Project: IoT "Hugs The Rails"

Team 9: Topham Hatts

Members: Eleanor Katsman, Alfredo Mendez, Simrun Heir, Murat Ulu

Table of Contents

Section 1: Introduction

1.1: Problem Statement

Currently, trains get their operation data from a small number of sensors that relay fuel level data or rudimentary environmental data, or from the experiential expertise of the conductor. Many railroad systems implement a process that updates train operation rules only when the train stops at its original station which, although practical in its use cases, is far from the best possible solution for ensuring safe train travel. A train running offline lacks the ability to implement critical operation rule updates when they are needed, such as in the case of unexpected weather conditions or other hazards that were previously not accounted for. In addition, a train disconnecting from the internet poses numerous issues for all parties involved in the locomotive process. When a train loses its ability to communicate through the internet, the train loses the ability to collect environmental data. Without environmental data, the train conductor cannot make certain decisions locally, as they do not have access to the proper data. Restricting the trains capability to make decisions locally without internet/server access compromises the safety and efficiency of the locomotive operation, and potentially cuts into the company's finances since the company must either enforce delays if the train loses internet connection or open themselves up to lawsuits if they risk operating under unknown conditions. Whether the locomotive is a passenger or freight train, these problems are relevant to the company, the locomotive users, and the company stakeholders, and must be addressed in order to keep the locomotive process as a whole working soundly.

1.2: Purpose of the Product (IoT Hugs the Rail)

The purpose of our product is to make the locomotive processes safer, less costly, and more efficient by developing a system that can collect vital local operation data regardless of whether the train is connected to Fog/Cloud Servers. The product will use IoT devices to capture data from locomotives, process the data, and communicate with the Locomotive Control System and Fog/Cloud server to ensure safe locomotive operation. The conductor will be able to enter simple commands to the system and receive transit information, as the access to this information not only helps our software run properly but also helps the conductor make informed decisions about the operation of the locomotive, making their job easier and decreasing the chance of potential errors in decision making due to a lack of data.

1.2.1: Importance and Value of Product to User

The safety of locomotive passengers is greatly important to the conductor and company. This can be improved by giving the conductor a set of commands to receive status updates or important information, and adjust the locomotive's travel if necessary. The product simplifies the conductor's job and streamlines safety precautions. The addition of multiple sensors to the locomotive adds to the efficiency of the product via better data collection, which is also important to the user. Increase in efficiency also leads to a decrease in cost.

1.3: Current System Used By Trains

Currently, locomotives lack an IoT implementation that can solve the issues we highlighted above. To evolve the current operation of these locomotives, we must add additional inputs on the locomotive in case they are not already fitted with speed and location data sensors. We must then be able to access that data in order to have a link to an IoT device for data communication and data processing.

1.4: Approach to the Solution

To solve the issues posed by current locomotive systems, there are a few key processes that IoT must take into account then implement. First, research should be conducted into what the current procedure in trains is for making decisions via cellular and wifi connectivity, as well as the points of failure within that current system that we must address. Then, we must identify the necessary data that needs to be collected for the train to operate properly both online and offline, and choose locomotive sensors accordingly in order to resolve these points of failure.

After gathering a list of potential sensors and components to use, we must note the ideal locations for our data collection on the locomotive as well as the ideal method for downloading the latest rules of operations (updating the train software) and communicating with external processes (sending data between trains) from the Fog/Cloud into the IoT Engine.

Pairing these sensors with an IoT connected device would be the next step, as the implementation of the sensors in order to collect precise and correct data for our software is necessary before moving forward. Some key data points that would be mandatory as input to our software would be location data, speed and velocity data, as well as general environment data, whether it is directly related to the environment of the current locomotive or related to a nearby locomotives environment (for example, if a further train is stopped on the tracks, we must account for this and communicate between trains to ensure both locomotives arrive at their destination safely and in the most efficient manner possible).

After we are able to access this data and provide accurate input into our software, we must create an algorithm that uses the information from the Fog/Cloud to make efficient and safe decisions locally using whatever data is available. This algorithm should also be able to take the input data and create a readable and straightforward output for the conductor. If a train goes offline for a certain period of time, it should be able to run safely utilizing its local data, and be able to store this data for future upload when IoT/internet connectivity resumes.

1.5: Product Review

1.5.1: Overview of the team

Team of Computer Science students with experience with project development, management, and programming.

Eleanor Katsman:

Languages: Python, Scheme, Java

Three major qualifications: Task Oriented, Curious, Outgoing

Alfredo Mendez:

Languages: Python, C++, Java, JS, Kotlin, Scheme

Three major qualifications: Coding, Detail-Oriented, Problem Solving

Murat Ulu:

Languages: Python, Java, C++, JS

Three major qualifications: Problem Solving, Task-Oriented

Simrun Heir:

Languages: Python, Java, C++, JS

Three major qualifications: Analysis, Patience, Communication

Section 2: Overview

2.1 Timeline and Process Model

2.1.1: Timeline

By 2/27: Communication/Planning/Modeling: Find a way to implement sensors. Identify key points where LCS, Back Offices, and Locomotive connect and interact

By 3/5: Modeling/Construction: Implement Sensors (speed, location data), IoT device

By 3/12: Construction: Create algorithm to ensure safe locomotive travel

By 3/19: Construction: Create algorithm to output data to conductor

By 4/1: Deployment: Clean up software / work out bugs

2.1.2: Process Model

The **Unified Process Model** was chosen because of how varied the potential solutions to the project are. The problem that our IoT has to solve is straightforward, but there are many factors and different implementations that can cause the software development process to change and pivot in a different direction at any time, an aspect of development that the unified process model takes into account by allowing for continuous revision and implementation where need be. In addition to the variability, because this IoT system is built upon and aids the current system in which trains operate through the Locomotive Control System (LCS) and wifi and cellular connection, this project has a maintenance aspect that is especially instrumental in regards to the current system. The Unified Process Model suggests a process flow that is iterative and incremental with nearly all of the requirements documented, which coincides with the parameters of our IoT system. It is also a risk-focused model, meaning safety and maintenance will be emphasized during the development process. In most projects, the production phase may call for work to begin on the next software increment, meaning that the five Unified Process Model phases can be staggered and overlapped to increase workflow. These reasons together influenced the decision to use the Unified Process Model.

2.2: Hazardous Conditions to Address

Extreme weather conditions can pose significant hazards to a train's operation. Ice, snow, and heavy rain can all cause wheel slippage. Snow and rain can also hamper visibility, and snow in the train wheel could freeze solid. High winds can destabilize the train, create airborne debris, and, at significant enough power (such as in the case of a tornado), even knock the train off the rails.

There are also significant hazards associated with railroad conditions. Debris on the track can damage the train or sensors, while people or animals on the track pose safety issues in both directions, as well as opening the company up to lawsuits if a person or their pet gets hurt or killed by the train. Damage to the railroad itself can also pose significant risk. The most common cause of railroad accidents is broken rails and welding; buckled tracks are also a significant accident cause.[1] Even undamaged tracks can cause accidents in some situations. Unfenced bridges are dangerous to cross even without extenuating weather conditions, and changing track gauges -- aka, the width of the track -- are another leading cause of train accidents.

2.3: Potential Sensors

Before deciding any sensors that can be used, a good understanding of the hazards and necessary information needed for the train to operate under optimal conditions (no hazards while connected to wifi/cellular signals) is needed. Based upon these two criteria certain sensors are necessary.

For example, we can use a wheel rotation sensor to count how many rotations per minute a train wheel is spinning, and use that to calculate the train's alleged speed and the distance it has traveled. However, issues like wheel slippage tend to occur, so we could use the wheel rotation sensor in tandem with a GPS to ensure that the distance traveled corresponds to each other (if distances don't correspond, wheel slippage has occurred, and we can relay that info to the train conductor). GPS is also good for communicating location data between trains when internet connectivity is available, as we can relay sensor data to trains nearby to warn of dangerous weather conditions or track obstructions.

We would also use a Radar Sensor to alert of an object on the track or for getting general environment information (i.e. Detect surroundings to see if there could be potential collisions with objects, such as falling trees or landslides approaching tracks). Side and front sensors could also be added to the train to get additional environment data, which addresses hazards like unfenced bridges and other height/width related issues.

Some sensors have more straightforward applications, such as a barometer to warn of potential storms nearby. All of these sensors should speak to our program, then output the applicable data to the LCS so the train conductor can take action accordingly.

Additionally it is important to note that because the purpose of IoT is to deal with the issues posed when internet connectivity to the train is lost, it would be beneficial to have some sensor or indicator within the IoT hardware that detects when internet connection is reestablished. This allows for IoT's presentation of information to be kept to solely what is necessary, so that the conductor is not overwhelmed with duplicates of the same information.

Overview of Sensors:

- Wheel rotation sensor
    - Counts how many rotations per minute, can calculate speed and distance traveled
- GPS
    - Distance traveled (compare with wheel rotation sensor to check for wheel slippage)

[1] https://www.baumhedlundlaw.com/blog/2016/september/the-most-common-train-accident-causes/

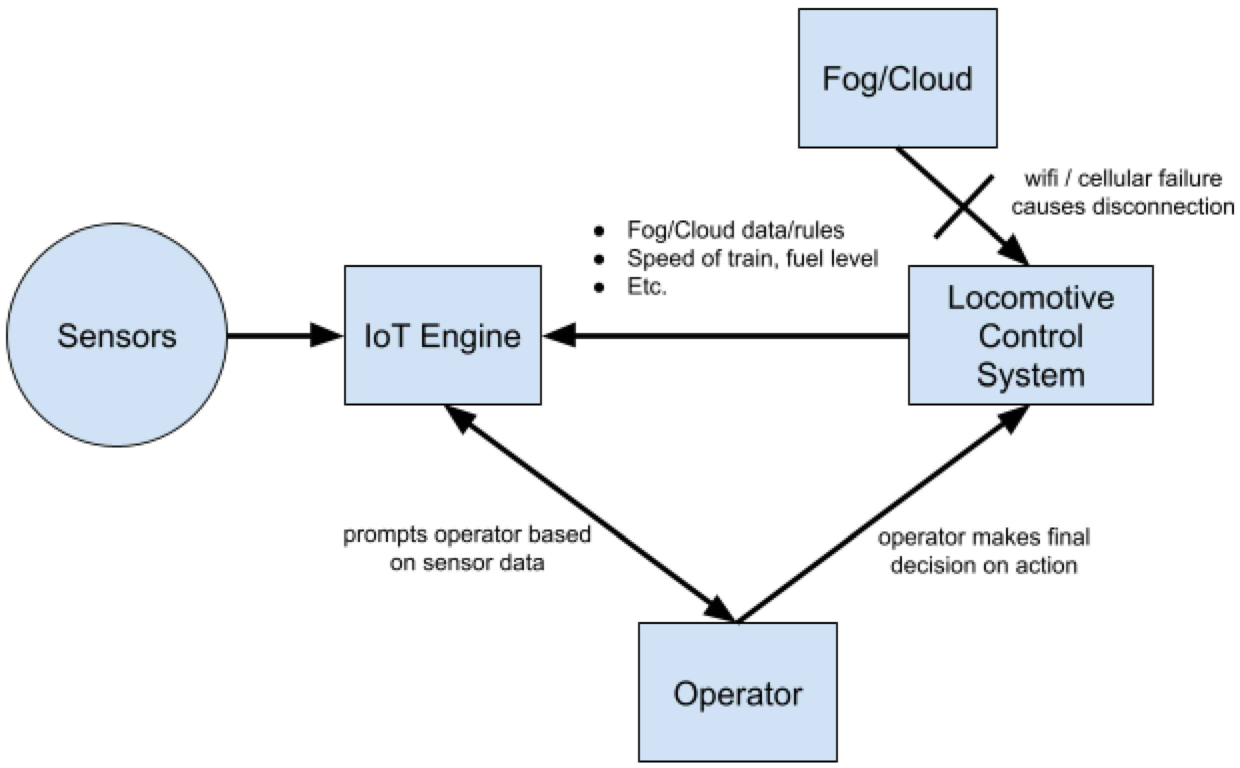
- Communicate location data (good for communicating with other stopped or moving trains)
- Front Radar Sensor
  - Detect if something is on the track, general environment information
  - Detect if something is getting too close to train from above such as falling trees
- Side Radar Sensor

  For unfenced bridges and other height or width related issues
- Rear Radar Sensor
  - Detect if something is approaching the track from behind.
- Barometer for air pressure
  - Severe weather conditions
- Sensor/indicator for wifi/cellular connectivity
  - Specifically for IoT, allows for smooth transition between activation and deactivation of certain sensors for when IoT is necessary

2.4: Description of Software Process (Conceptual Architecture)

2.4.1: Software Overview and Desired System Model

The goal for our software is to assist the conductor without replacing them. The conductor will still be essential and important to the locomotive system. The conductor's job will be simplified from detecting problems and dealing with them TO just dealing with them. Sensors will be used to capture data that is essential to determine whether the conductor needs to take some sort of action and what that action would be. Sensors will communicate with the conductor via some type of interface. Our system will also be in charge of suggesting any action(s) to the conductor corresponding to an alert system. In addition to the onboard sensors, our system will also periodically take in information from the cloud in the event of a disconnection from a wifi/cellular connection. Location data of all trains and data gathered by all trains will be uploaded to the cloud and retrieved by other trains. Only important and essential information will be displayed to the conductor. All final decisions will be made by the conductor.


Fog/Cloud
wifi / cellular failure causes disconnection
Fog/Cloud data/rules
Speed of train, fuel level
Etc.
Sensors
IoT Engine
Locomotive Control System
prompts operator based on sensor data
operator makes final decision on action
Operator

2.4.2: Rules of Operation Back-up System

In order for all locomotives to work correctly and consistently, each locomotive will need to be updated with a set of rules that will tell the conductor the optimal and appropriate way to conduct the train; these sets of rules will be called the “Rules of Operation”.

The Rules of Operation will be updated whenever necessary whether the locomotive is stationed or traveling. This is assuming that all trains are connected to the cloud at all times (which is already the case with some locomotive transit systems). Information like weather data, recommended speed, closed tracks/routes, alternate routes to take, or a delayed department of another train will be included in the Rules of Operation.

On board sensor data would not be essential to the locomotive while it is connected to the Cloud as most of the necessary information will be gathered from the Rules of Operation which is obtained through the cloud. However, if / when a locomotive’s LCS becomes disconnected from the cloud, the Locomotive will be more reliant on the onboard sensors through our IoT system.

For example, since a locomotive will not have access to any reports of obstacles on the track once it is disconnected from the Cloud, LiDAR sensors will be activated. Since weather data will be unavailable, the IoT system will try to predict the weather using humidity sensors and other weather related sensors.

In addition to the increase of sensor data use, once a locomotive is disconnected from the cloud, it will rely either on the most recent update to the Rules of Operation or from a default ‘Rules of Operation’ file.

Section 3: Requirements

3.1: Non-Functional Requirements

3.1.1: Hardware

3.1.1.1: Sensors Installation

HR-1	One wheel rotation sensor must be placed on each of the two back wheels of each respective train car on the train
HR-2	One GPS sensor must be placed on the top of the frontmost train car
HR-3	One backup GPS sensor must be placed on the top of the backmost train car
HR-4	One forward-facing Radar Sensor must be placed on the top of the front of the frontmost train car
HR-5	One backup forward-facing Radar Sensor must be placed on the top of each respective side of each respective train car on the train
HR-6	One barometer sensor must be placed on the top of the frontmost train car
HR-7	One backup barometer sensor must be placed on the top of the frontmost train car
HR-8	One Wifi / Cellular connectivity sensor must be placed in IoT central unit
HR-9	One backup Wifi/Cellular connectivity sensor must be placed on the top of the frontmost train car

3.1.1.2: Hardware

HR-10	IoT central unit must be placed in the frontmost train car (conductor train car)
HR-11	One 1080p 23inch screen will be installed next to LCS to display IoT information

HR-12	A 0.25in thick clear plastic panel must be placed over the screen to prevent screen breaking
HR-13	Screen must be connected to IoT central unit
HR-14	Standard QWERTY keyboard must be installed next to IoT info display
HR-15	Keyboard must have a track point mouse equipped for navigation of IoT display
HR-16	Keyboard must be connected to the IoT central unit
HR-17	Trackpad must be installed below the keyboard for navigation
HR-18	Trackpad must be connected to the IoT central unit
HR-19	Time Sensitive Networking Router (TSNR) must be used as interim data storage for our software

3.1.1.3: Power for IoT systems and sensors

HR-20	IoT must be linked to and powered by existing power source for LCS
HR-21	IoT must turn on once LCS obtains power
HR-22	All wheel rotation sensors must be connected to and draw power from existing power infrastructure in each train car
HR-23	GPS sensors must be connected to and draw power from existing power infrastructure in the frontmost train car and the backmost train car respectively
HR-24	All Radar sensors must be connected to and draw power from existing power infrastructure in each train car
HR-25	Barometer sensors must be connected to and draw power from existing power infrastructure in the frontmost train car
HR-26	Wifi / Cellular sensors must be connected to and draw power from existing power infrastructure in the frontmost train car and the backmost train car respectively

3.1.1.4: Sensors on outside of train must be waterproof to an extent

HR-27	Sensors must be able to withstand heavy rain.
HR-28	Sensors must be able to withstand 140mph wind speeds.
HR-29	Sensors must be able to last for 500 hours under water at a time.

