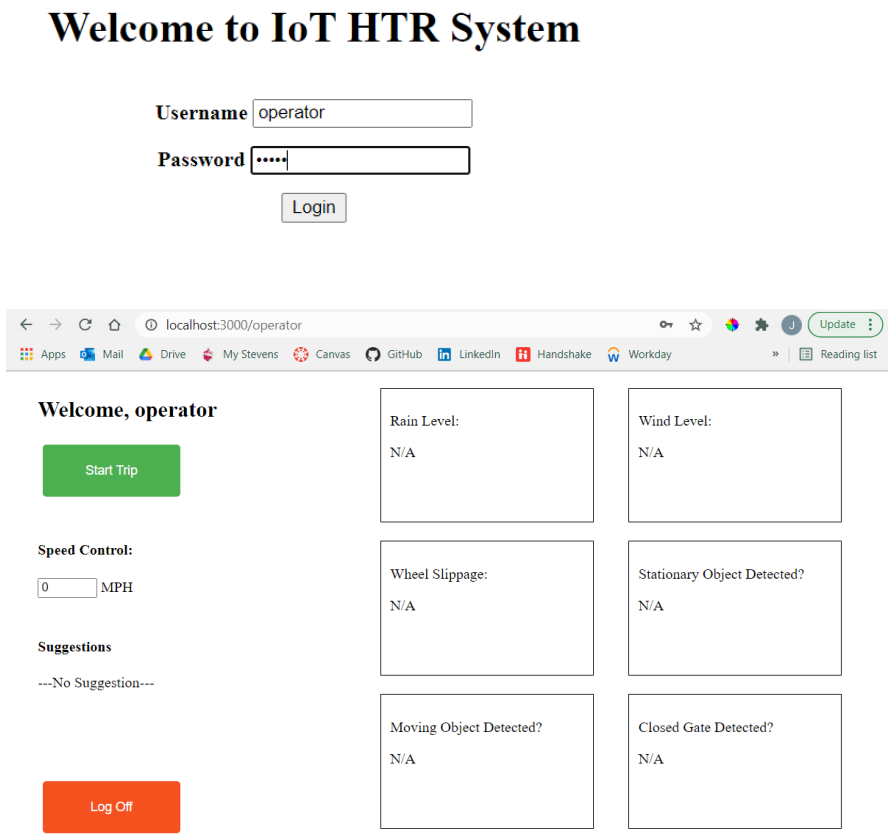
Test Description: This test is to verify the login process for the users of the IoT Display.

Methodology:

- Enter “operator” in the username field and “cs347” in the password field
- Click “Login”

Expected Outcome: Input will result in a successful login.

Actual Outcome: Matches Expected Outcome ✓



7.1.3 Rain Gauge Detection

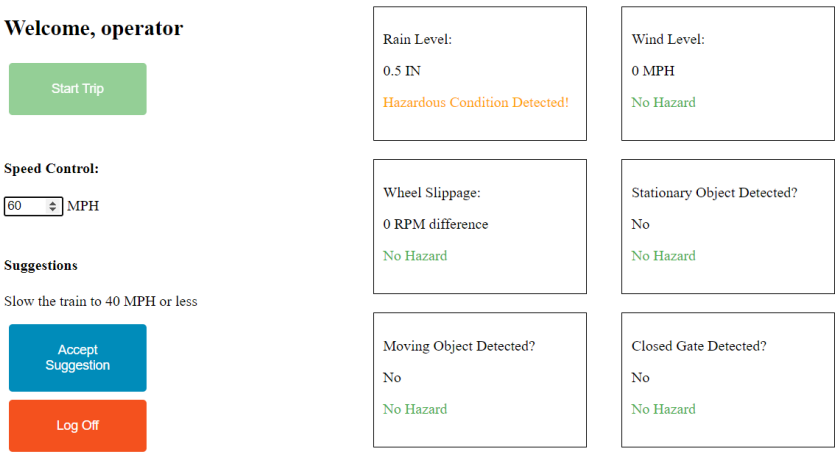
Test Description: This test is to verify the accuracy of the suggestions based on the data received for the Rain Gauge Sensors.

Methodology:

- Operator starts trip
- Operator raises speed to 60 MPH
- Rain Gauge detects 0.5 IN of rain

Expected Outcome: “Hazardous Conditions Detected!” in orange text should display, as well as a suggestion message to slow down to 40 mph.

Actual Outcome: Matches Expected Outcome ✓



7.1.4 Anemometer Detection

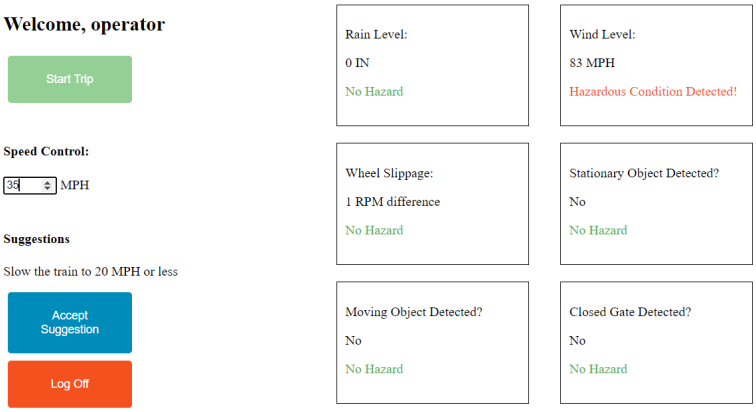
Test Description: This test is to verify the accuracy of the suggestions based on the data received for the Anemometer Sensors.