3.1.1.5: IoT must be supported by a hardwired LoRaWan network

HR-30	Data transferring wiring system must be linked between each train car for local communication
HR-31	Sensors must have wired connection into data transferring wiring system
HR-32	All data and wiring must route to IoT in frontmost train car
HR-33	IoT must be able to withstand constant multiple data inputs through wiring system
HR-34	IoT must be supported by the LoRaWan network

3.1.1.6: Hardware should have security that prevents from physical tampering

HR-35	Sensor wiring connections shall not be physically exposed
HR-36	Sensors must have physical barriers protecting them
HR-37	Barrier should be 0.5in impact resistant plastic dome covering the maximum area of sensors without inhibiting sensor function

3.1.2: Performance

3.1.2.1: Conductor must be able to obtain updated data

PR-01	When the conductor logs into the IoT system, the IoT system must be operational within 2 seconds
PR-02	Data that is being displayed must be calculated in less than 1 second

PR-03	If a sensor becomes disabled a notification will be displayed on the IoT screen within 1 second of failure
PR-04	If a sensor is malfunctioning (sensor data is failing to be obtained by software consistently but not completely), a warning notification will appear on display
PR-05	The speed of the train must be calculated at least once a second
PR-06	The normal operation of the locomotive must be able to continue if IoT fails
PR-07	The TSNR will collect all sensor data so all data is stored in one place

3.1.3: Reliability

3.1.3.1: Locomotive must communicate data with necessary devices

RR-01	The locomotive must be connected to a LoRaWan network gateway node
RR-02	Locomotive must send IoT processed data to the network via the nearest gateway node
RR-03	Each locomotive must send their processed sensor data to the network every 5 seconds
RR-04	The temperature surrounding the the train will be sent to the network
RR-05	The probability of weather events surrounding the train will be sent to the network
RR-06	The locations of encountered objects will be sent to the network
RR-07	The locations of wheel slippage occurrence will be sent to the network
RR-08	The size and possible danger level of an encountered object will be sent to the network

3.1.3.2: Sensor data must always be obtainable to the conductor

RR-09	Locomotive must update IoT-processed data every 2 seconds
RR-10	Display system should always be working when sensor data is in use
RR-11	Data from multiple sensors of the same kind should be available
RR-12	If one sensor becomes of unavailable/damaged, use ONLY the data of the fully functioning sensor/sensors
RR-13	Speed will be gathered from the GPS module
RR-14	Radar sensors will be used to calculate speed in the case the GPS and wheel rotation sensors are unavailable

3.1.4: Security

3.1.4.1: Software should have measures in place to deter digital tampering

SR-01	A Sha-256 hashing algorithm must be implemented into IoT network for password storage
SR-02	IoT shall only be accessed by an user ID and password
SR-03	The system administrator shall have their own user ID and password
SR-04	System administrator must be able to access all data sent to IoT (processed data), update the system, and update the system operation rules
SR-05	Conductor shall have their own password and ID
SR-06	Updates to software should only be applied when the train is docked and when the command is given by a system administrator

3.2: Functional Requirements

3.2.1: Object Detection

3.2.1.1: Detect both stationary and moving objects in front of and behind the train and suggest action to the conductor for braking or increasing/decreasing the speed

ODR-01	A Radar Sensor that can detect objects at a distance of at least five miles must be attached to the front and to the back of the train.
ODR-02	The front and back Radar Sensors shall detect objects on the track every 0.25 seconds
ODR-03	The Radar Sensors shall send data to the IoT every 0.25 seconds
ODR-04	A third party software[2] shall process the Radar data and create an image of the track ahead.
ODR-05	A third party software shall analyze the data to determine which objects are on the track and which objects are off of it.
ODR-06	A third party software shall calculate the size (in feet) and distance from the train (in feet) of any objects on the track.
ODR-07	If an object larger than 1.5 feet is detected on the track, IoT will display a notification on the screen.
ODR-08	If the distance between the train and an object on the track consistently decreases over four consecutive readings, the IoT will register the object as “approaching”
ODR-09	The IoT will calculate risk for an approaching object in front of the train based on the Time Until Collision which is calculated with the formula: distance/speed
ODR-10	If the Time Until Collision is less than 200 seconds, the IoT will display a “red warning” that instructs the conductor to stop the train.
ODR-11	If the Time Until Collision is less than 260 seconds, the IoT will display a “yellow warning” that instructs the conductor to brake to half-speed.
ODR-12	If the Time Until Collision is more than 260 seconds, the IoT will display an “orange warning” that instructs the conductor to match the object’s speed.

3.2.1.2: Detect gate crossing open/closed, distance and suggest speed change or brake to stop to conductor

[2] https://usradar.com/gpr-software/radar-studio/

ODR-13	IoT must be able to detect if a gate is open or closed
ODR-14	IoT will define gate open as gates are up, allowing objects to come onto the track
ODR-15	IoT will define gate closed as gates are down, preventing objects from coming onto the track
ODR-16	IoT will notify the conductor a crossing is approaching if gates are detected
ODR-17	IoT will notify the conductor if the gates detected are closed
ODR-18	IoT will notify the conductor if the gates detected are open
ODR-19	IoT will prompt the conductor to brake if gates are open
ODR-20	IoT must specify current speed, suggested speed, and difference between the two

3.2.1.3: Detect wheel slippage using GPS speed data and comparing it with wheel RPM and suggest to conductor change speed or break

ODR-21	GPS modules must always be working
ODR-22	On wheel sensors must always be on
ODR-23	Wheel slippage algorithm must have wheel radius as a pre-stored value
ODR-24	GPS module will send speed data to IoT
ODR-25	If GPS modules stop working, the speedometer will send speed data to IoT
ODR-26	On wheel sensors will send angular velocity of wheel to IoT
ODR-27	Wheel slippage calculation will use the average of all the data from the wheel sensors
ODR-28	Wheel slippage calculation will use speed data

ODR-29	Wheel slippage calculation will use pre-stored wheel radius
ODR-30	Wheel slippage will be calculated using SAE J670 slip ratio equation
ODR-31	IoT recommends action based on severity level of wheel slippage
ODR-32	If any wheels have slip ratio under 10% notify the conductor to put sand on tracks
ODR-33	If slip ratio is under 7.5% notify the conductor to put sand on tracks and slow down
ODR-34	If slip ratio is under 5% notify the conductor to put sand on tracks and to brake aggressively

3.2.1.4: Will have to know what sensors are needed specifically when wifi/cellular connection is disconnected

ODR-35	A predetermined set of sensors will automatically be activated once wifi/cellular connection is lost
ODR-36	Predetermined set of sensors can be changed by administrator
ODR-37	Conductor can turn on or off any sensor
ODR-38	Recommended sensors will also be displayed based on previously gathered data

3.2.2: Weather Detection

3.2.2.1 Temperature

WDR-01	Each on board weather detection module will send its data to our IoT Systemat a consistent rate
WDR-02	The average of all the temperature readings will be calculated
WDR-03	The average temperature will be displayed to the conductor
WDR-04	The average temperature will be sent to the network

3.2.2.2 Air Pressure

WDR-05	Each on board weather detection module will send its data to our IoT System at a consistent rate
WDR-06	The average of all the pressure readings will be sent to our IoT Systemat a consistent rate
WDR-07	Our software will predict what bad weather will be present based on the average air pressure
WDR-08	The weather prediction will be displayed to the conductor
WDR-09	The weather prediction will be sent to the network

3.2.3: Data Processing and IoT Display

3.2.3.1: IoT Display

DPR-01	The display will be updated every 1 second.
DPR-02	Screen must have brightness adjustability
DPR-03	Display will be divided into multiple sections
DPR-04	Display will have a section to display IoT emergency messages
DPR-05	Display will have a section to display IoT suggested actions
DPR-06	Display will have a section to display wifi/cellular connection status
DPR-07	Display will have a section to display the condition of the train
DPR-08	Display will have a section to display the weather conditions

3.2.3.2:Processing data from sensors

DPR-09	Processing algorithms will use sensor data and LCS' downloaded copy of rules of operation

DPR-10	The necessary processed sensor data will be displayed on the IoT screen.
DPR-11	When processing sensor data software will use the data gathered by all sensors
DPR-12	IoT will store all processed data in variables
DPR-13	The TSNR will hold all raw sensor data before it is sent to the IoT System.

3.2.3.3: Display result of data processing algorithms to conductor

DPR-14	Variables in main file storing processed data shall be read by display software
DPR-15	Display software shall display processed data to the conductor
DPR-16	Processed data shall be displayed visually on a screen
DPR-17	If an emergency is detected a visual prompt will be displayed

3.2.3.4: Give the conductor the ability to make adjustments and decisions to the LCS based on the result of processing the data

DPR-18	IoT shall not be able to change operation conditions in LCS
DPR-19	Suggested actions based on IoT processed data shall be presented to conductor along with processed data

Section 4: Requirement Modeling

4.1: Use Cases

Use Case 1: Conductor logs into their account

Primary actor: Conductor

Secondary actor: IoT Display

Goal in context: Start the IoT System for the conductor.

Preconditions:

1. Locomotive is on.
2. IoT display is on and working.

Trigger: Conductor enters their username and password.

Scenario:

1. The screen displays the login page.
2. The conductor enters the correct username and password.
3. IoT screen displays Conductor view (data, warning, average sensor readings, etc.).

Exceptions:

1. Conductor enters wrong username and or password.
    a. Redisplay login page.
    b. Display error message “Invalid username or password received”.

Use Case 2: Administrator logs into their account

Primary actor: Administrator

Secondary actor: IoT Display

Goal in context: Start the IoT System for the administrator.

Preconditions:

1. Locomotive is on.
2. IoT display is on and working.

Trigger: Administrator enters their username and password.

Scenario:

1. The screen displays the login page.
2. The administrator enters the correct username and password.
3. IoT screen displays Administrator view (button to request log).

Exceptions:

1. Administrator enters wrong username and or password.

a. Return to and display login page.
b. Display error message “Invalid username or password received”.

Use Case 3: IoT System detects wheel slippage

Primary actor: IoT Wheel sensors

Secondary actor: IoT System, GPS Sensor, IoT Display

Goal in context: Give the Conductor a warning and advice on what to do when wheel slippage occurs.

Preconditions: Locomotive is on and Conductor is logged in.

Trigger: Average speed calculated by Wheel Sensors and GPS speed are not equal.

Scenario:

1. GPS sensor sends speed of locomotive to IoT System.
2. All Wheel Sensors send RPM to the TSNR.
3. The IoT System pulls Wheel Sensor data from the TSNR.
4. The IoT System calculates the average RPM.
5. The IoT System reads the diameter of the wheel from the “Rules of Operation” file.
6. The IoT System calculates the speed of the locomotive using the GPS speed data.
7. The IoT System calculates angular velocity of wheels using average of all wheel’s RPMs.
    a. Angular velocity = avg. wheel RPM * 6
8. The IoT System calculates the slip ratio using the SAE J670 slip ratio equation.
    a. $Slip\ Ratio\ \% \ = \ (\frac{Angular\ Velocity\ *\ Wheel\ Radius}{Speed\ of\ Train} - 1)\ *\ 100\%$
9. The IoT System will display the slip ratio and the recommended action for that slip ratio.
    a. If the slip ratio is <10% , notify the conductor to put sand on tracks.
    b. If the slip ratio is <7.5%, notify the conductor to put sand on tracks and slow down.
    c. If the slip ratio is <5%, notify the conductor to put sand on tracks and slow down aggressively.

Exceptions:

1. GPS sensor returns error when sending speed of locomotive to IoT System.
    a. IoT screen displays error message “error in retrieving GPS sensor data”.
2. >25% of wheel sensors return error when sending RPM to the IoT System.
    a. IoT screen displays error message “error in retrieving sufficient wheel sensor data”.

Use Case 4: IoT detects a dangerous object on the track

Primary actor: Radar Sensor

Secondary actor: IoT System, Radar Sensor, IoT Display, Object Detection Software (ODS), GPS Sensor

Goal in context: Display a warning to the conductor that alerts him of a hazardous object on the track.

Preconditions: Locomotive is on and Conductor has logged in.

Trigger: Object detection software detects an object on the track that is more than 1.5 feet tall.

Scenario:

1. Radar Sensor sends a frame to ODS.
2. ODS detects an object larger than 1.5 ft. tall.
3. ODS sends the distance and speed of the object to the TSNR.
4. The IoT System pulls ODS data from the TSNR.
5. The IoT System calculates Time Until Collision using Speed of the locomotive, distance to the object, and speed of the object.
6. IoT System will display information about the object to the Conductor and will give appropriate warning
    a. If the Time Until Collision is less than 200 seconds, the object is bigger than 1.5 feet tall, and the object is in front, the IoT will display "RED WARNING: OBJECT DETECTED AHEAD - Stop train.”
    b. If the Time Until Collision is less than 260 seconds but greater than or equal to 200 seconds, the object is in front, and the object is bigger than 1.5 feet tall, the IoT will display "ORANGE WARNING: OBJECT DETECTED AHEAD - Apply break to half-speed.”
    c. If the Time Until Collision is less than 260 seconds, the object is behind, and the object is bigger than 1.5 feet tall, the IoT will display "YELLOW WARNING: OBJECT DETECTED BEHIND - Match speed of rear object - obj.getSpeed().” where obj.getSpeed() is a call to the given objects speed parameter value.

Exceptions:

1. Radar Sensor returns error when sending data to IoT System
    a. IoT screen displays error message “error in sending Radar Sensor data”
2. Object detection software returns error when trying to retrieve Radar Sensor data from IoT System
    a. IoT screen displays error message “error in object detection software: error when providing Radar Sensor data”

Use Case 5: IoT System detects a gate

Primary actor: IoT System

Secondary actor: Object Detection Software (ODS), IoT Display

Goal in context: Display a warning to Conductor that a gate is open.

Preconditions: Locomotive is on and Conductor is logged in.

Trigger: Object Detection Software detects a gate.

Scenario:

1. Radar sends a frame to TSNR.
2. ODS detects that there is a gate in front of the train.
3. ODS determines if the gate is open or closed.
4. ODS sends the status of the gate along with the distance between the gate and the locomotive to the TSNR, which relays the information to the IoT System.
    a. If the gate is open, IoT System warns the Conductor that the locomotive is approaching a gate that is open and recommends him an action.
        i. Display warning "OPEN GATE DETECTED: Please use horn and apply breaks."
    b. If the gate is closed, IoT System warns the Conductor that the locomotive is approaching a gate that is closed and recommends him an action.
        i. When the train is less than a mile away from the gate, display the warning "HONK HORN FOR 15 SECONDS: Closed gate is 1 mile ahead."
        ii. When the train is approaching the gate (less than or equal to 400 meters away from gate), display the warning "HONK HORN FOR 6 SECONDS: Passing through closed gate."

Exceptions:

1. Rules of Operation specify to ignore the status of gates.

Use Case 6: IoT System warns the Conductor of present/upcoming bad weather conditions

Primary actor: IoT System

Secondary actor: Weather sensors, IoT Display

Goal in context: Alert Conductor of upcoming hazardous weather.

Preconditions: Locomotive is on and Conductor is logged in.

Trigger: IoT System detects average air pressure of 30 Hg or lower.

Scenario:

1. The two Weather Sensors send an air pressure reading to the TSNR
2. The IoT System pulls the air pressure readings from the TSNR.
3. The IoT System calculates the average in Hg (Inches of Mercury).
4. The IoT System displays a warning about the weather if necessary.
    a. If average Hg is 29.8 or below, warn the Conductor that rain storm / snow is incoming.
    b. If average Hg is above 29.8 and below 30.2; Send that rain is incoming
    c. If average Hg is above 30.2 , Nothing is displayed as a warning display.

Exceptions:

1. Weather sensors return errors when sending data to IoT System.
    a. IoT screen displays error message “error in receiving weather sensor data”.

Use Case 7: IoT System detects network disconnection

Primary actor: IoT System

Secondary actor: Wifi / Cellular connectivity sensor, IoT Display

Goal in context: Alert Conductor of IoT network disconnection.

Preconditions: Locomotive is on and Conductor is logged in.

Trigger: IoT System receives data that there is no current Wifi / Cellular connection.

Scenario:

1. Wifi / Cellular connectivity sensors send network connectivity reading to the TSNR
2. The IoT system pulls network connectivity reading from TSNR.
3. IoT System detects that there is currently no Wifi/Cellular connectivity for IoT system.
4. Screen displays “Wifi / Cellular” on/off area as off.
5. Screen displays “Wifi / Cellular” on/off area as on when Wifi/Cellular connectivity is reestablished.

Exceptions:

1. Wifi / Cellular connectivity sensor returns error when sending data to IoT System.
    a. IoT screen displays error message “error in receiving Wifi/Cellular connectivity data”.

Use Case 8: Administrator requests log of activity

Primary actor: Administrator

Secondary actor: IoT System, IoT Display

Goal in context: Provide Administrator with log activity.

Preconditions: Locomotive is on and Conductor is logged in.

Trigger: Administrator requests log information through terminal.

Scenario:

1. Administrator logs in.
2. Administrator opens a terminal window.
3. Administrator requests log information along with start and end date.
4. IoT System returns requested log entries.
5. Administrator saves log data to an external USB device.

Exceptions:

1. Administrator enters wrong username and or password.
    a. Redisplay login page.
    b. Display error message “Invalid username or password received”.

Use Case 9: User logs out of their account

Primary actor: Conductor, Administrator

Secondary actor: IoT Display

Goal in context: Return to the log-in page for IoT HTR Software

Preconditions:

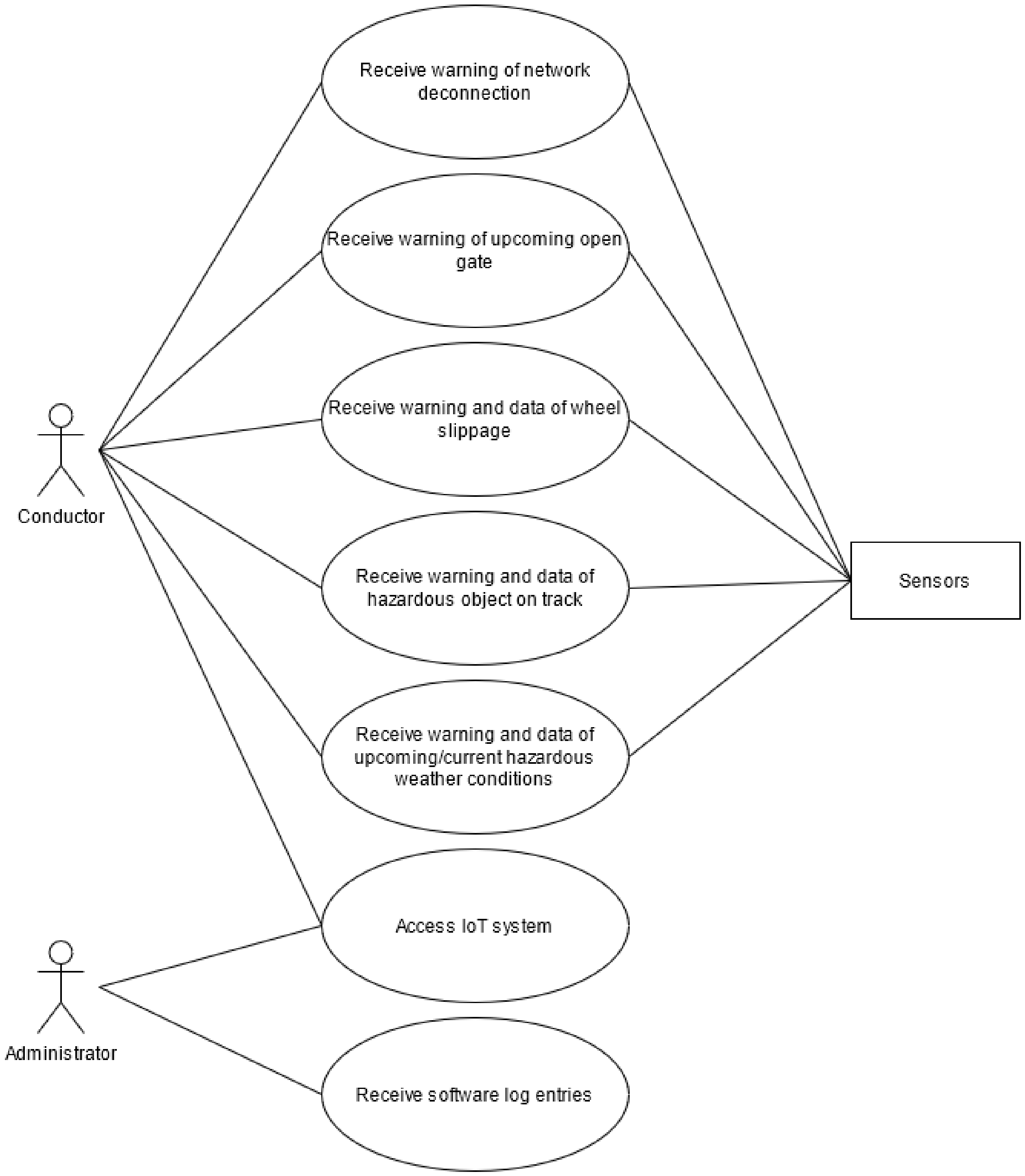
3. Locomotive is on.
4. IoT display is on and working.

Trigger: Conductor presses “log-out” button.

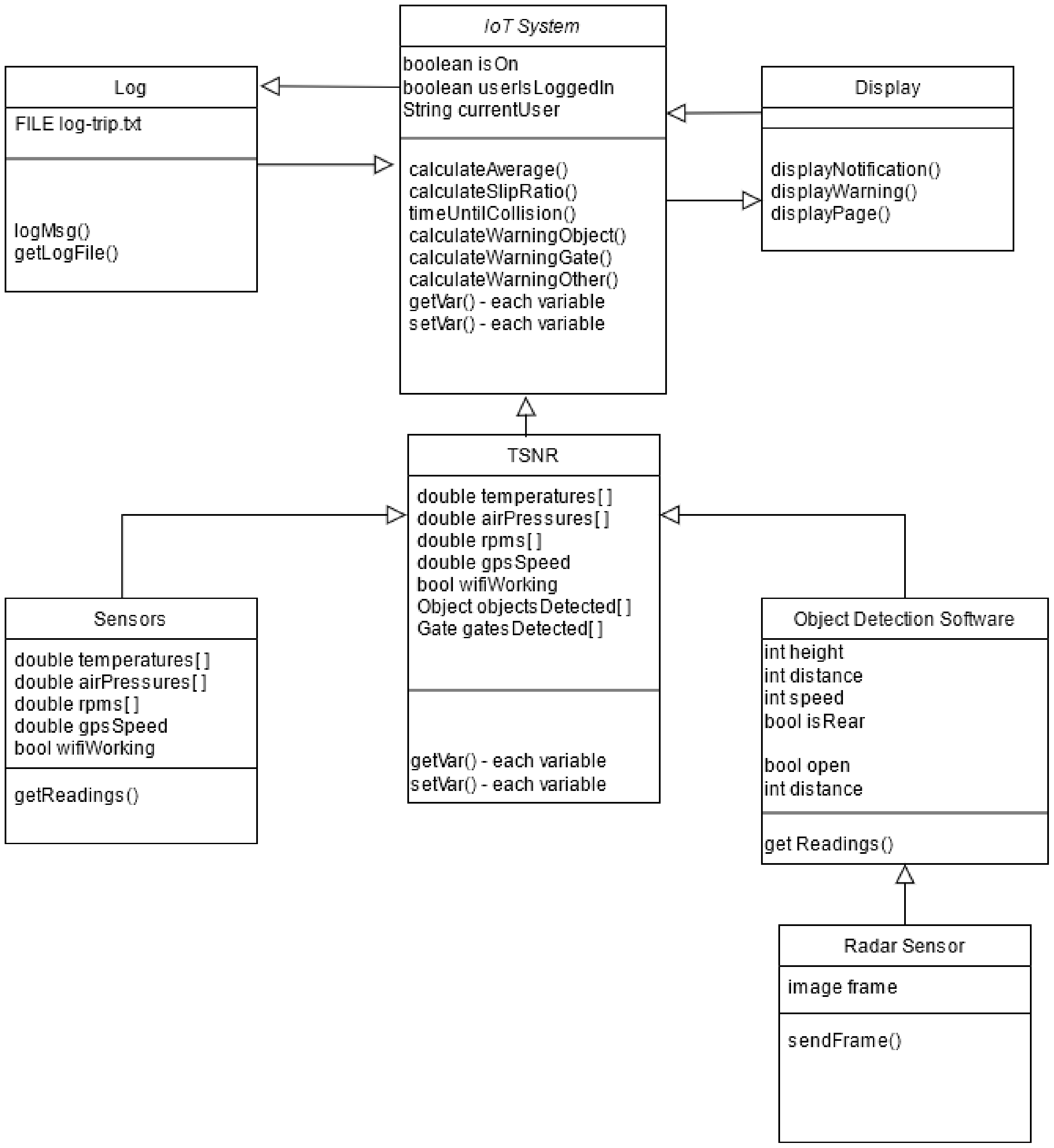
Scenario:

4. The screen displays the Administrator/Conductor view.
5. The conductor clicks the log-out button.
6. User is returned to the log-in page.

4.2: Use Case Diagram


Receive warning of network deconnection
Receive warning of upcoming open gate
Receive warning and data of wheel slippage
Receive warning and data of hazardous object on track
Receive warning and data of upcoming/current hazardous weather conditions
Access IoT system
Receive software log entries
Conductor
Administrator
Sensors

4.3: Class Based Diagram



4.4: CRC Modeling/Cards

IoT System
Description: Handle user login. Make calculations using sensor data.

Responsibility	**Collaborator**
calculateAverage() Calculates average of sensor data when needed	Sensors, Time Sensitive Network Router
calculateSlipRatio() Calculates the slip ratio using average RPM	Sensors, Time Sensitive Network Router
timeUntilCollision() Calculates time until collision for an between the train and an object	Sensors, Time Sensitive Network Router, Object Detection Software
calculateWarning() Decides what warning to send to display	Display, Sensors, Time Sensitive Network Router, Object Detection Software
bool isOn, userIsLoggedIn() Used to inform other classes that IoT System is ready to operate	Display, Log
String currentUser() Used by log and used to decide what permissions to give user	Display, Log

Display

Description: Used to display information to Conductor

Responsibility	**Collaborator**
displayNotification() Displays a visual notification to the Conductor	IoT System
displayWarning() Displays a visual warning to the conductor	IoT System
displayPage() Displays the correct page layout (i.e. administrator view, conductor view, conductor view during warnings, etc.) depending on various variable values	IoT System

Log

Description: Keeps track of IoT System actions

Responsibility	**Collaborator**
logMsg() Appends given string to log text file	IoT System
getLogFile() Downloads current log	IoT System
FILE log-trip.txt Text file of log	IoT System

Sensors

Description: Obtain readings on condition of/around train for IoT System

Responsibility	**Collaborator**
getReadings() Assigns sensor values to variables	IoT System
double temperatures[ ] Gets the temperature of the surrounding area	IoT System
double airPressures[ ] Gets air pressure of surrounding area (in Hg)	IoT System
double rpms[ ] Gets the rpm of the corresponding wheel	IoT System
double gpsSpeed Gets the gps speed from the GPS sensor	Iot System
bool wifiWorking Lets IoT System know that Wifi Is functional	IoT System

Object Detection Software (ODS)

Description: Detects objects/gates and notifies conductor

Responsibility	Collaborator
int height Stores the height of object on tracks	IoT System, Radar Sensor
int distance Stores the distance of object from train	IoT System, Radar Sensor
int speed Stores the speed of object from train	IoT System, Radar Sensor
bool isRear Stores the direction of the object from train	IoT System, Radar Sensor
bool open If gate is open, it is true, else false	IoT System, Radar Sensor
int distance Stores the distance of a gate	IoT System, Radar Senso

Radar Sensor

Description: Obtains images for Object Detection Software

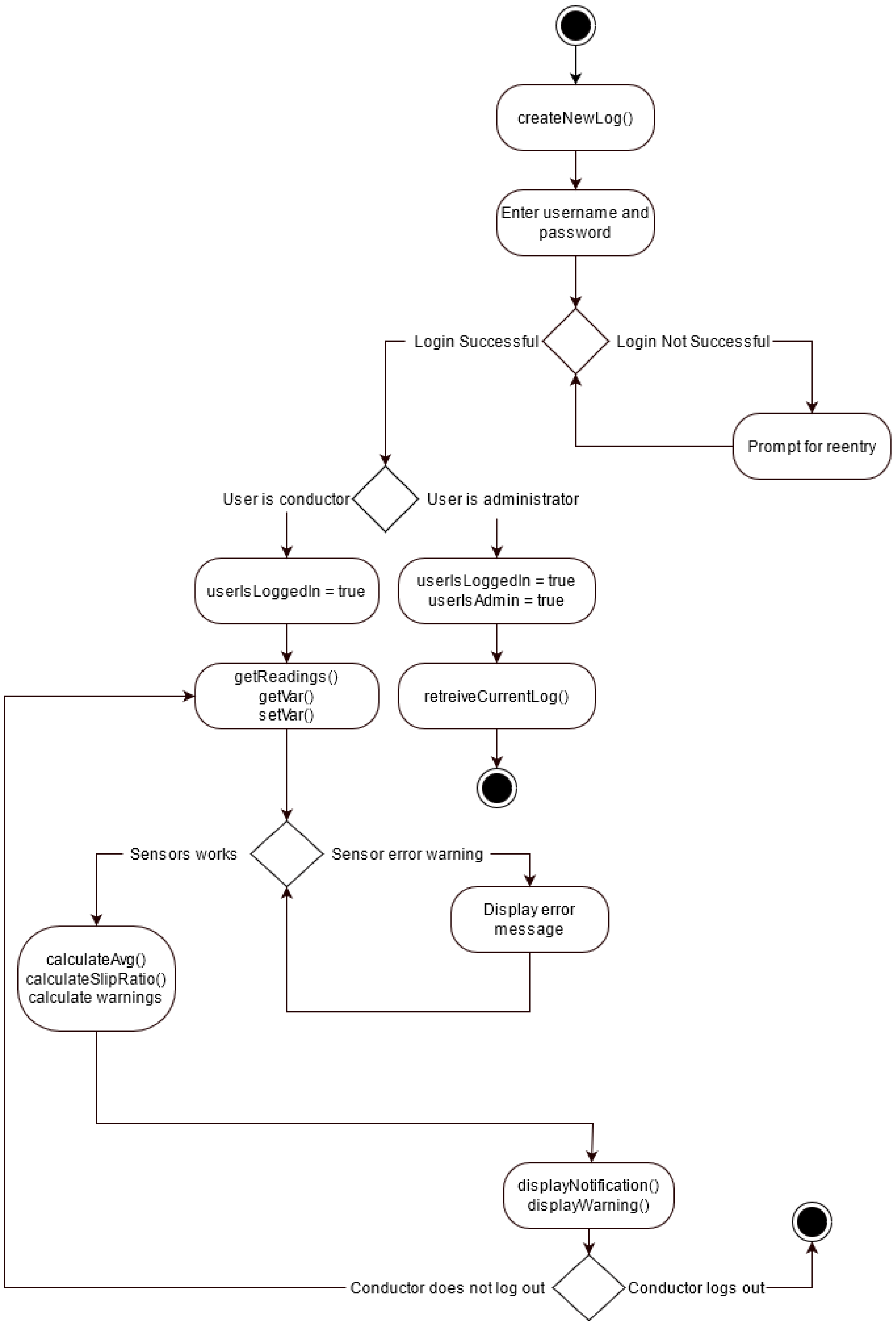
Responsibility	Collaborator
sendFrame() Sends a frame object to Object Detection Software	Object Detection Software, Time Sensitive Network Router
image frame A file that contains a frame obtained from the Radar Sensor	Object Detection Software

Time Sensitive Network Router (TSNR)

Description: Stores all raw sensor data and data calculated by the ODS

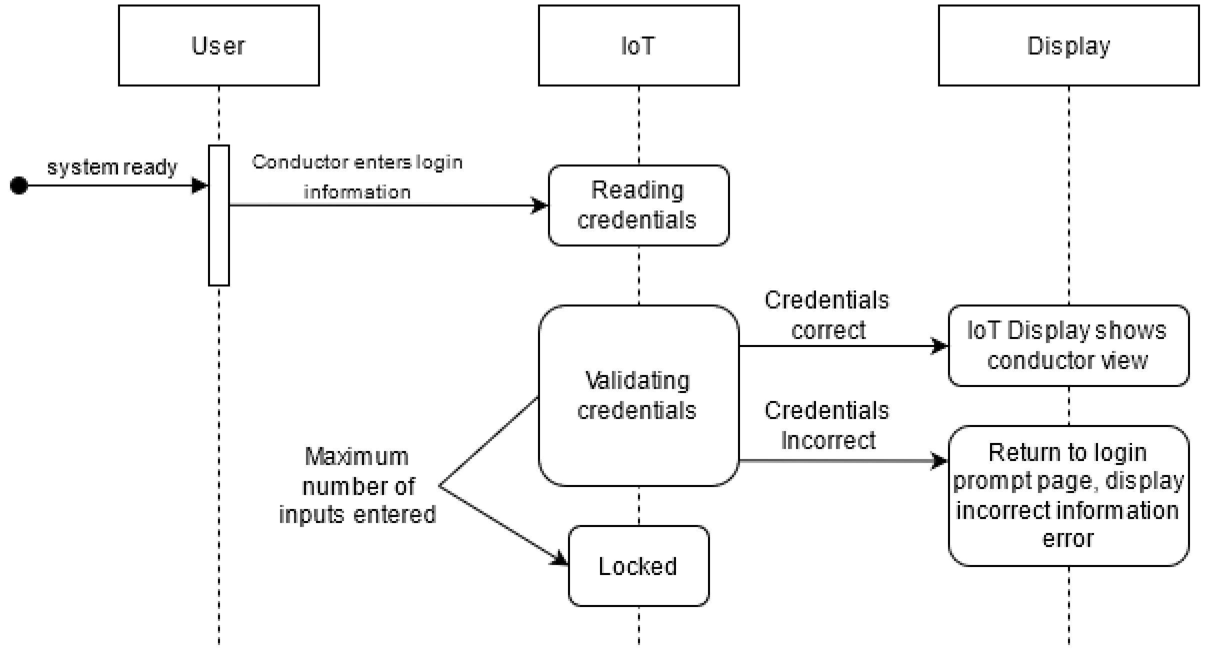
Responsibility	**Collaborator**
double temperatures[ ], double airPressures[ ], double rpms[ ], double gpsSpeed Raw sensor data	Sensors
bool wifiWorking Information on state of wifi connectivity	Sensors
Object objectsDetected[ ], Gate gates Detected[ ] Object Detection Related Information	Object Detection Software, Radar Sensor
getVar(), setVar() Get the variables readings from the sensors and set them to corresponding variable values	Sensors

4.5: Activity Diagram

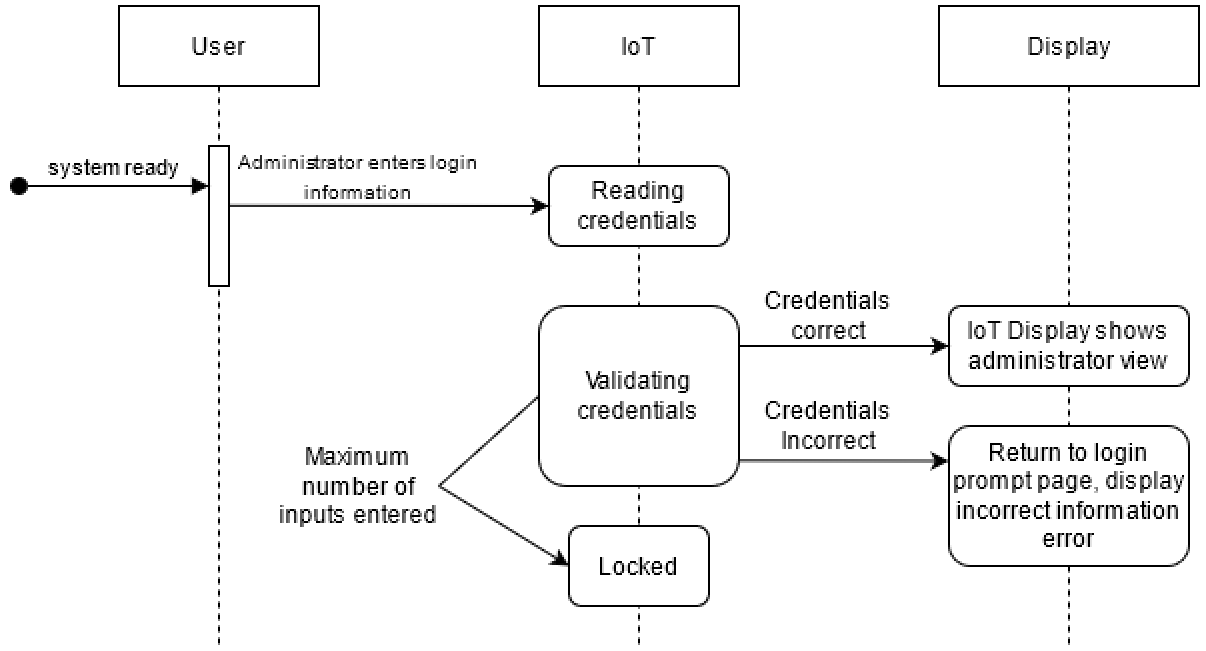


4.6: Sequence Diagrams

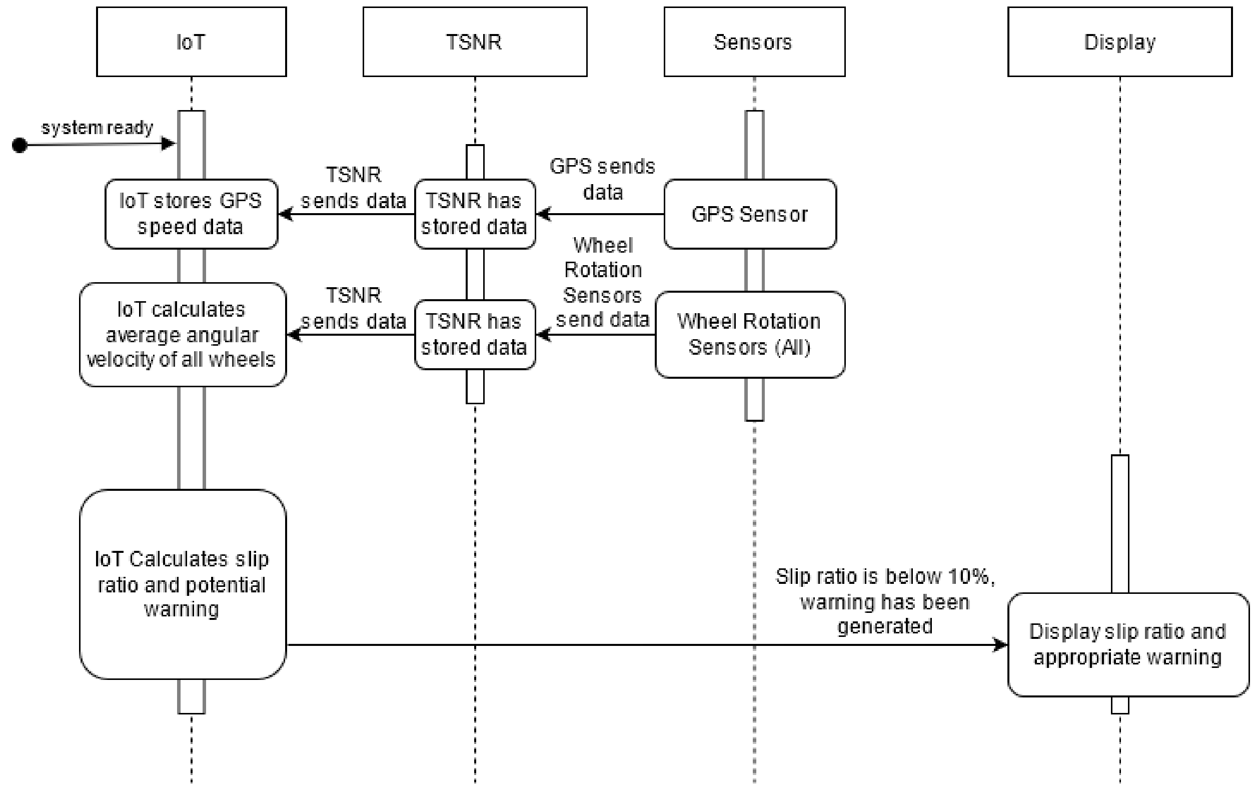
4.6.1: Conductor Log-In Sequence Diagram