Methodology:

- Operator starts trip
- Operator raises speed to 35 mph
- Anemometer detects wind speeds of 83 mph

Expected Outcome: “Hazardous Conditions Detected!” in red text should display, as well as a suggestion to slow down to 20 mph.

Actual Outcome: Matches Expected Outcome ✓



7.1.5 Wheel Slippage Sensor Detection

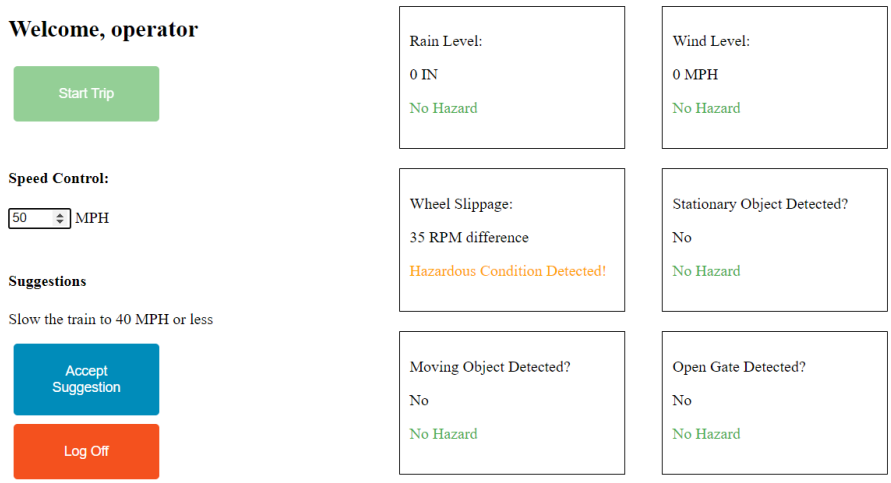
Test Description: This test is to verify the accuracy of the suggestions based on the data received for the Wheel Slippage Sensors.

Methodology:

- Operator starts trip

- Operator raises speed to 50 mph
- Wheel Slippage Sensors detects 35 amount of wheel slippage

Expected Outcome: “Hazardous Condition Detected!” in orange text should display, as well as a suggestion to reduce speed to 40 mph.
Actual Outcome: Matches Expected Outcome ✓

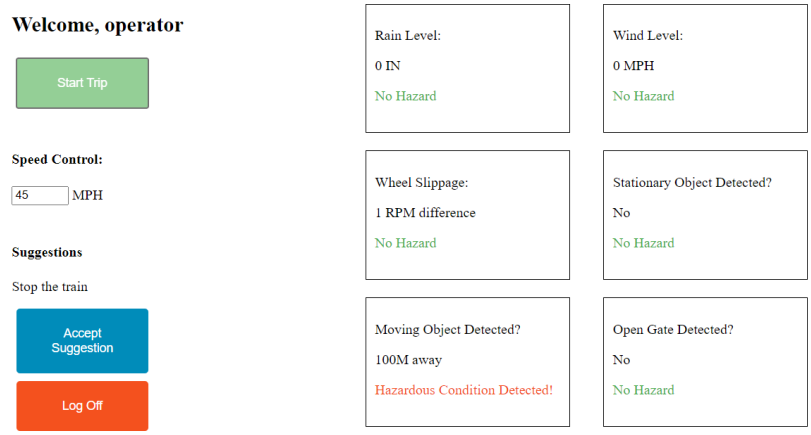


7.1.6 Moving Object Detection

Test Description: This test is to verify the accuracy of the suggestions based on the data received from the Radar Sensor for the moving objects.
Methodology:

- Operator starts trip
- Operator raises speed to 45 mph
- Radar sensor detects a moving object in the path of the train

Expected Outcome: “Hazardous Condition Detected!” in red text should display, as well as a suggestion to stop the train.
Actual Outcome: Matches Expected Outcome ✓



7.1.7 Stationary Object Detection

Test Description: This test is to verify the accuracy of the suggestions based on the data received from the Radar Sensor for the stationary objects.

Methodology:

- Operator starts trip
- Operator raises speed to 30 mph
- Radar sensor detects a stationary object in the path of the train

Expected Outcome: “Hazardous Condition Detected!” in red text should display, as well as a suggestion to stop the train.

Actual Outcome: Matches Expected Outcome ✓

7.1.8 Open Gate Detection

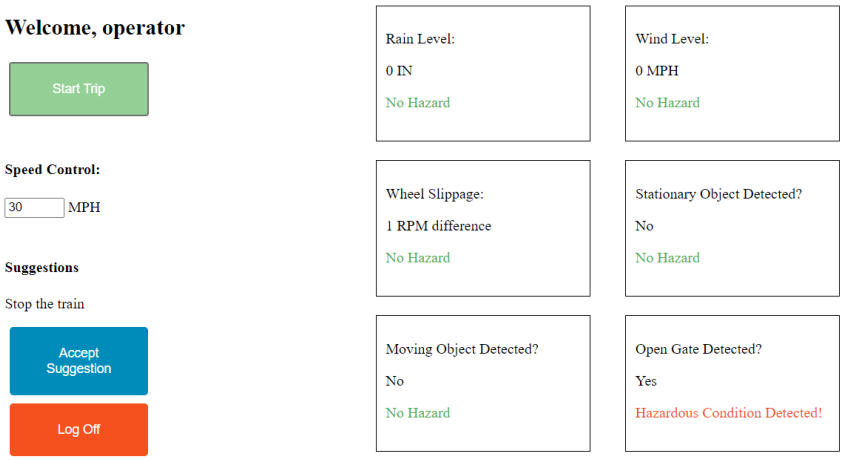
Test Description: This test is to verify the accuracy of the suggestions based on the data received from the Camera Sensor for detecting whether a gate is opened or closed.

Methodology:

- Operator starts trip
- Operator raises speed to 30 mph
- Camera sensor detects a gate is open

Expected Outcome: “Hazardous Condition Detected!” in red text should display, as well as a suggestion to stop the train.

Actual Outcome: Matches Expected Outcome ✓



7.1.9 Technician uploads Log File to Cloud Log

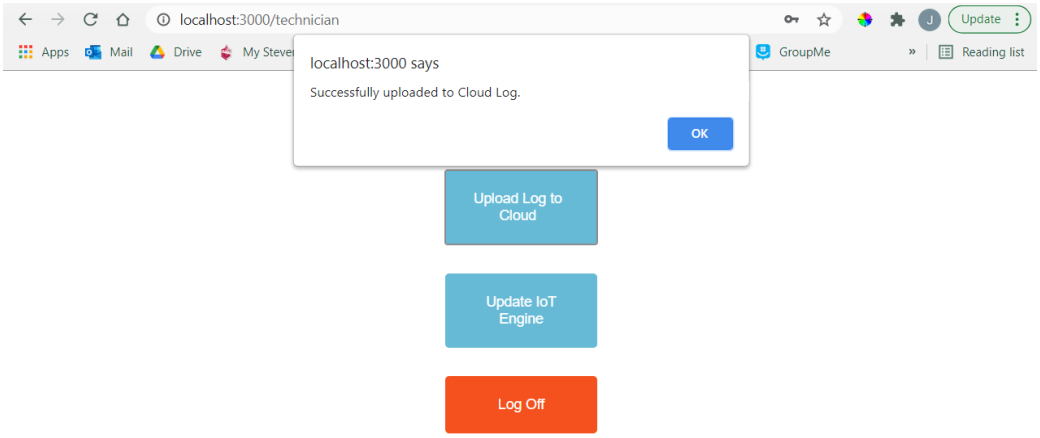
Test Description: This test is to verify whether the technician can upload the Log File to the Cloud Log successfully, depending on if there is a wifi connection or not.

Methodology:

- Click the “Upload Log to Cloud button

Expected Outcome: “Successfully uploaded to Cloud Log.” is displayed

Actual Outcome:Matches Expected Outcome ✓



7.1.10 Technician makes software update to the Iot Engine system software

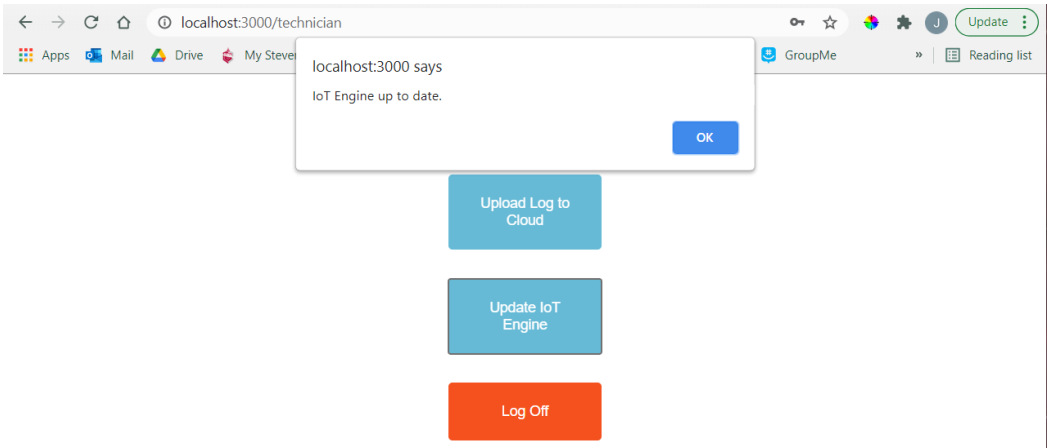
Test Description: This test is to verify whether the technician can update the IoT Engine system, depending on if there is a wifi connection or not.