User
IoT
Display
system ready
Conductor enters login information
Reading credentials
Credentials correct
IoT Display shows conductor view
Validating credentials
Credentials Incorrect
Return to login prompt page, display incorrect information error
Maximum number of inputs entered
Locked

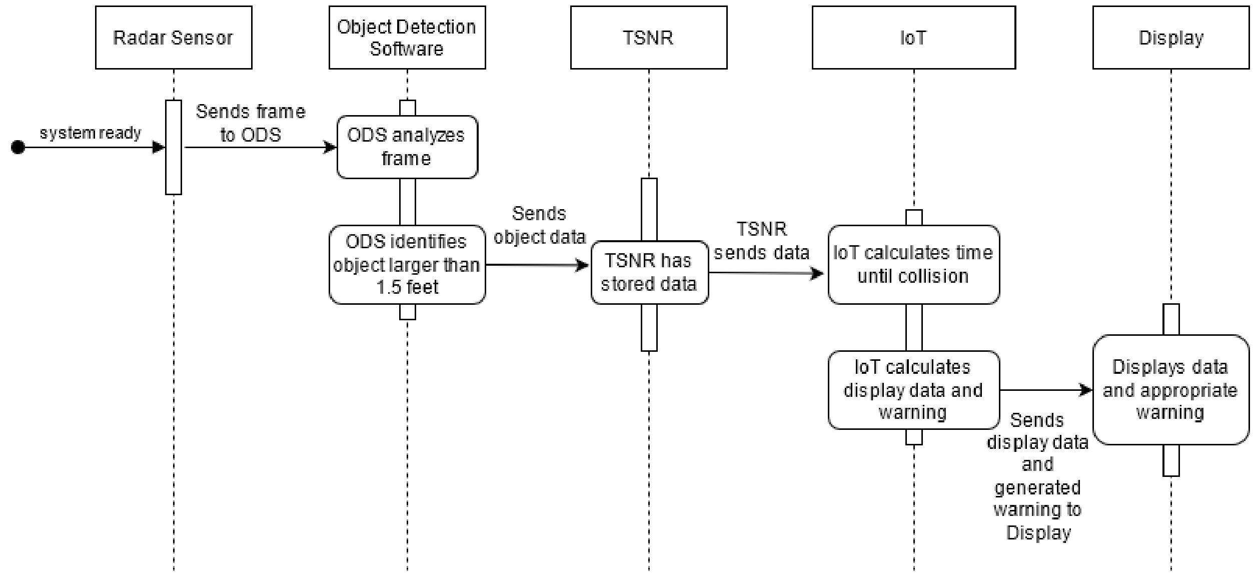
4.6.2: Administrator Log-In Sequence Diagram


User
IoT
Display
system ready
Administrator enters login information
Reading credentials
Credentials correct
IoT Display shows administrator view
Validating credentials
Credentials Incorrect
Return to login prompt page, display incorrect information error
Maximum number of inputs entered
Locked

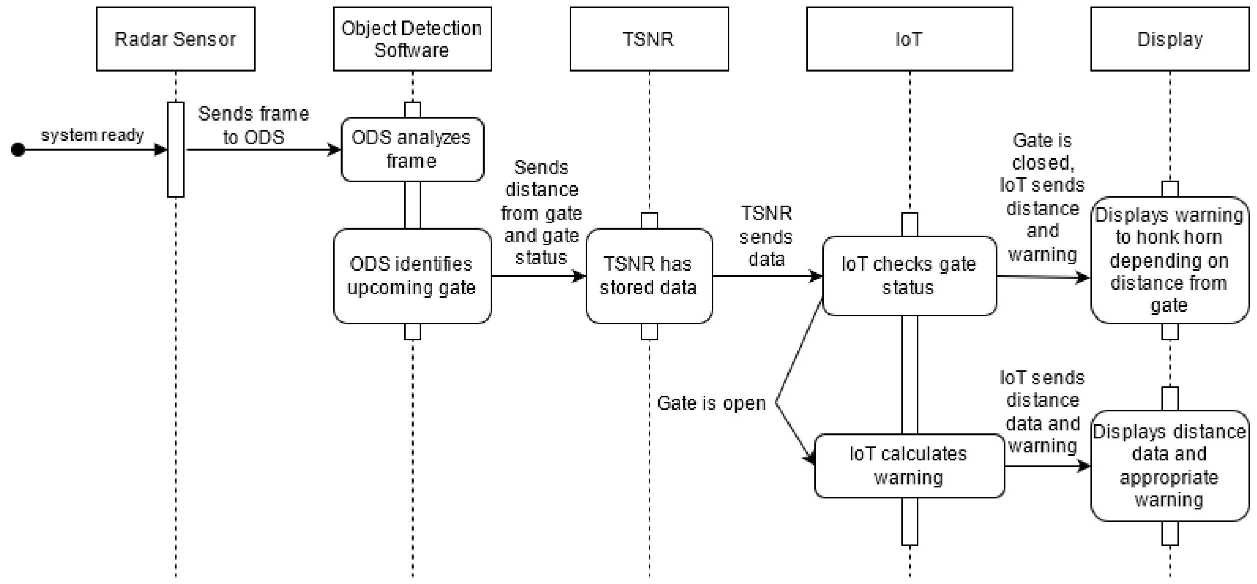
4.6.3: Wheel Slippage Sequence Diagram


IoT
TSNR
Sensors
Display
system ready
IoT stores GPS speed data
TSNR sends data
TSNR has stored data
GPS sends data
GPS Sensor
Wheel Rotation Sensors send data
IoT calculates average angular velocity of all wheels
TSNR sends data
TSNR has stored data
Wheel Rotation Sensors (All)
IoT Calculates slip ratio and potential warning
Slip ratio is below 10%, warning has been generated
Display slip ratio and appropriate warning

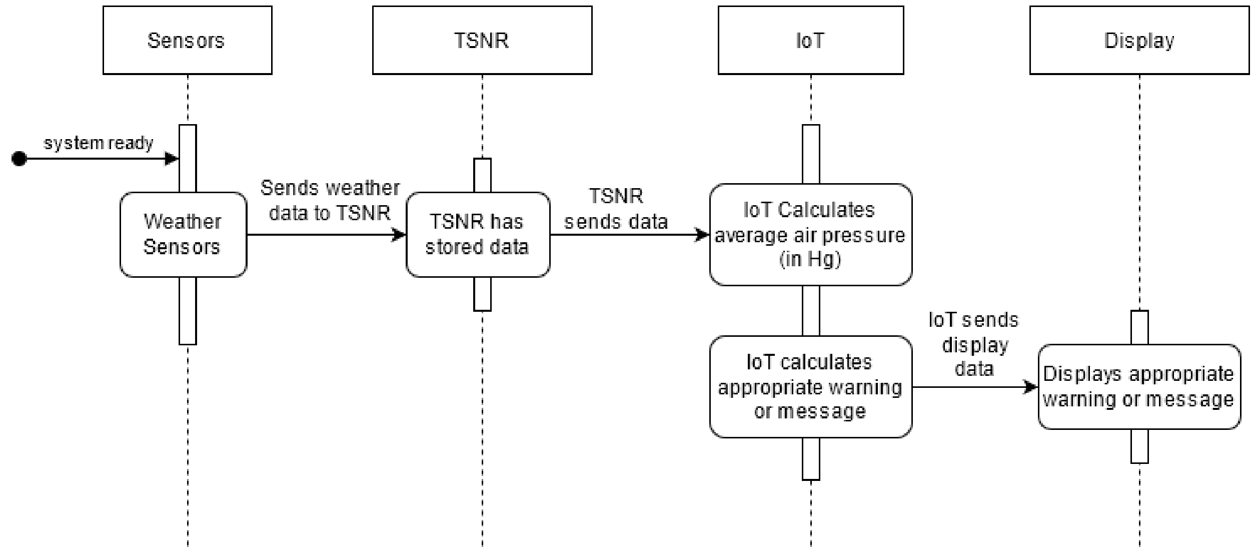
4.6.4: Object Detection Sequence Diagram


Radar Sensor
Object Detection Software
TSNR
IoT
Display
system ready
Sends frame to ODS
ODS analyzes frame
ODS identifies object larger than 1.5 feet
Sends object data
TSNR has stored data
TSNR sends data
IoT calculates time until collision
IoT calculates display data and warning
Sends display data and generated warning to Display
Displays data and appropriate warning

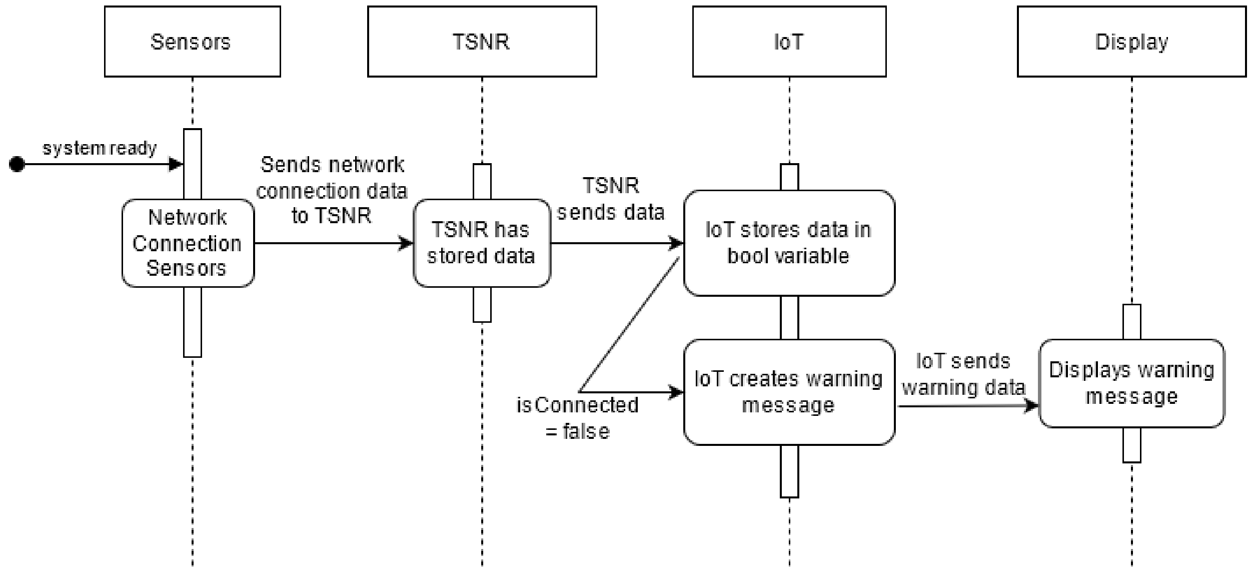
4.6.5: Gate Detection Sequence Diagram


Radar Sensor
Object Detection Software
TSNR
IoT
Display
system ready
Sends frame to ODS
ODS analyzes frame
ODS identifies upcoming gate
Sends distance from gate and gate status
TSNR has stored data
TSNR sends data
IoT checks gate status
Gate is closed, IoT sends distance and warning
Displays warning to honk horn depending on distance from gate
Gate is open
IoT calculates warning
IoT sends distance data and warning
Displays distance data and appropriate warning

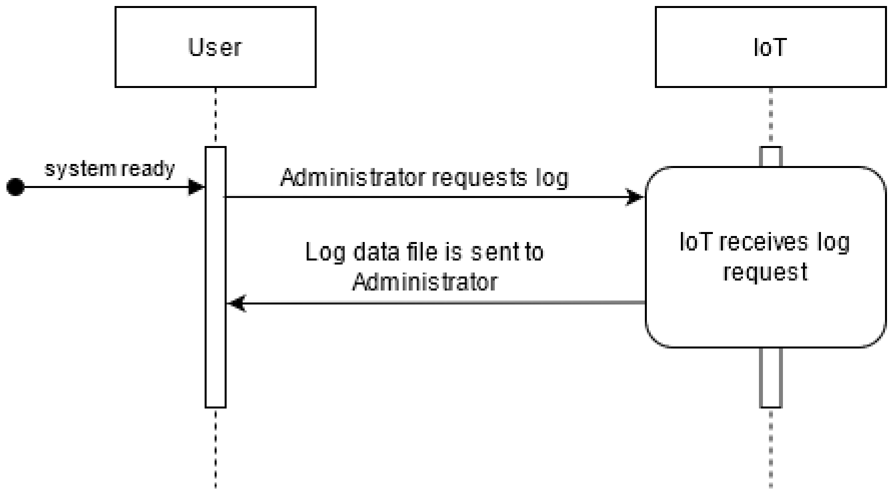
4.6.6: Weather Sequence Diagram


Sensors
TSNR
IoT
Display
system ready
Weather Sensors
Sends weather data to TSNR
TSNR has stored data
TSNR sends data
IoT Calculates average air pressure (in Hg)
IoT calculates appropriate warning or message
IoT sends display data
Displays appropriate warning or message

4.6.7: Network Connection Sequence Diagram

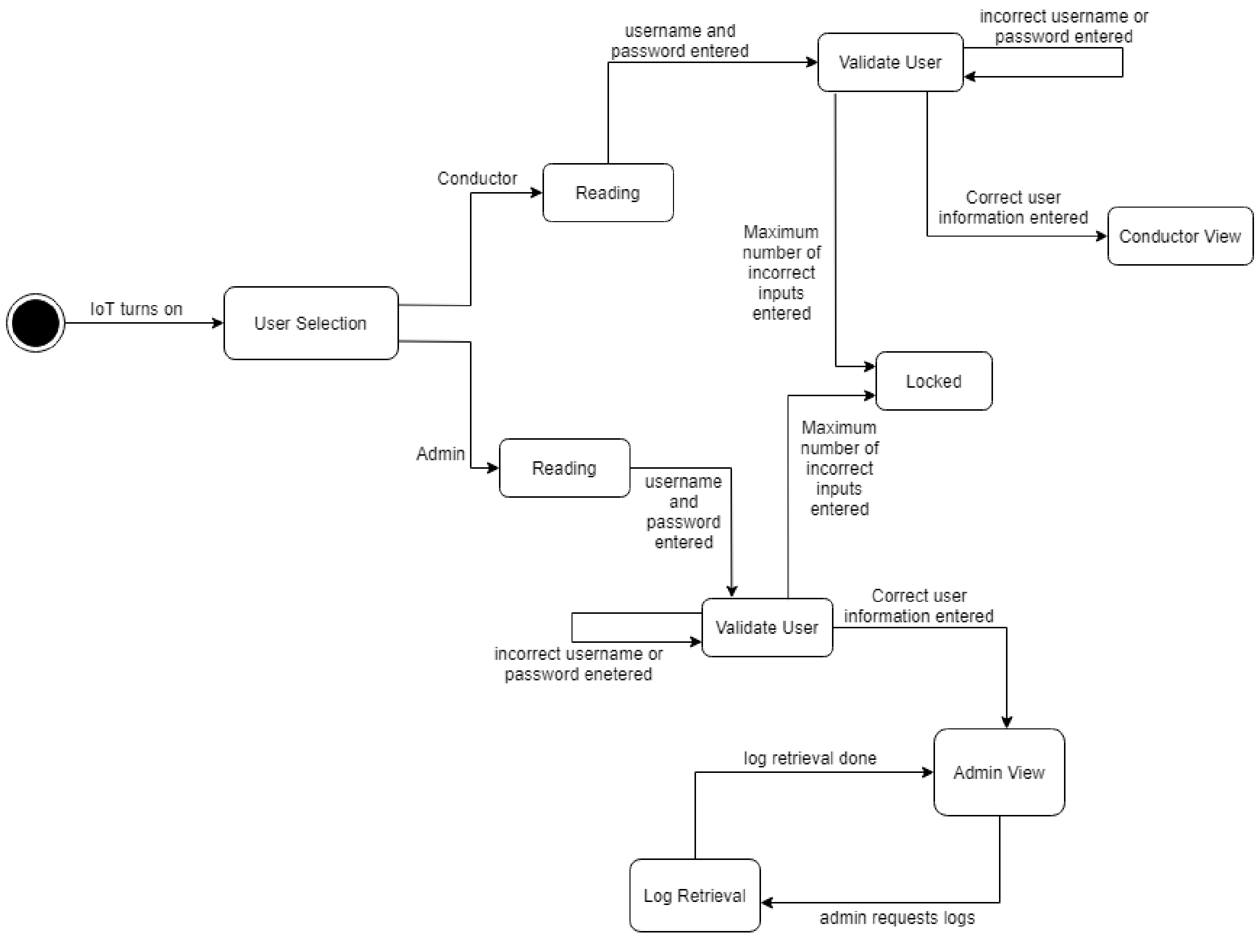


4.6.8: Log Request Sequence Diagram

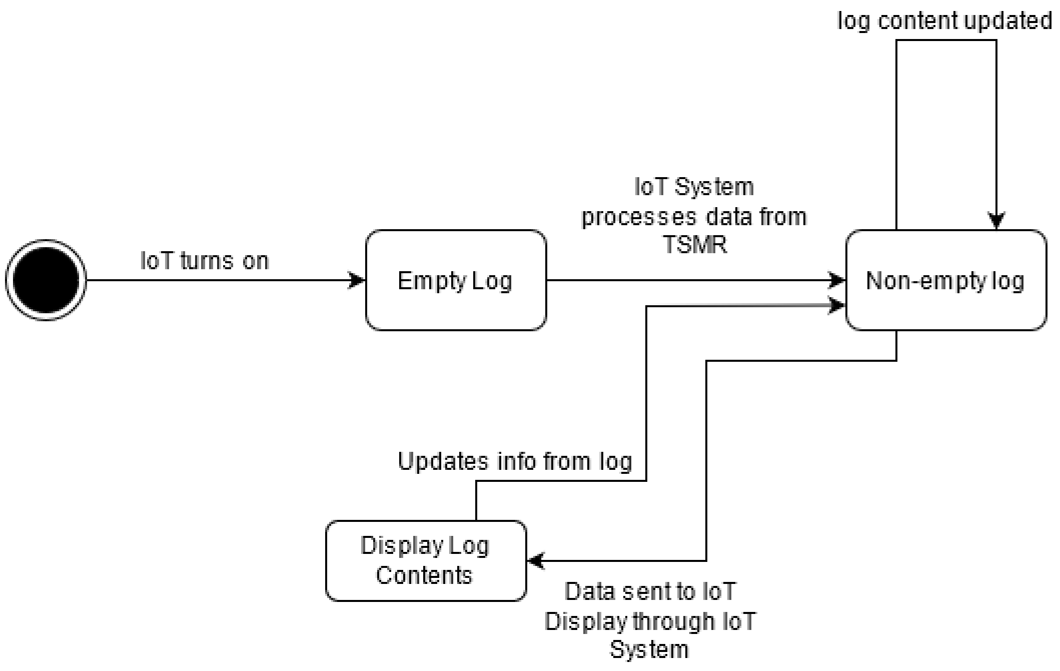

User
IoT
system ready
Administrator requests log
Log data file is sent to Administrator
IoT receives log request

4.7: State Diagrams

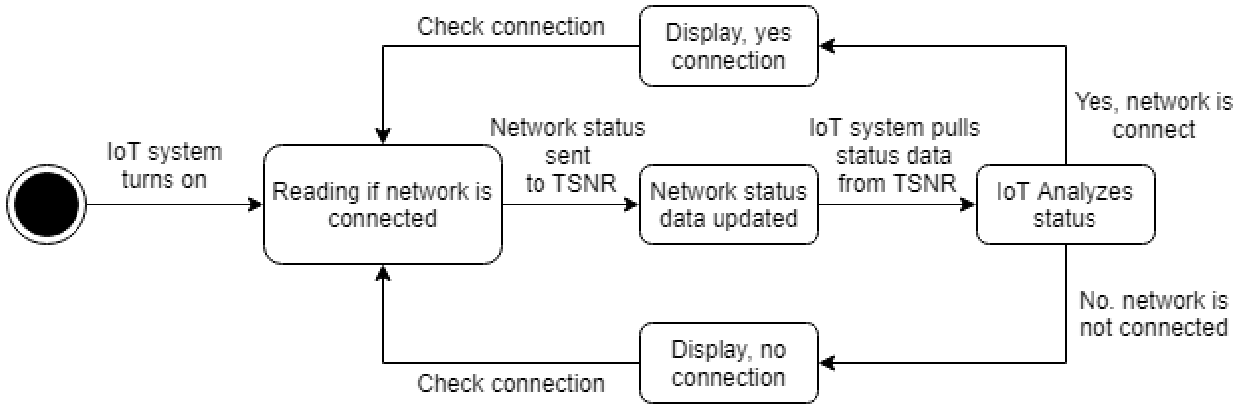
4.7.1: User Log-In State Diagram


username and password entered
Validate User
incorrect username or password entered
Conductor
Reading
Correct user information entered
Conductor View
Maximum number of incorrect inputs entered
IoT turns on
User Selection
Locked
Maximum number of incorrect inputs entered
Admin
Reading
username and password entered
Validate User
Correct user information entered
incorrect username or password enetered
Admin View
log retrieval done
Log Retrieval
admin requests logs

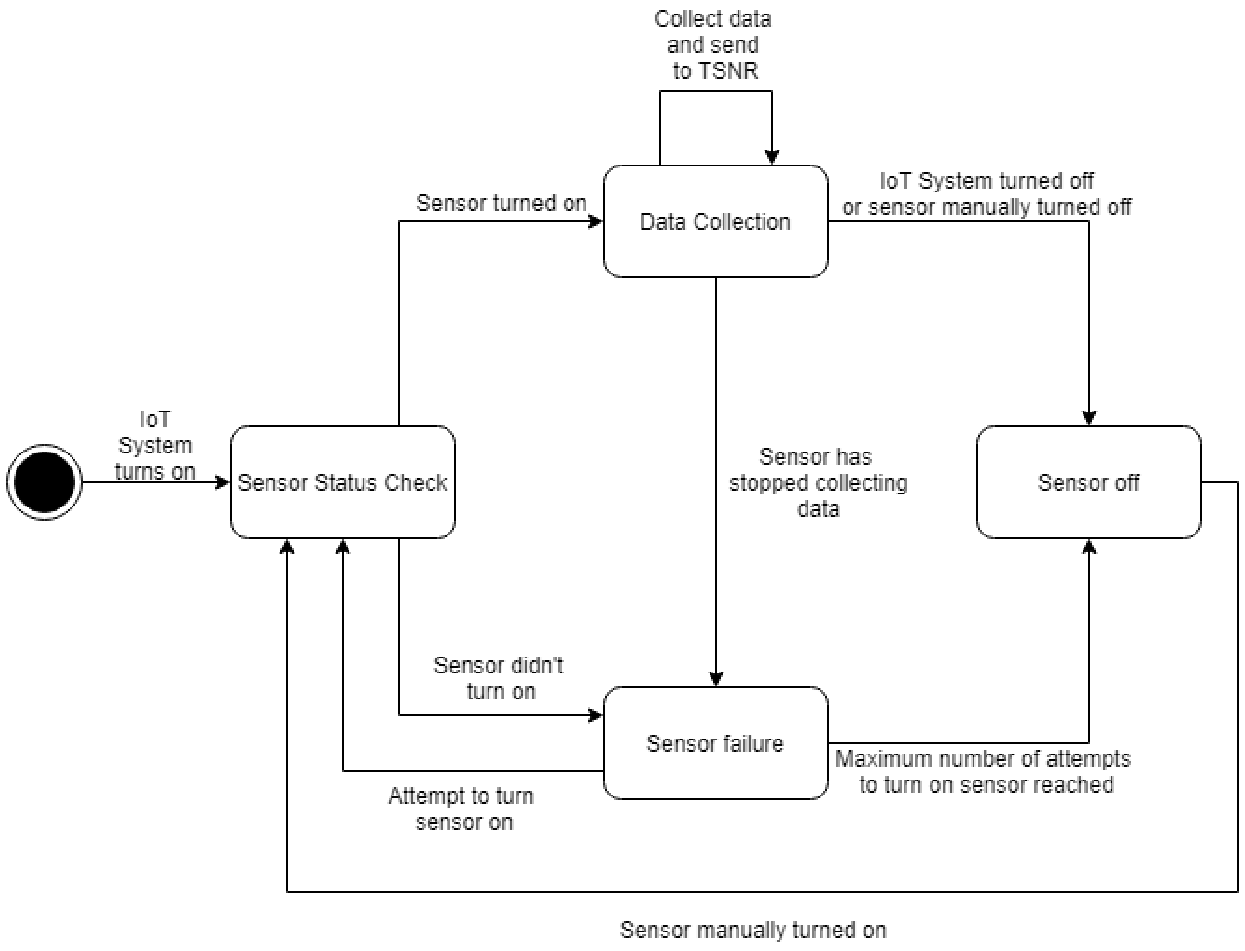
4.7.2: Log State Diagram



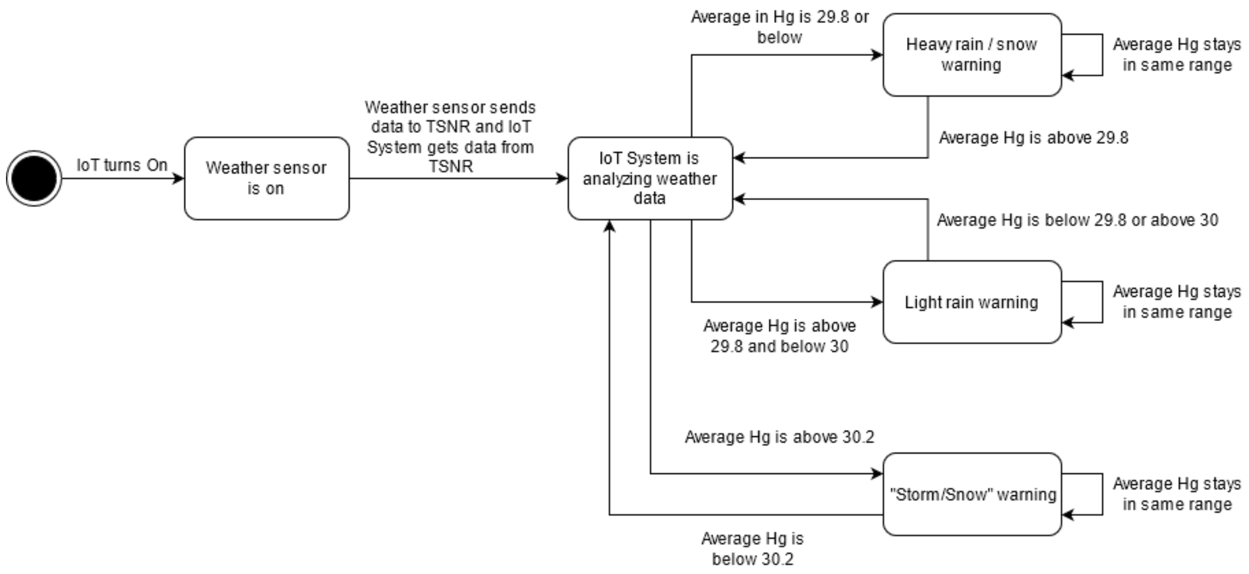
4.7.3: Network Connection State Diagram


Check connection
Display, yes connection
Yes, network is connect
IoT system turns on
Reading if network is connected
Network status sent to TSNR
Network status data updated
IoT system pulls status data from TSNR
IoT Analyzes status
No. network is not connected
Display, no connection
Check connection

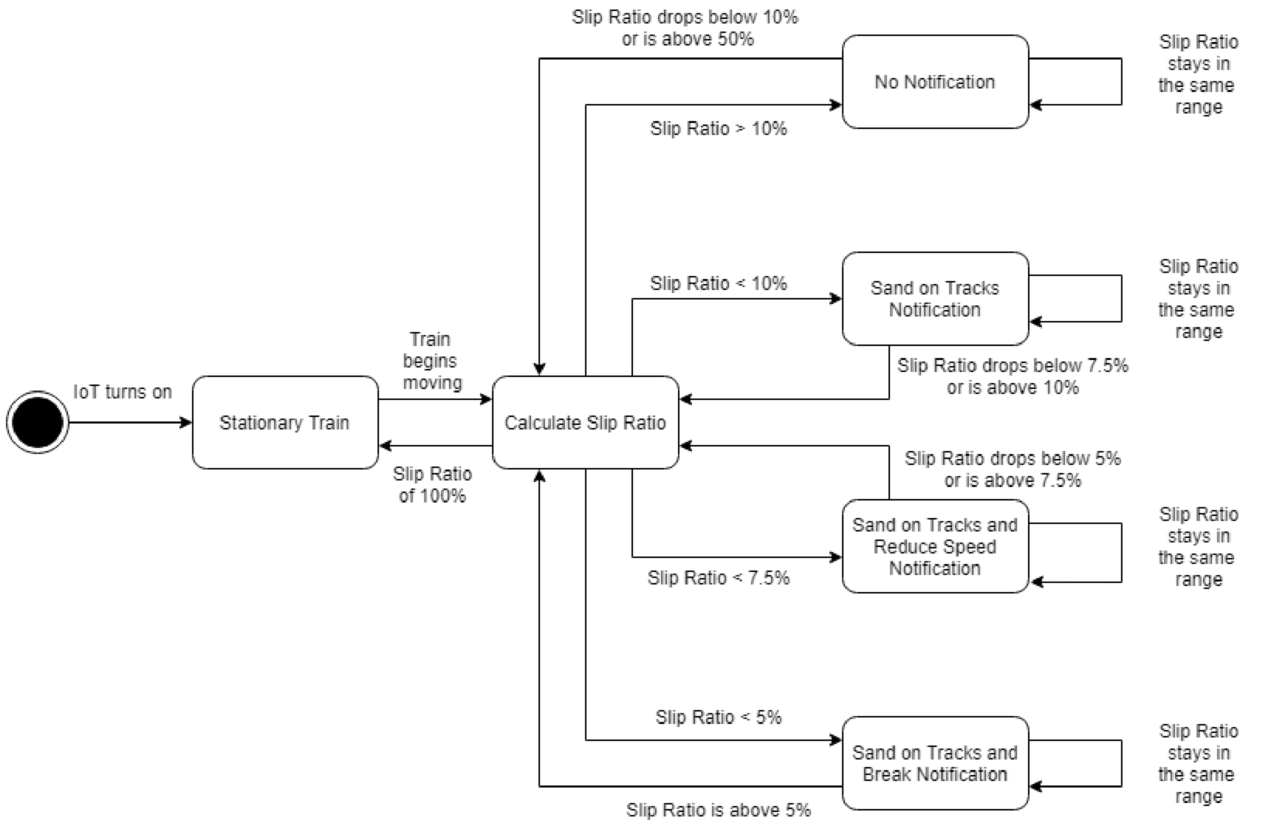
4.7.4: Sensor State Diagram


Collect data and send to TSNR
Sensor turned on
Data Collection
IoT System turned off or sensor manually turned off
IoT System turns on
Sensor Status Check
Sensor has stopped collecting data
Sensor off
Sensor didn't turn on
Sensor failure
Maximum number of attempts to turn on sensor reached
Attempt to turn sensor on
Sensor manually turned on

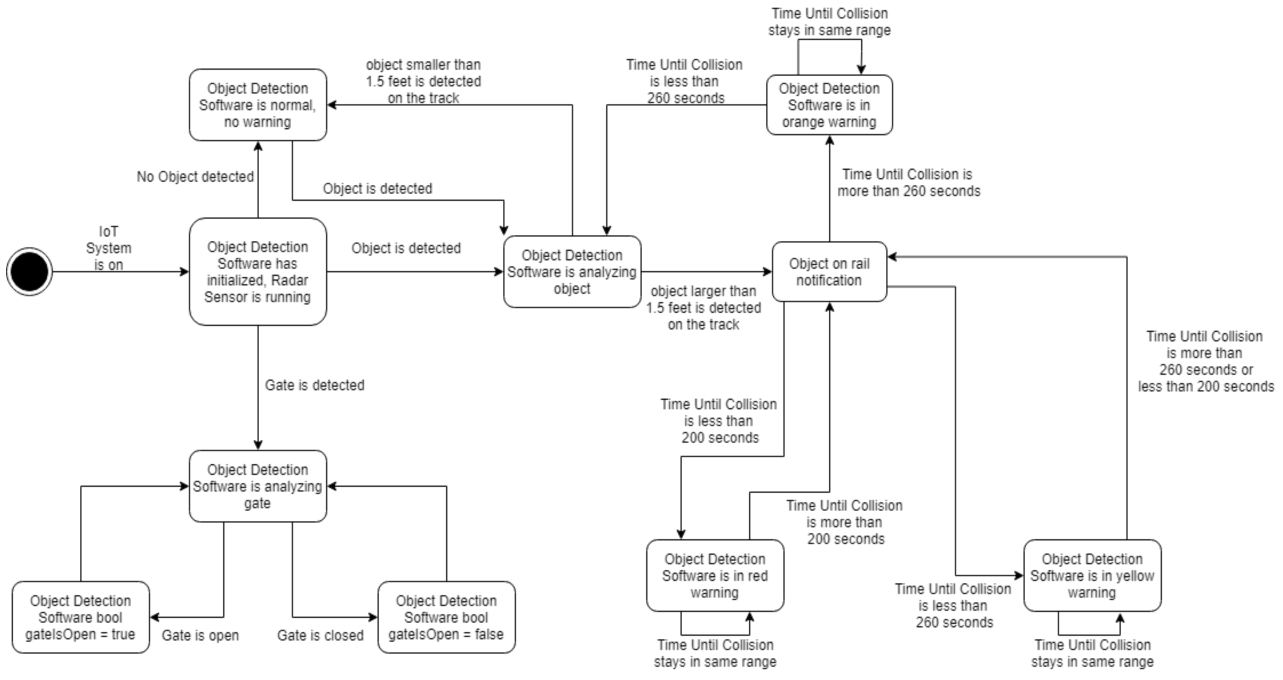
4.7.5: Weather State Diagram


Average in Hg is 29.8 or below
Heavy rain / snow warning
Average Hg stays in same range
Weather sensor sends data to TSNR and IoT System gets data from TSNR
Average Hg is above 29.8
IoT turns On
Weather sensor is on
IoT System is analyzing weather data
Average Hg is below 29.8 or above 30
Light rain warning
Average Hg stays in same range
Average Hg is above 29.8 and below 30
Average Hg is above 30.2
"Storm/Snow" warning
Average Hg stays in same range
Average Hg is below 30.2

4.7.6: Wheel Slippage State Diagram


IoT turns on
Stationary Train
Train begins moving
Slip Ratio of 100%
Calculate Slip Ratio
Slip Ratio drops below 10% or is above 50%
Slip Ratio > 10%
No Notification
Slip Ratio stays in the same range
Slip Ratio < 10%
Sand on Tracks Notification
Slip Ratio drops below 7.5% or is above 10%
Slip Ratio drops below 5% or is above 7.5%
Sand on Tracks and Reduce Speed Notification
Slip Ratio < 7.5%
Slip Ratio < 5%
Sand on Tracks and Break Notification
Slip Ratio is above 5%

4.7.7: Object Detection Software State Diagram


IoT System is on
Object Detection Software has initialized, Radar Sensor is running
No Object detected
Object Detection Software is normal, no warning
object smaller than 1.5 feet is detected on the track
Object is detected
Object is detected
Object Detection Software is analyzing object
object larger than 1.5 feet is detected on the track
Object on rail notification
Time Until Collision is less than 260 seconds
Object Detection Software is in orange warning
Time Until Collision stays in same range
Time Until Collision is more than 260 seconds
Time Until Collision is less than 200 seconds
Object Detection Software is in red warning
Time Until Collision stays in same range
Time Until Collision is more than 200 seconds
Time Until Collision is less than 260 seconds
Object Detection Software is in yellow warning
Time Until Collision stays in same range
Time Until Collision is more than 260 seconds or less than 200 seconds
Gate is detected
Object Detection Software is analyzing gate
Gate is open
Gate is closed
Object Detection Software bool gateIsOpen = true
Object Detection Software bool gateIsOpen = false

Section 5: Software Architecture

5.1: IoT Implementations and Pros and Cons

5.1.1: Data Centered Architecture

Using a Data Centered Architecture would be relatively simple because it would able to be constructed similar to our class based diagram. The data store would be the TSNR, where the different filters and client softwares being the differing classes that connect to and interact with the TSNR would act as the pipes and filters. This follows the current diagrams for the IoT software and would be straightforward to implement, but deviates slightly. Instead of the system revolving around the analysis and data processing within the IoT System class it would be revolving around the uploading and pulling of data from the TSNR, and less of the computations which is the most important factor for this entire system. Along with this an over-reliance on the TSNR can lead to catastrophic system failure if the TSNR is compromised in any way, which will cause the system to be unable to send data between filters (functions to collect and analyze data) and display the information to the display.