Methodology:

- Click the “Update IoT Engine”

Expected Outcome: “IoT Engine up to date.” is displayed

Actual Outcome: Matches Expected Outcome ✓

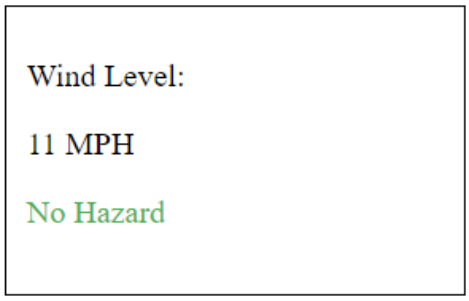

localhost:3000/technician
Update
Apps
Mail
Drive
GroupMe
Reading list
localhost:3000 says
IoT Engine up to date.
OK
Upload Log to Cloud
Update IoT Engine
Log Off

7.2: Validation Testing

The Validation tests have been developed to document the functional requirements outlined in Section 3.2.

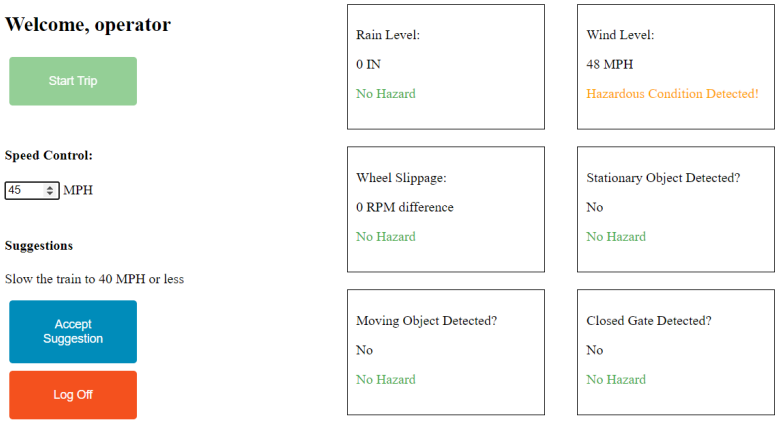
Detect the Wind Speed

R-12: The wind speed sensor shall detect the wind speed in miles per hour in the area in which the train is operating.



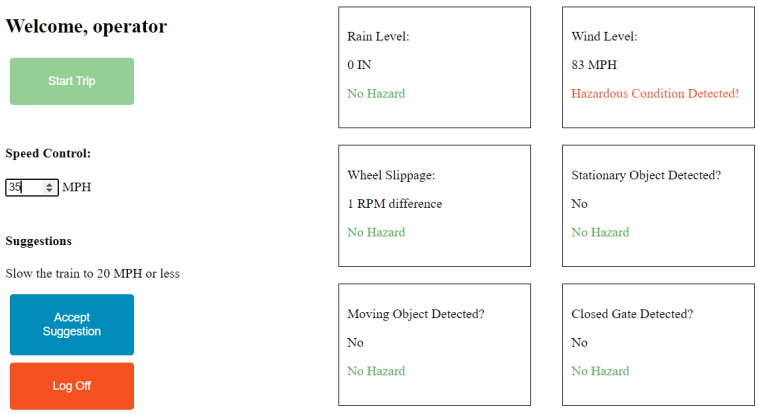
This requirement is satisfied by the units “MPH” under the Wind Level

R-13: If the speed of the wind is 45+ mph and the train speed itself is greater than 40mph, the IoT Display shall display a warning message in orange and a suggestion to lower speed to 40mph.



This requirement is satisfied by the orange display message in the Wind Level box and the suggestion “Slow the train to 40 MPH or less”

R-14: If the speed of the wind is 65+ mph and the train speed itself is greater than 20mph, the IoT Display shall display a warning message in red and a suggestion to lower speed to 20mph.



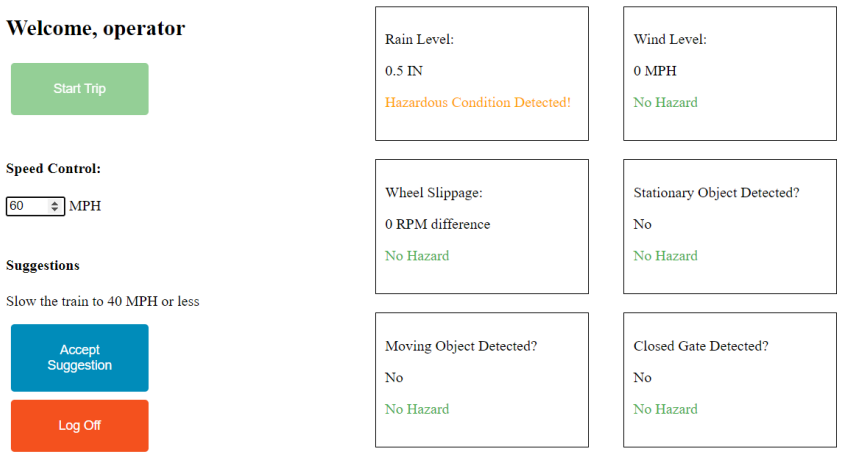
This requirement is satisfied by the red display message under "Wind Level" and the suggestion to "Slow the train to 20 MPH or less"

Detect the Amount of Precipitation

R-15: The precipitation sensor shall detect the amount of precipitation in inches that is in the area in which the train is operating.