Pros	**Cons**
● Adding new feature is just adding new client (easy to improve system or add more sensors in the future) ● (Integrability) Adding new client/component does not affect other clients ● All clients take from the same database which avoids inconsistent information between clients ● Data can be passed between clients/components using "blackboard" mechanism	● Clients/components cannot DIRECTLY communicate with each other which decreases speed of our system ● Too much reliability on one data storage ● Need really good security to protect from malicious attacks ● Accidental data deletion can cripple entire system

5.1.2: Data Flow Architecture

Using the Data Flow Architecture to implement IoT HTR would be relatively easy and most efficient, as our IoT HTR parses and processes data in a filter-like fashion without requiring us to know how it was gathered. We pass the data from the sensors through "pipes" to the TSNR, which acts as a filter. The TSNR would then pass data through a pipe to the IoT System which in itself would act as a filter for the data, sending the appropriate data to the log and sending the appropriate data/messages to the IoT Display. This is a great way to separate functions and further simply the actual processes within our code. Our data just needs to be transformed through a series of computational or manipulative components into output data, which is exactly what the Data-Flow Architecture calls for. Also, a filter doesn't require knowledge of the workings of its neighboring filters, which is applicable to our IoT HTR implementation.

Pros	Cons
● good way to separate complicated processes into step by step actions (parsing sensor data) ● good because the way we parse data does not require us to know how it was gathered	● Only goes one way, which means you cannot go back (idk how this affects us yet) ● Does not involve classes, which may make things complicated

5.1.3: Call Return Architecture

Using the Call Return Architecture to implement IoT HTR would be relatively challenging, as our IoT HTR, despite being relatively modular in nature, doesn't just have one central "main program" that can call upon other "subprograms". Another issue to confront would be that most of our system actions between the sensors and TSNR classes and between the TSNR and IoT Display classes start from the lower level components rather than the higher level ones. To work with a Call Return Architecture, we would have to reformat our IoT HTR to start with the main program as the IoT display, with the IoT Display calling for data from the subprogram IoT System, which calls a subprogram for both Log and TSNR. TSNR would then call for the subprograms of sensors and ODS, which would just be data transfer between the two classes. The remote procedure call aspect of this architecture would be redundant, however, notably because we would do everything locally on IoT HTR instead of calling for subprograms on remote computers. This feature would be helpful if we had many computations to do and if we wanted our program to be massively scalable, but that is not the case for IoT HTR.

Pros	Cons
● Simplifies actions that occur in our system ● Well organized ● Easy to modify or scale existing code	● There is no real Main program in our IoT HTR. The whole system is the main program ● Most of our system actions start from the lower level components (the sensors) rather than the higher level components ● More components makes system very complicated ● More components makes testing and debugging much more challenging

5.1.4: Object-Oriented Architecture

Using the Object-Oriented Architecture to implement IoT HTR would be easiest from a programming/code point of view. Our diagrams and class structure shown in Section 4 can be easily converted to a working program using an object-oriented programming language, as our class diagrams, crc cards, and state diagrams highlight, in great detail, what each component (piece of data) means/is and what each operation (the functions in our class diagrams/crc cards) would do. Our operations, when applied, manipulate or send/store data depending on their respective task. Our overall code would be cluttered (IoT HTR has many classes with their own respective components and operations) but still very manageable, and our previous work is tailored specifically for ease of implementation to this architecture due to the nature of our already created class systems and how they nearly identically correlate to the Object-Oriented Architecture.

Pros	**Cons**
● Our system is already subdivided into classes, could easily be converted into "objects" ● Coordination between components is accomplished via message passing, works well with sending data or warning messages ● Easier for testing and debugging ● Class distinction makes code easier to follow	● Our system does not have a lot of distinct actors ● Smaller components (i.e. Radar sensor) have limited functionality, creating an object class for small components would be redundant ● Can get complex when lots of classes are needed

5.1.5: Layered Architecture

Using the Layered Architecture to implement IoT HTR would be relatively efficient, as our system already has a class system that can be arranged into the 4 layers of the Layered Architecture. The "core layer" would contain each of our sensors and the ODS as components. The "core layer" would only communicate with the "utility layer", which in our IoT Systems case would be the TSNR, the class where each of the sensor's information would be stored. This "utility layer" would then communicate that information to the "application layer", which would be our IoT Systems class, where all of the calculations regarding the sensor data as well as possible warnings are created. The "application layer" would then communicate this information to the "user interface layer", which is our IoT Display. Each "component" of the "user interface layer" would be the different sections of our display for each piece of information we present to the conductor.

Pros	Cons
● Our system has a good amount of layers. ● Our layers can be: Sensors, Sensor data processing, warning calculation, and display ● Brings organization to our system	● confusing and complicated ● Each layer can increase workload for basic operations ● More layers can introduce more points of failure if any operation requires communication between layers

5.1.6: Finite State Machine Architecture

Using the Finite State Machine Architecture to implement IoT HTR would be challenging, as this architecture doesn't allow our system to be in multiple states at the same time. However, one workaround to implement IoT HTR would be to have each class in our system be their own component with their own respective states. We have the state diagrams already available to us, which would allow for easy translation into this architecture. Each of our state diagrams depict what state our components could be in, and which process/action would change the component's state. The main issue with the Finite State Machine Architecture would be our implementation of our display class, as it needs to display multiple pieces of information separately for one another (IoT Display has individually changing sections such as weather status section, emergency information section, wifi connection section, etc.). We cannot have the entirety of the display class as one state because some information on the display could change while others remain the same, creating a need for an unnecessary amount of states with the number of states exponentially increasing for each piece of information we want to display. We could work around this by having each of the sections on the IoT Display be their own respective sections/classes. For example, our wifi connection section of our display would be its own class and have two states: wifi on, wifi off. Our emergency information section would have a specific state for every possible combination of emergency messages, and switch to the appropriate state when asked to by the IoT. This would drastically reduce the complexity of our IoT implementation using this architecture and make programming IoT HTR more manageable.

Pros	Cons
● The states of our system could be the warnings given ● State diagrams are already made which makes this architecture easier to approach	● This architecture does not allow our system to be in multiple states at the same time ● Need to account for all necessary states and variations within total system, making our code for IoT very complex

5.2: Our Choice (Data Flow Architecture)

5.2.1: Architecture Choice

We believe that the Data Flow Architecture works best for our system because it fits the process in which our data is handled. The Data Flow Architecture focuses on passing data through pipes and filters. The data that we are passing through these filters is all data collected by the sensors. Each filter is a step in the parsing process of our data. For example, when an object is detected by the Radar, the Radar data will go through the Object Detection Software "filter" to have height and distance calculated. This data will then go to the IoT System "filter" so that the appropriate warning can be chosen. And finally, the warning data will go to the IoT display "filter" to decide how and where the data will be displayed. The largest general drawback to using this Architecture is that data cannot go backwards from the IoT System to the sensors. However this does not impact our system because we will never need data to go backwards. The processing of our data is very streamlined and unidirectional. All architectures had their pros and cons, but the Data Flow Architecture fits our System too perfectly for us to choose another Architecture.

5.2.2: Component: functions, actions, and filters

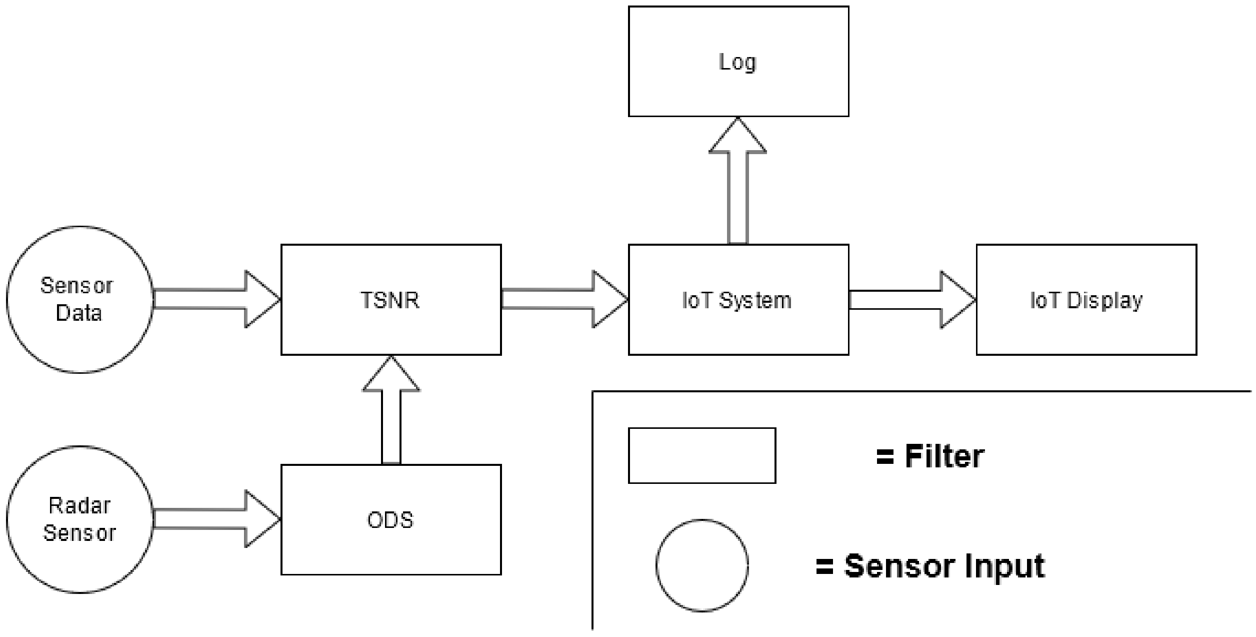
- Sensors
    - Weather sensor
        - Collects air pressure data
            - Call function getReadings()
            - Store in variable "double airPressure"
        - Collects temperature data
            - Call function getReadings()
            - Store in variable "int temperatures[ ]"
        - Sends data to TSNR (through pipe)
    - Radar Sensor
        - Used by ODS to sense object on the track
            - Uses sendFrame() to send object to ODS (through pipe)
        - Used by ODS to detect if gate is open or closed
            - Uses sendFrame() to send gate data to ODS (through pipe)
    - Object Detection Software
        - Calculates and stores information of each detected object
            - Store height of object int "int height"
            - Store the distance between the locomotive and the object in "int distance"
            - Store the speed of the object in "int speed"
            - Store the position of the object relative to the train in "isRear"
            - Calculates using velocity/speed and stores the seconds until collision in "timeUntilCollision"
            - Calculates if detected gate is open and stores it in "bool open"

            - Stores distance of gate in “int distance”
        - Sends data to TSNR (through pipe)
    - Wheel Sensor
        - Calculates the RPM of a wheel using getRPM()
        - Sends data using “getReadings()” to TSNR (through pipe)
    - Wifi Sensor
        - Uses “wifiWorking()” to tell the user if the wifi is working or not
        - Sends data to TSNR (through pipe)
- Time Sensitive Network Router
    - Stores the temperature int “int temperatures[ ]”
    - Stores the air pressure “int airPreasures [ ]”
    - Stores RPM in “double rpms[ ]”
    - Stores if wifi is on or off in “bool wifiIsWorking”
    - Store height of object int “int height”
    - Store the distance between the locomotive and the object in “int distance”
    - Store the speed of the object in “int speed”
    - Store the position of the object relative to the train in “isRear”
    - Calculates using velocity/speed and stores the seconds until collision in “timeUntilCollision”
    - Calculates if detected gate is open and stores it in “bool open”
    - Stores distance of gate in “int distance”
    - Sends data to IoT (through pipe)
- Log
    - Stores whole log text in “log-trip.txt”
    - IoT System can call “getLogFile()” to retrieve the current log data
    - IoT System can call “logMsg()” to append a String to the log file text
    - Retrieves data from IoT System (through pipe)
- IoT Display
    - Uses “displayWarning()” and “displayNotification()” to display warning and notifications on the IoTDisplay
    - Uses “displayPage()” to correctly display either the “Administrator View” if an administrator is logged in, or the various “Operator View”’s corresponding to the warning status’ received
    - Retrieves data from IoT (through pipe)
- IoT System
    - Uses “bool isOn” to tell the rest of the system that the IoT System is on
    - Uses “String currentUser” to tell the IoT system and the log who the current user is
    - Uses “bool userIsLoggedIn” to determine if the system should display the log-in screen or the standard screen
    - Uses “calculatesAverage()” to calculate the average of any sensor data for example the average RPM or average Air pressure
    - Uses “calculateSlipRatio()” to calculate the slip ratio using SAE J670 slip ratio equation

- ○ Uses “calculateWarning()” to calculate what the warning that should be displayed is using the sensor data (see requirements section)
- ○ Retrieves data from TSNR (through pipe)
- ○ Sends data to Log and Display (through pipe)

Filters with this architecture would be the ODS (filters radar sensor data), TSNR (filters sensor data), IoT System (filters all data for the Log and IoT Display), and IoT Display (filters given data and puts into appropriate sections on Display). All data transfer between sensors and filters go through pipes.

5.2.3: Data Flow Architecture Diagram



5.2.4: IoT HTR Program Blueprint Data Flow

IoT Initialization, User Log-in and Log-out

- <u>These system actions occur separately from the other functions of the IoT HTR System.</u> They do not occur within the continuous looping of our program. The IoT HTR System, when a user is logged in, is constantly updating and performing its intended actions, whereas when a user is logged out, the IoT HTR System only prompts for log-in credentials.

IoT System turns on (when train connects to power):

- bool isOn = true
- bool userIsLoggedIn = false
- createNewLog(): creates a new log
  - String logEntry = "New log created."
  - updateLog(): appends whatever is in String logEntry into the log
- displayPage(): displays correct page depending on value of various boolean variables
  - If "bool userIsLoggedIn = false", displayPage() will always only display the login page until user has provided correct credentials

User provides correct Credentials:

- String currentUser = "*entered username*"
  - If current user is an administrator,
    - bool userIsLoggedIn = true
    - String logEntry = "Administrator *String currentUser* logged in."
      - updateLog()
    - displayPage():
      - If user credentials belong to administrator, displayPage() will display the appropriate "administrator view" layout of IoT Display
  - If current user is an Operator
    - bool userIsLoggedIn = true
    - String logEntry = "Operator *String currentUser* logged in."
      - updateLog()
    - displayPage():
      - If user credentials belong to operator, displayPage() will display the appropriate "operator view" layout of IoT Display

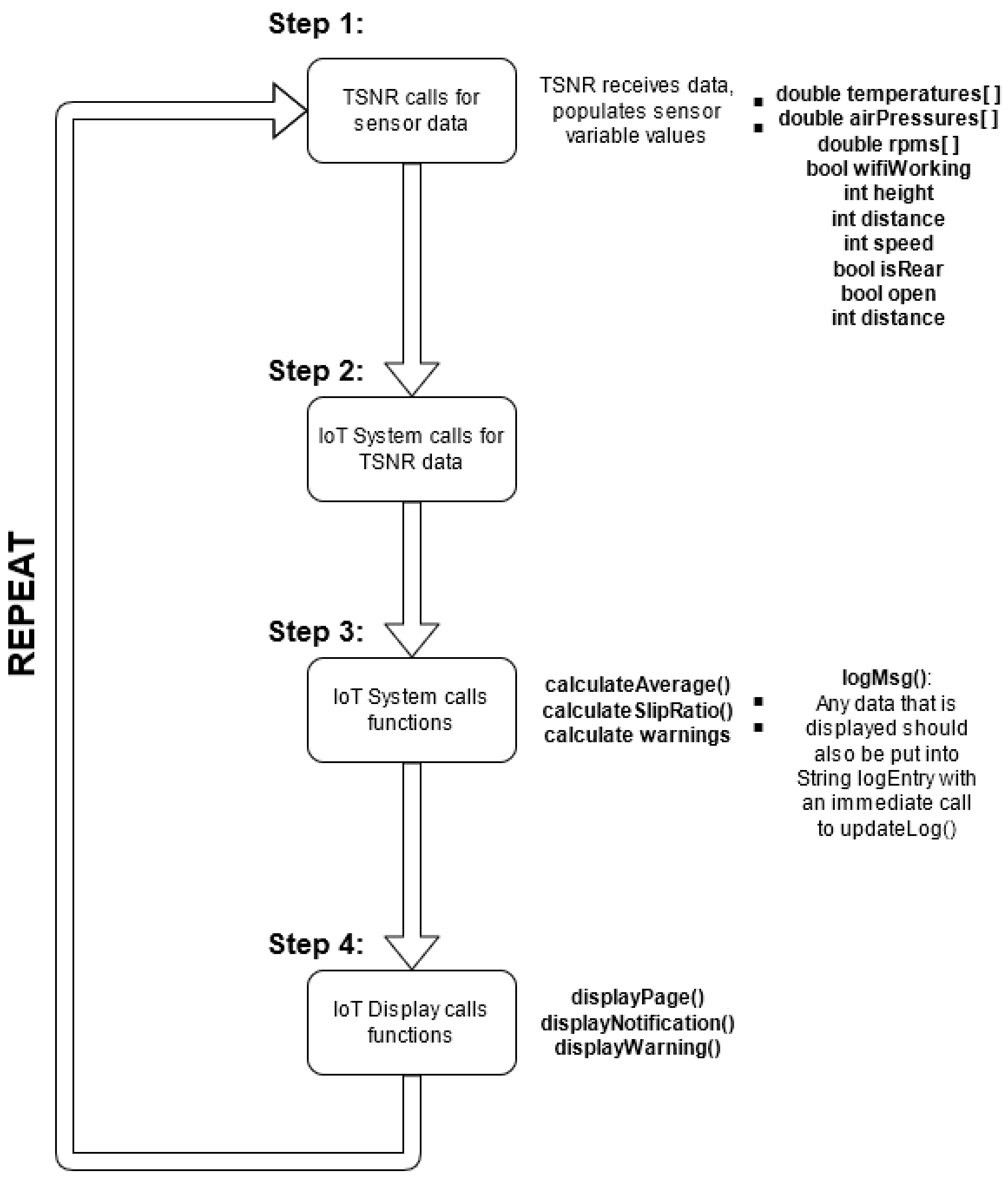
User logs out:

- bool UserIsLoggedIn = false
- String logEntry = "User *String currentUser* has logged off."