This requirement is satisfied by the "IN" unit in the Rain Level box.

R-16: If the amount of rain reaches 0.3+ inches and the train speed itself is greater than 40mph, the IoT Display shall display a warning message in orange and a suggestion to lower speed to 40mph.



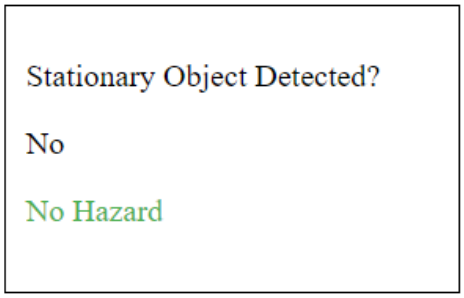
This requirement is satisfied by the orange display message in the Rain Level box and the suggestion "Slow the train to 40 MPH or less."

R-17: If the speed of the amount of rain is 0.6+ inches and the train speed itself is greater than 20mph, the IoT Display shall display a warning message in red and a suggestion to lower speed to 20mph.

This requirement is satisfied by the red display message under "Rain Level" and the suggestion "Slow the train to 20 MPH or less."

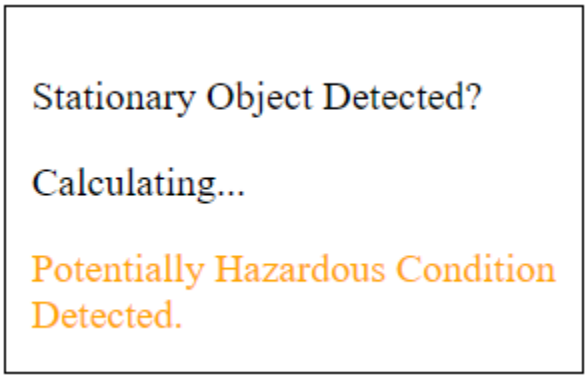
<u>Detect standing objects on the path with distance and suggest speed changes or brake</u>

R-18: The radar sensor shall detect whether there is a stationary object in the path of the train and how far away it is



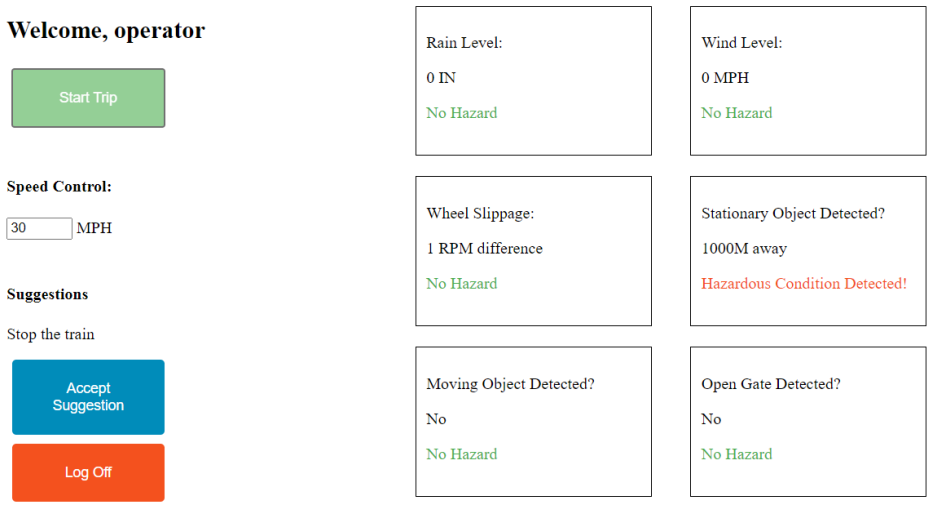
This requirement is satisfied by the “No” caption in the “Stationary Object Detected?” box.

R-19: For the first 2 positive radar readings, the IoT display shall show a warning message in orange alerting the operator that there is a potential hazard detected ahead.



This requirement is satisfied by the orange display message under “Stationary Object Detected?”

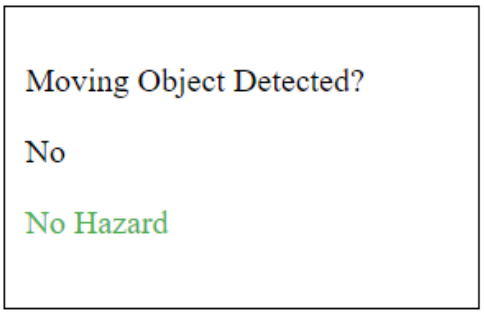
R-20: If the object ahead is determined by the IoT Engine to be stationary, subsequent positive radar readings shall be displayed as warnings in red and shall tell the operator to stop the train.



This requirement is satisfied by the red display message under "Stationary Object Detected?" and the recommendation to stop the train under suggestions.