    - updateLog()
- String currentUser = “”
- displayPage():
    - If “bool userIsLoggedIn = false”, displayPage() will always only display the login page until user has provided correct credentials

IoT "Main" Loop

- These system actions comprise the bulk of our program and are run continuously in a looping fashion. A continuous loop ensures that the data we are utilizing is the most recent and in turn the most effective data possible for IoT HTR. This occurs after the user has logged if the user is an operator. Below is the general outline of how the IoT HTR System works within a continuous loop.



Most of the functions serve to digitally access what the sensors are reading (ex. getGpsSpeed(): Returns the speed of the train by accessing and returning the GPS measured speed), however there are also those that make calculations based off of the data values available from the sensors / calculations. One example of this would be calculateSlipRatio(). A sample pseudo-implementation is shown below:

```
Function calculatesSlipRatio(avgRpm, radius, speed) →
    -   Set variable “angularV” (angular velocity) to avgRpm*6
    -   Return (((angularV * radius) / speed) - 1)
```

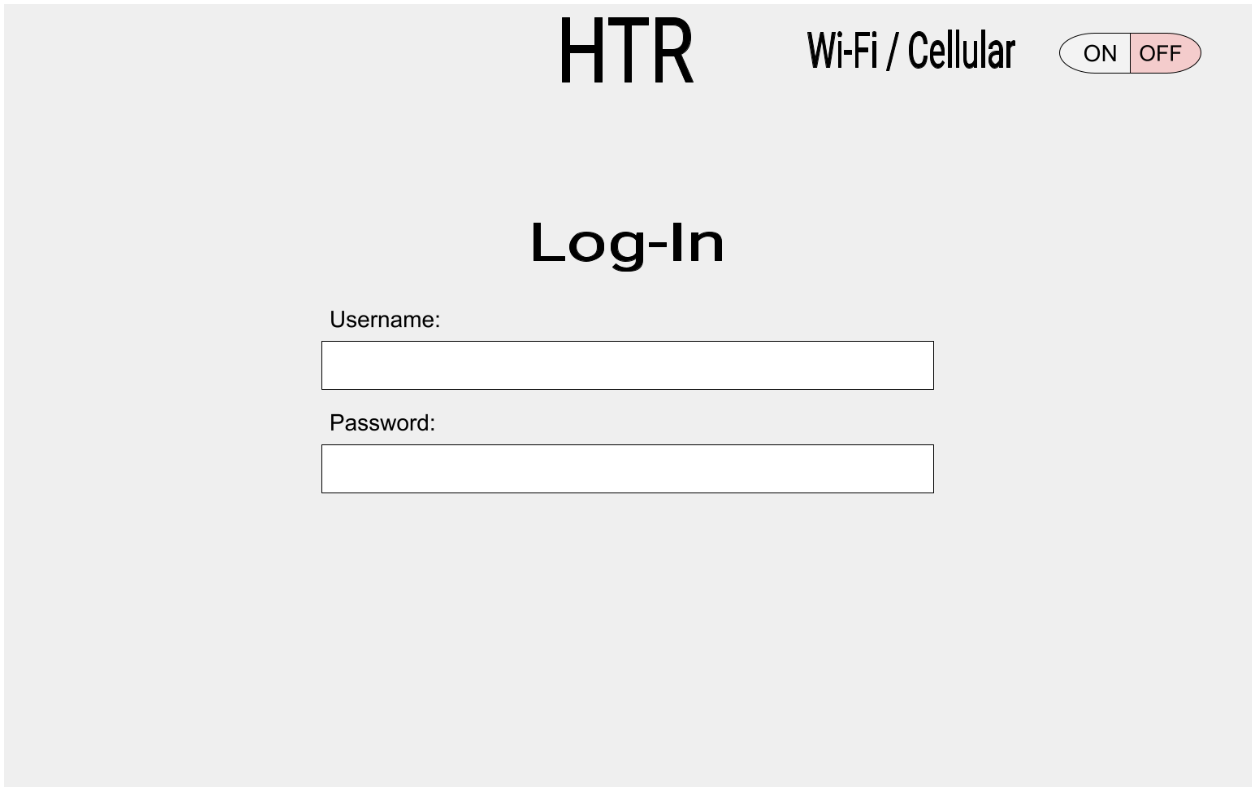
Here we are using data that is known / can be calculated in order to compute the data that we want displayed on the screen. All of the functions’ inputs, outputs, and inner workings have been highlighted within Section 4 and Section 5 of this document. For example, the scenario section from “Use Case 4” in Section 4.1, shown below, demonstrates the data flow between the radar sensor and the IoT display, and should be referenced when generating warnings / suggested actions for the operator.

Scenario:

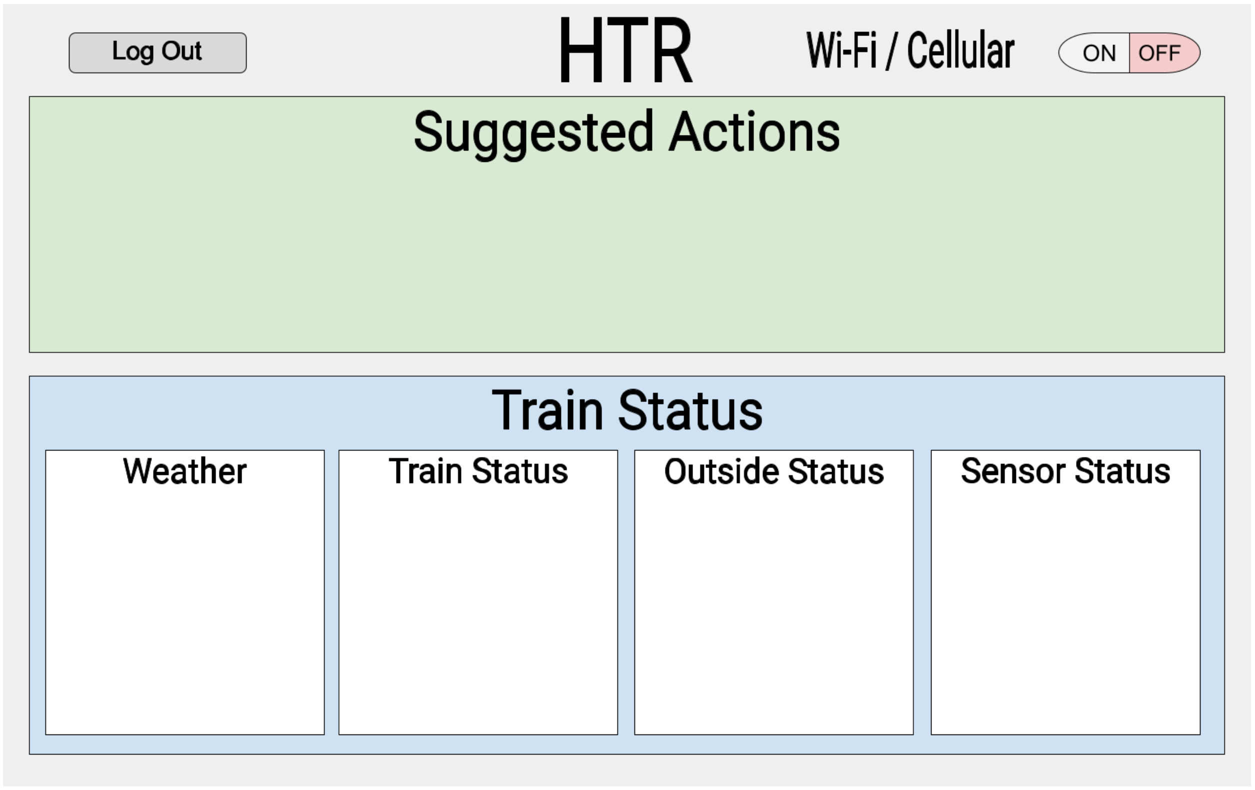
7. *Radar Sensor sends a frame to ODS.*
8. *ODS detects an object larger than 1.5 ft. tall.*
9. *ODS sends the distance and speed of the object to the TSNR.*
10. *The IoT System pulls ODS data from the TSNR.*
11. *The IoT System calculates Time Until Collision using Speed of the locomotive, distance to the object, and speed of the object.*
12. *IoT System will display information about the object to the Conductor and will give appropriate warning*
    a. *If the Time Until Collision is less than 200 seconds, the object is bigger than 1.5 feet tall, and the object is in front, the IoT will display "RED WARNING: OBJECT DETECTED AHEAD - Stop train.”*
    b. *If the Time Until Collision is less than 260 seconds but greater than or equal to 200 seconds, the object is in front, and the object is bigger than 1.5 feet tall, the IoT will display "ORANGE WARNING: OBJECT DETECTED AHEAD - Apply break to half-speed.”*
    c. *If the Time Until Collision is less than 260 seconds, the object is behind, and the object is bigger than 1.5 feet tall, the IoT will display "YELLOW WARNING: OBJECT DETECTED BEHIND - Match speed of rear object - obj.getSpeed().” where obj.getSpeed() is a call to the given objects speed parameter value.*

Note: Although we are using a Data-Flow architecture, we can implement our “filters” as objects (ex. TSNR, IoT System, Log, Display, ODS) because the Java language we use is object-oriented. This makes working with a filter’s respective variables and data fields much more efficient and streamlined than if we used text files to communicate these parameter values. Although we may use objects, the use and implementation of “filters” and “pipes” remains the same and stays within the bounds of our software being considered to be using a Data-Flow architecture.

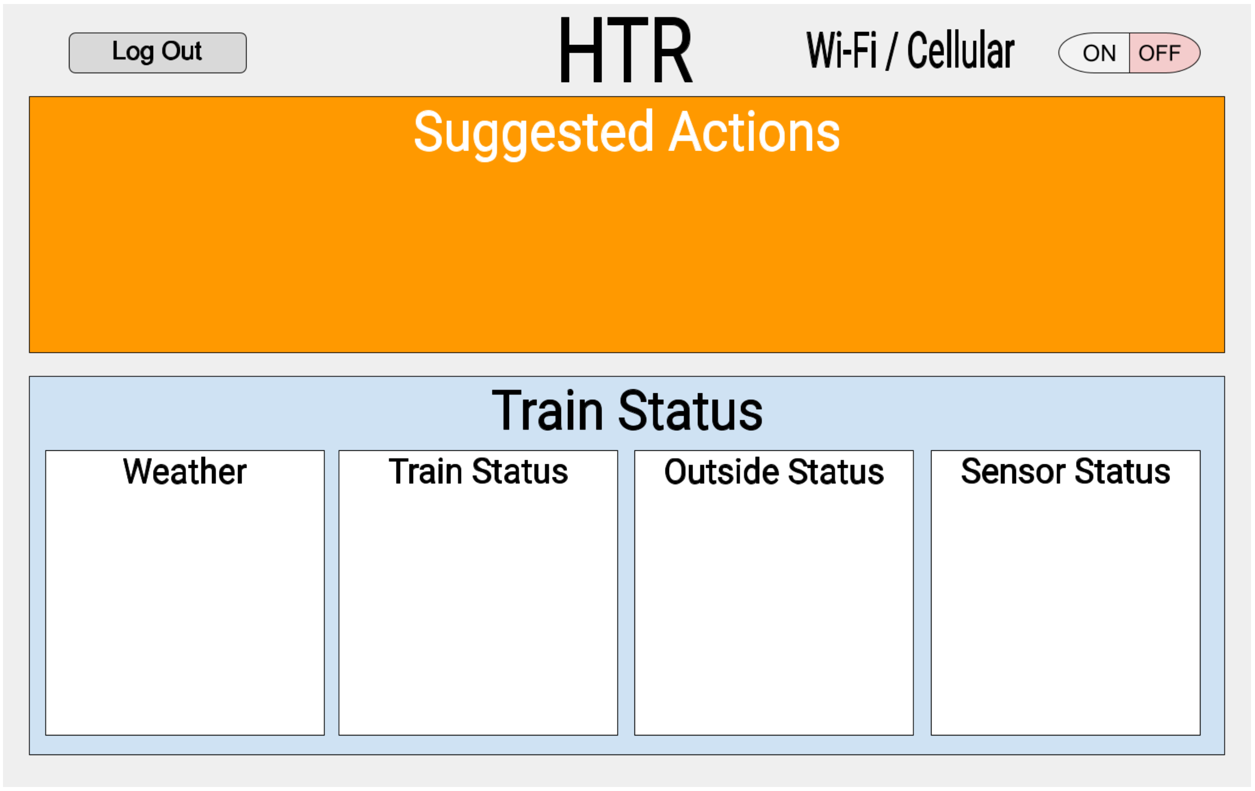
5.2.5: General Display Format



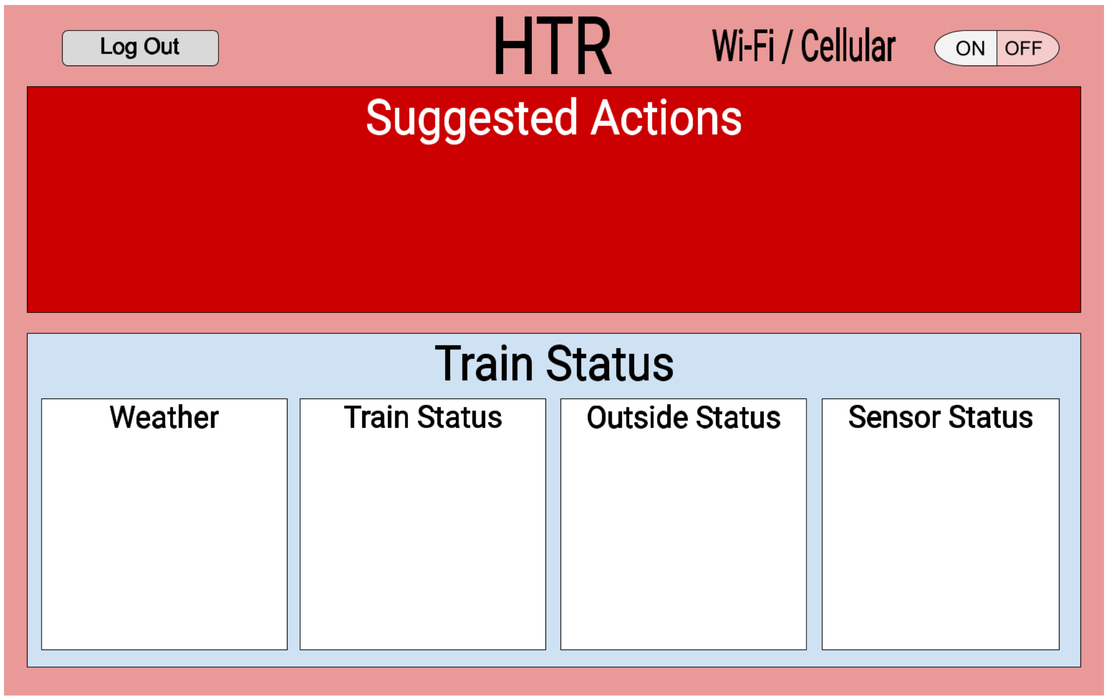
IoT System Log-In Display
Note: Wifi/Cellular Portion may be excluded as it is unnecessary



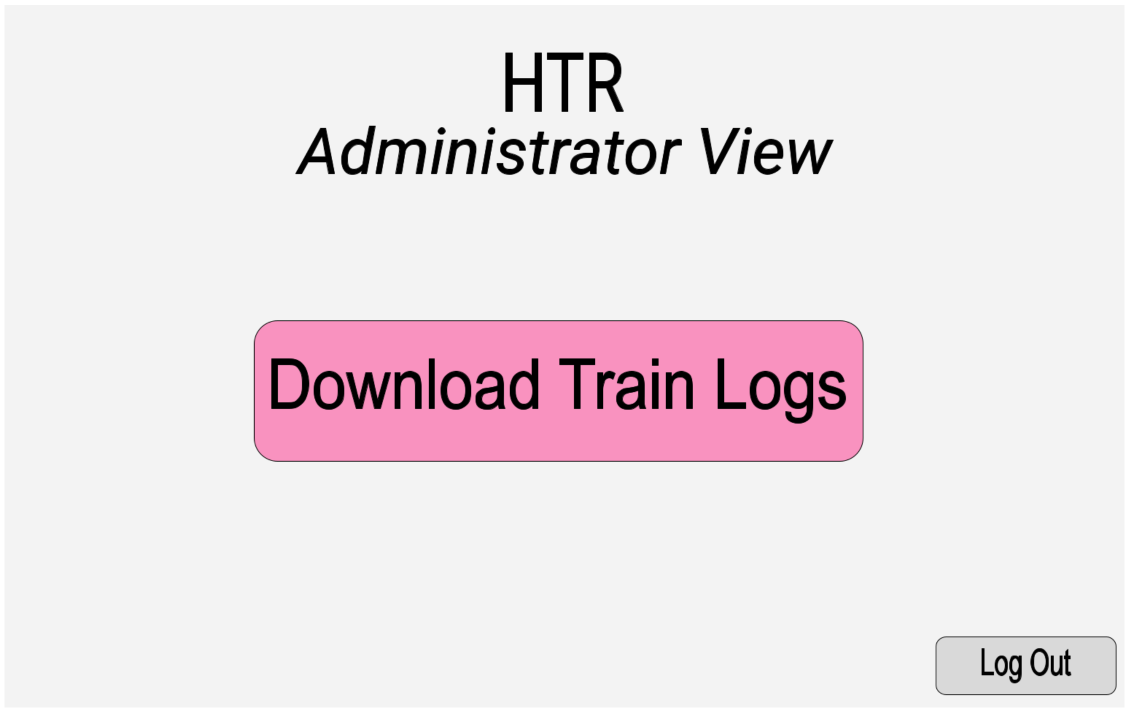
General outline of IoT System Conductor View Display shown above



General outline of IoT System Conductor View Display in "Orange State" shown above