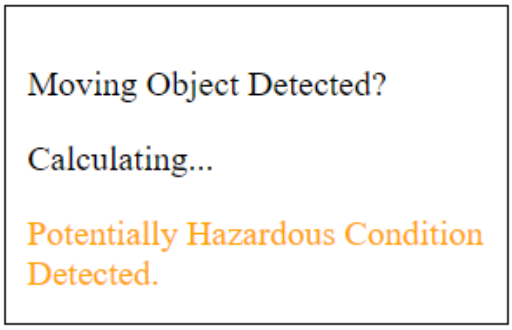
<u>Detect a moving object ahead and behind and their speed, direction of move and suggest speed changes or brake</u>

R-21: The radar sensor shall detect any other objects on the rails, and calculate the distance and speed of said object.



This requirement is satisfied by the "No" caption in the "Open Gate Detected?" box.

R-22: For the first 2 positive radar readings, the IoT display shall show a warning message in orange alerting the operator that there is a potential hazard detected ahead.



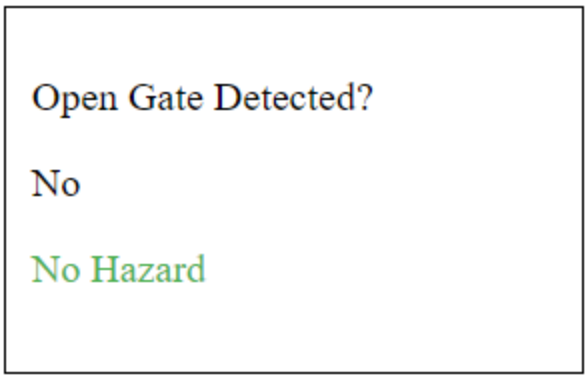
This requirement is satisfied by the orange display message under "Moving Object Detected?"

R-23: If the object ahead is determined by the IoT Engine to be moving, subsequent positive radar readings shall be displayed as warnings in red and shall tell the operator to stop the train.

This requirement is satisfied by the red display message under "Moving Object Detected?" and the recommendation to stop the train under suggestions.

<u>Detect gate crossing open or closed, distance and speed (based on GPS), suggest speed changes or brake</u>

R-24: The camera sensor shall be used to detect if the gate is open or closed, whether the red lights are flashing, the distance the train is away from the gate, and the speed.



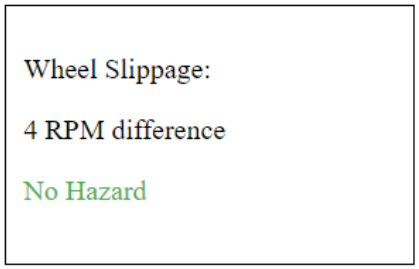
This requirement is satisfied by the “No” caption in the “Open Gate Detected?” box.

R-25: If the gate is in the open position, the IoT Display shall display a red warning and suggest the train to stop.

This requirement is satisfied by the red warning “” and the suggestion “Stop the train”.

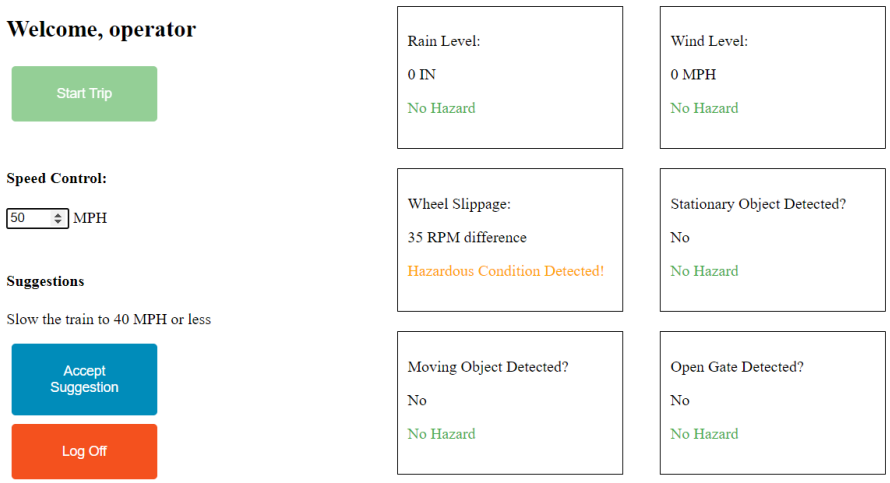
<u>Detect wheel slippage, and its amount using GPS data and the wheel RPM, suggest speed changes, if any, or brake</u>

R-26: The wheel slippage sensor shall be used to detect the amount of slippage occurring in the wheels, based on the wheels RPM and its projected RPM, and based on this data the conductor shall be informed of any speed changes that should be made based on the calculated RPM difference.



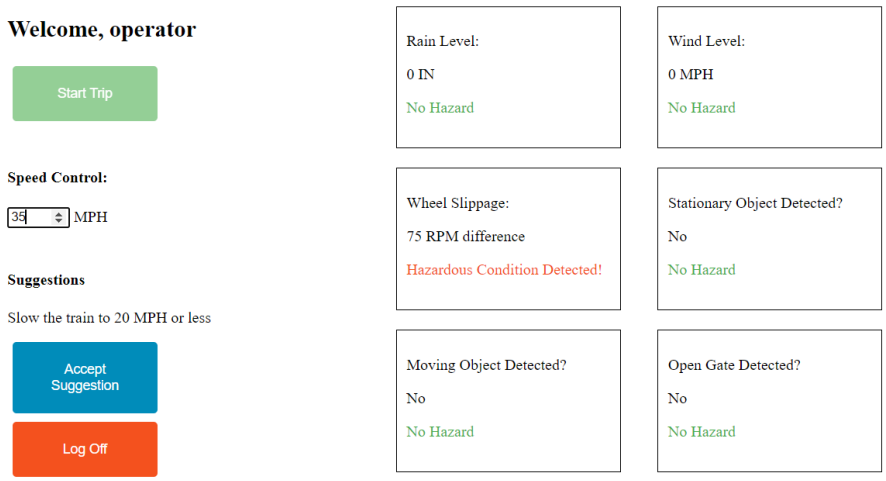
This requirement is satisfied by the slippage calculation displayed under the “Wheel Slippage” box.

R-27: If the difference between the projected speed and calculated wheel speed is between 20-50 RPM and the train speed itself is above 40mph, the IoT Engine shall inform the operator of wheel slippage using the IoT interface and display a caution message in orange and recommend the speed be decreased to 40mph.



This requirement is satisfied by the orange caution message displayed in the “Wheel Slippage” box and the suggestion to decrease speed to 40mph under the “Suggestion:” heading.

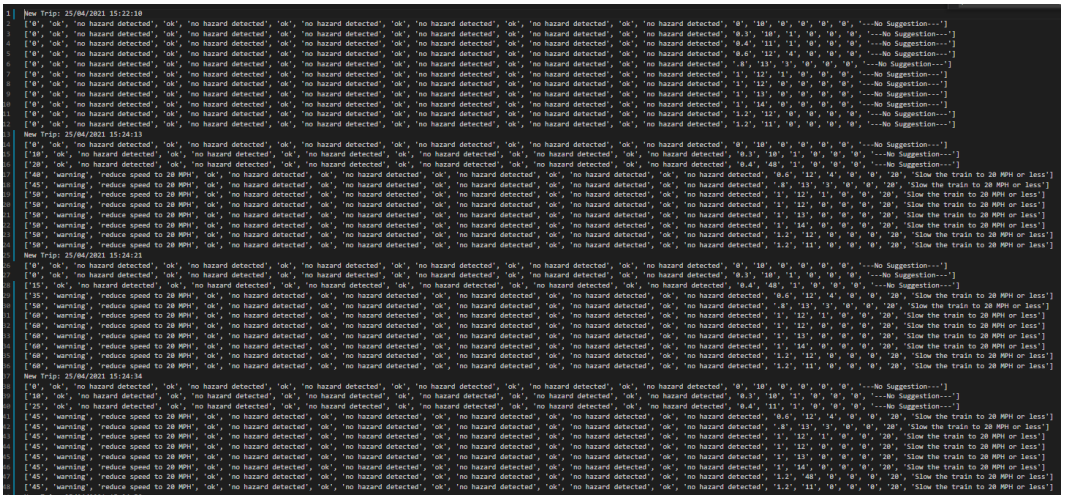
R-28: If the difference between the projected speed and calculated wheel speed is greater than 50 and the train speed itself is above 20mph, the IoT Engine shall inform the operator of wheel slippage using the IoT interface and display an extreme caution message in red and recommend the speed be decreased to 20mph.



This requirement is satisfied by the red caution message in the “Wheel Slippage” box and the suggestion to decrease speed to 20mph under the “Suggestion:” heading.

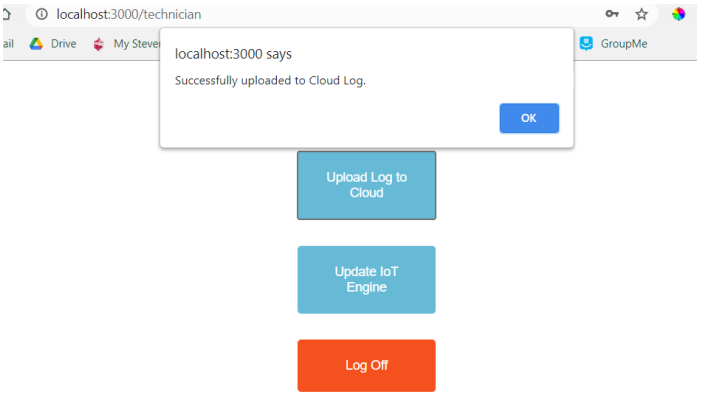
<u>Sensors shall store their readings in a Log File</u>

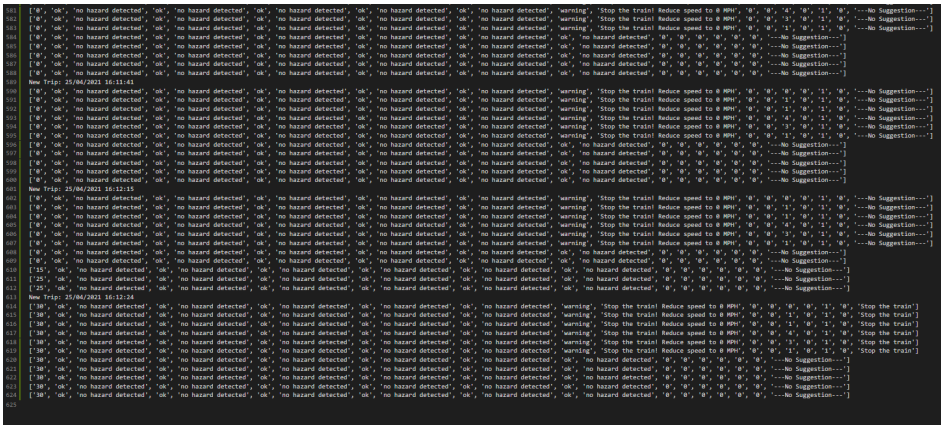
R-29: The Log File shall store the current speed, sensor readings, and IoT recommendations every 0.5 seconds during the span of the trip.

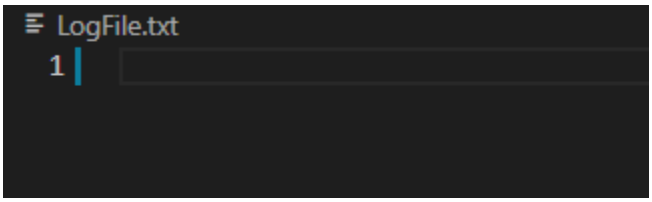


This requirement is satisfied by the Log File which stores all the above information for every trip.

R-30: The Log File shall be uploaded to the Cloud Log by the Technician when wifi connection is available, then cleared from the local database.

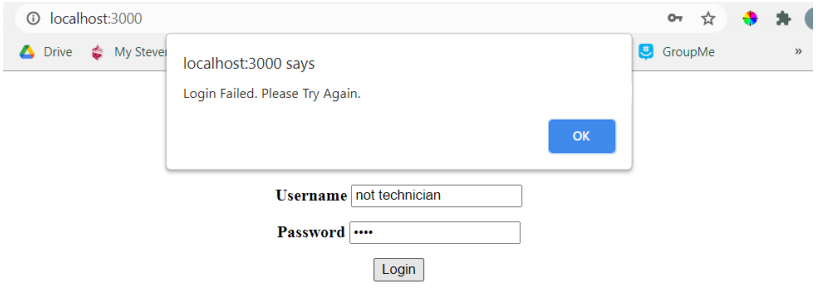






This requirement is satisfied by the “Upload Log to Cloud” button, which writes to the Cloud Log (top) and removes from the Log File (bottom).

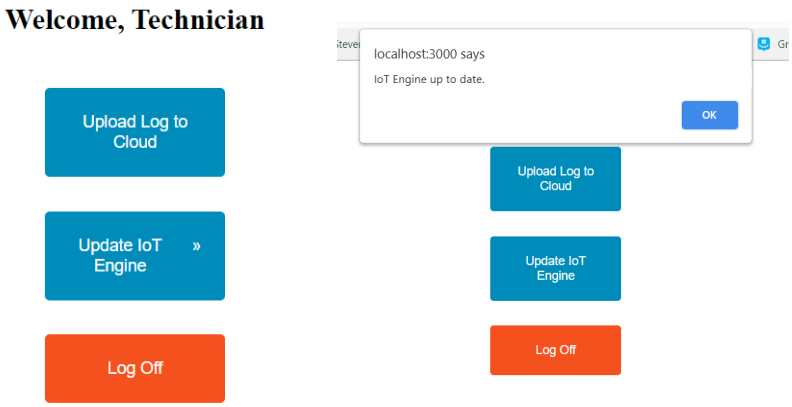
R-31: The Log File shall only be accessible with valid Technician login information.



This requirement is satisfied by the login validation check in the Login page.

IoT Software shall be updated by the Technician

R-32: When wifi connection is present, technician shall have option to update IoT Engine system software.



This requirement is satisfied by the “Update IoT Engine” button.

Works Cited

Admin. “Automatic Train Protection Railway Signalling Equipment.” *Railway Signalling Concepts*, 4 Sept. 2019, www.railwaysignallingconcepts.in/automatic-train-protection-railway-signalling-equipment/.

Bustos, Alejandro, et al. “EMD-Based Methodology for the Identification of a High-Speed Train Running in a Gear Operating State.” *MDPI*, Multidisciplinary Digital Publishing Institute, 6 Mar. 2018, www.mdpi.com/1424-8220/18/3/793/htm.

Du, Lei, et al. “Speed Calibration and Traceability for Train-Borne 24 GHz Continuous-Wave Doppler Radar Sensor.” *MDPI*, Multidisciplinary Digital Publishing Institute, 24 Feb. 2020, www.mdpi.com/1424-8220/20/4/1230/htm.

Ltd, PRC Rail Consulting. “The Railway Technical Website.” *Coach Parts | The Railway Technical Website | PRC Rail Consulting Ltd*, www.railway-technical.com/trains/rolling-stock-index-l/train-equipment/coach-parts.html

Pressman, R. and Maxim, B., 2015. Software engineering. New York [etc.]: McGraw Hill Higher Education.

Trend Micro . “Internet of Things (IoT).” *Definition*, www.trendmicro.com/vinfo/us/security/definition/internet-of-things.