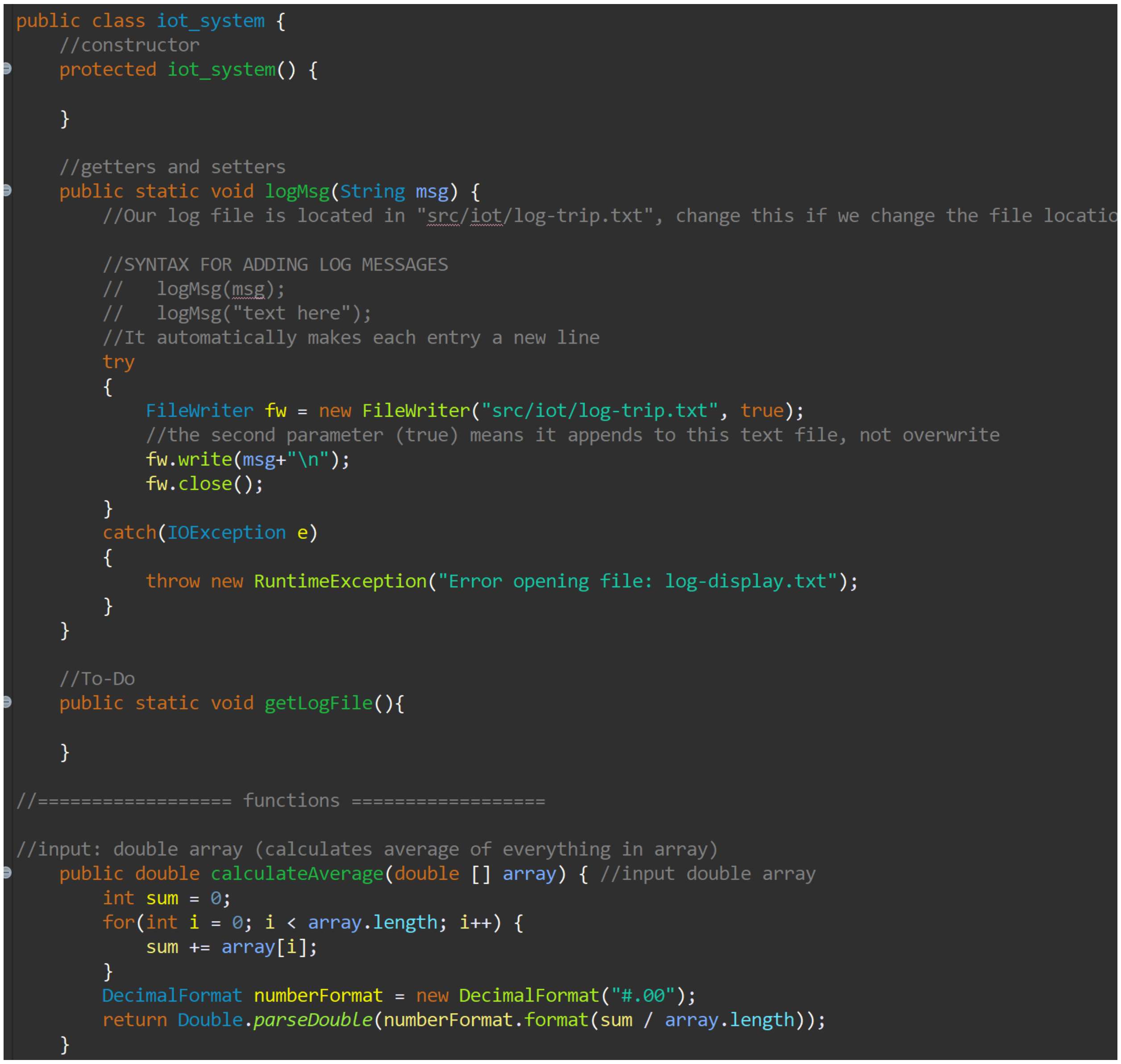
General outline of IoT System Conductor View Display in "Red State" shown above



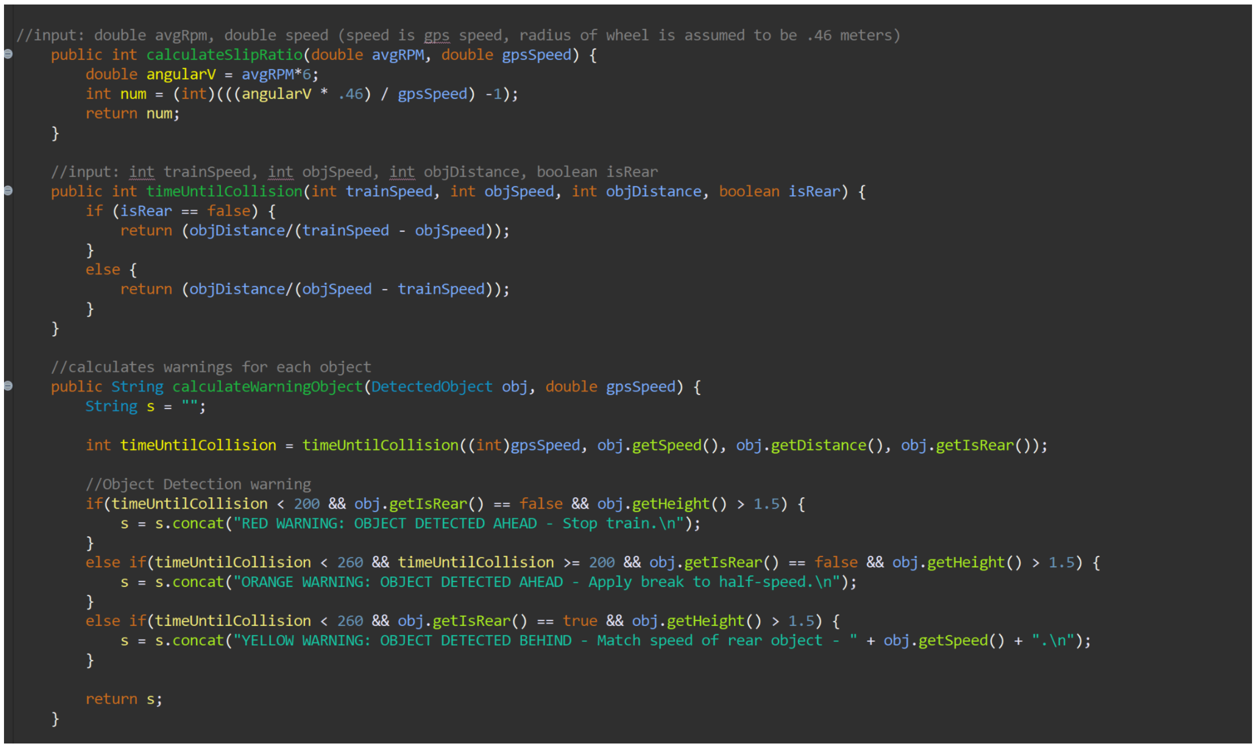
IoT System Administrator View shown above

Note: Administrator View may show same display format as Conductor view, but must have Administrator accessible "Download train Logs" button

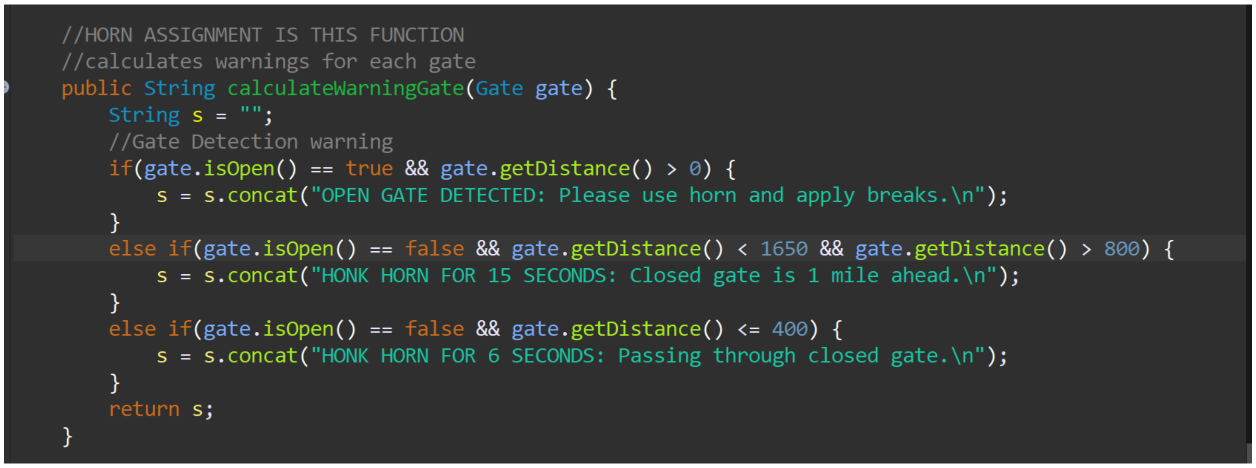
Section 6: Coding



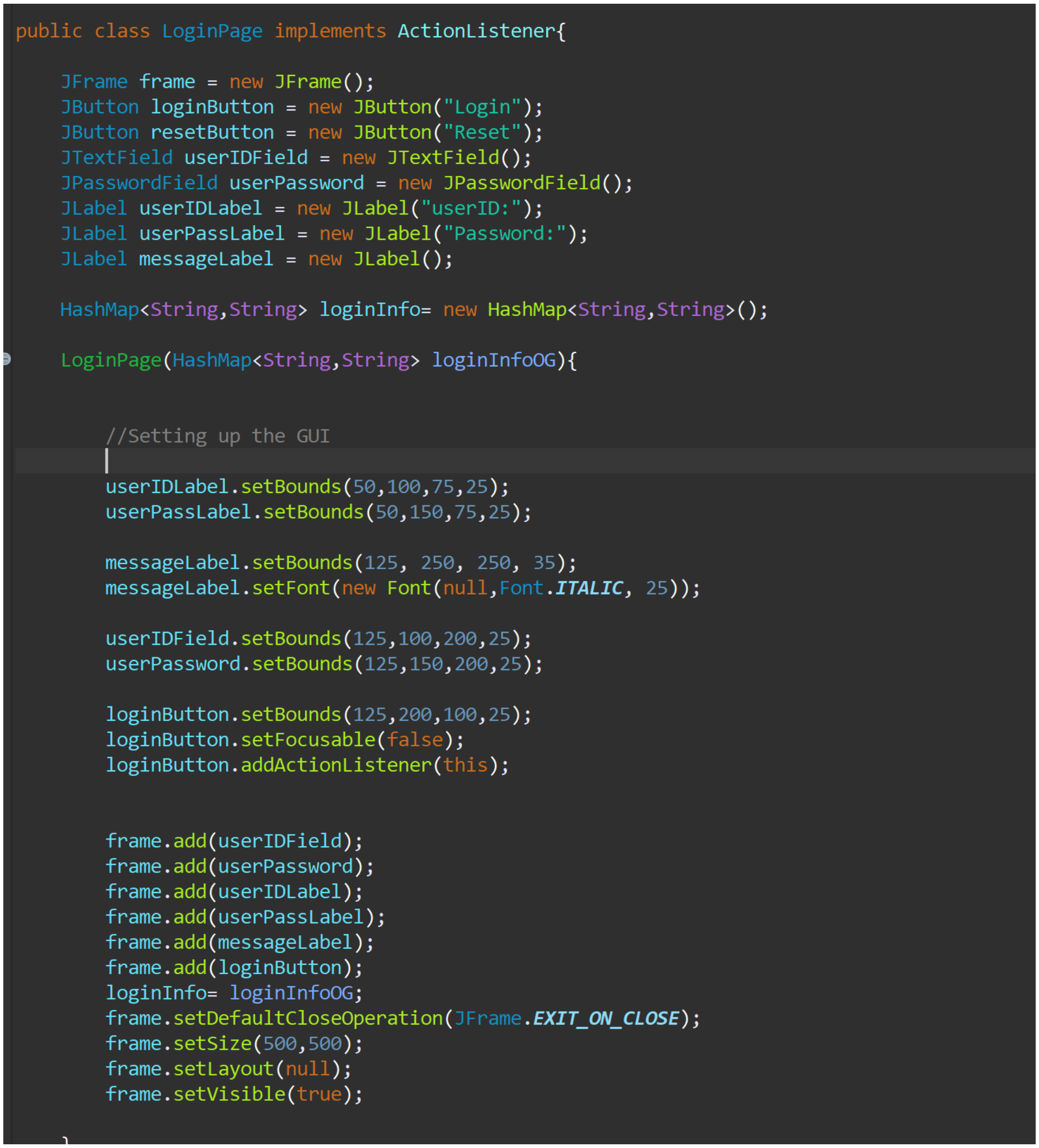
```
public class iot_system {
    //constructor
    protected iot_system() {

    }

    //getters and setters
    public static void logMsg(String msg) {
        //Our log file is located in "src/iot/log-trip.txt", change this if we change the file locatio

        //SYNTAX FOR ADDING LOG MESSAGES
        //   logMsg(msg);
        //   logMsg("text here");
        //It automatically makes each entry a new line
        try
        {
            FileWriter fw = new FileWriter("src/iot/log-trip.txt", true);
            //the second parameter (true) means it appends to this text file, not overwrite
            fw.write(msg+"\n");
            fw.close();
        }
        catch(IOException e)
        {
            throw new RuntimeException("Error opening file: log-display.txt");
        }
    }

    //To-Do
    public static void getLogFile(){

    }

//================== functions ==================

//input: double array (calculates average of everything in array)
    public double calculateAverage(double [] array) { //input double array
        int sum = 0;
        for(int i = 0; i < array.length; i++) {
            sum += array[i];
        }
        DecimalFormat numberFormat = new DecimalFormat("#.00");
        return Double.parseDouble(numberFormat.format(sum / array.length));
    }
```



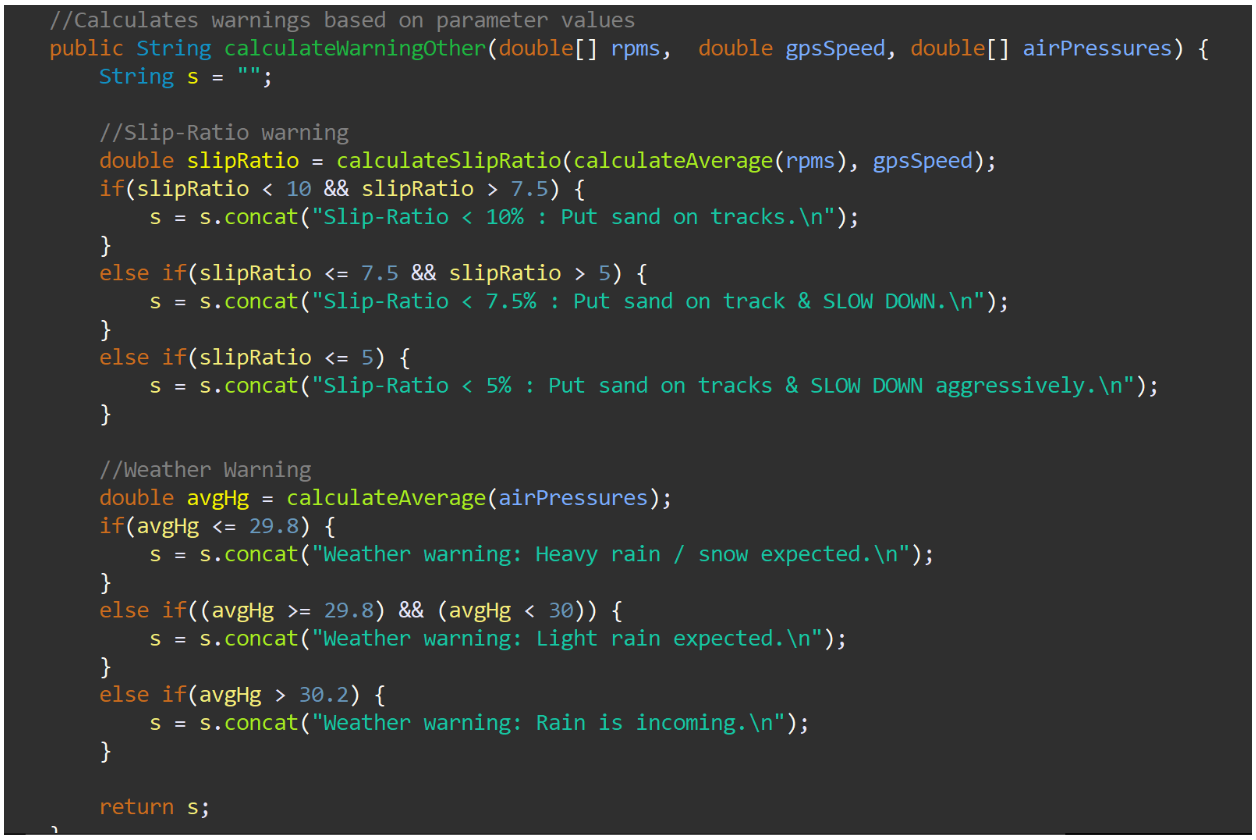
```
//input: double avgRpm, double speed (speed is gps speed, radius of wheel is assumed to be .46 meters)
    public int calculateSlipRatio(double avgRPM, double gpsSpeed) {
        double angularV = avgRPM*6;
        int num = (int)(((angularV * .46) / gpsSpeed) -1);
        return num;
    }

    //input: int trainSpeed, int objSpeed, int objDistance, boolean isRear
    public int timeUntilCollision(int trainSpeed, int objSpeed, int objDistance, boolean isRear) {
        if (isRear == false) {
            return (objDistance/(trainSpeed - objSpeed));
        }
        else {
            return (objDistance/(objSpeed - trainSpeed));
        }
    }

    //calculates warnings for each object
    public String calculateWarningObject(DetectedObject obj, double gpsSpeed) {
        String s = "";

        int timeUntilCollision = timeUntilCollision((int)gpsSpeed, obj.getSpeed(), obj.getDistance(), obj.getIsRear());

        //Object Detection warning
        if(timeUntilCollision < 200 && obj.getIsRear() == false && obj.getHeight() > 1.5) {
            s = s.concat("RED WARNING: OBJECT DETECTED AHEAD - Stop train.\n");
        }
        else if(timeUntilCollision < 260 && timeUntilCollision >= 200 && obj.getIsRear() == false && obj.getHeight() > 1.5) {
            s = s.concat("ORANGE WARNING: OBJECT DETECTED AHEAD - Apply break to half-speed.\n");
        }
        else if(timeUntilCollision < 260 && obj.getIsRear() == true && obj.getHeight() > 1.5) {
            s = s.concat("YELLOW WARNING: OBJECT DETECTED BEHIND - Match speed of rear object - " + obj.getSpeed() + ".\n");
        }

        return s;
    }
```



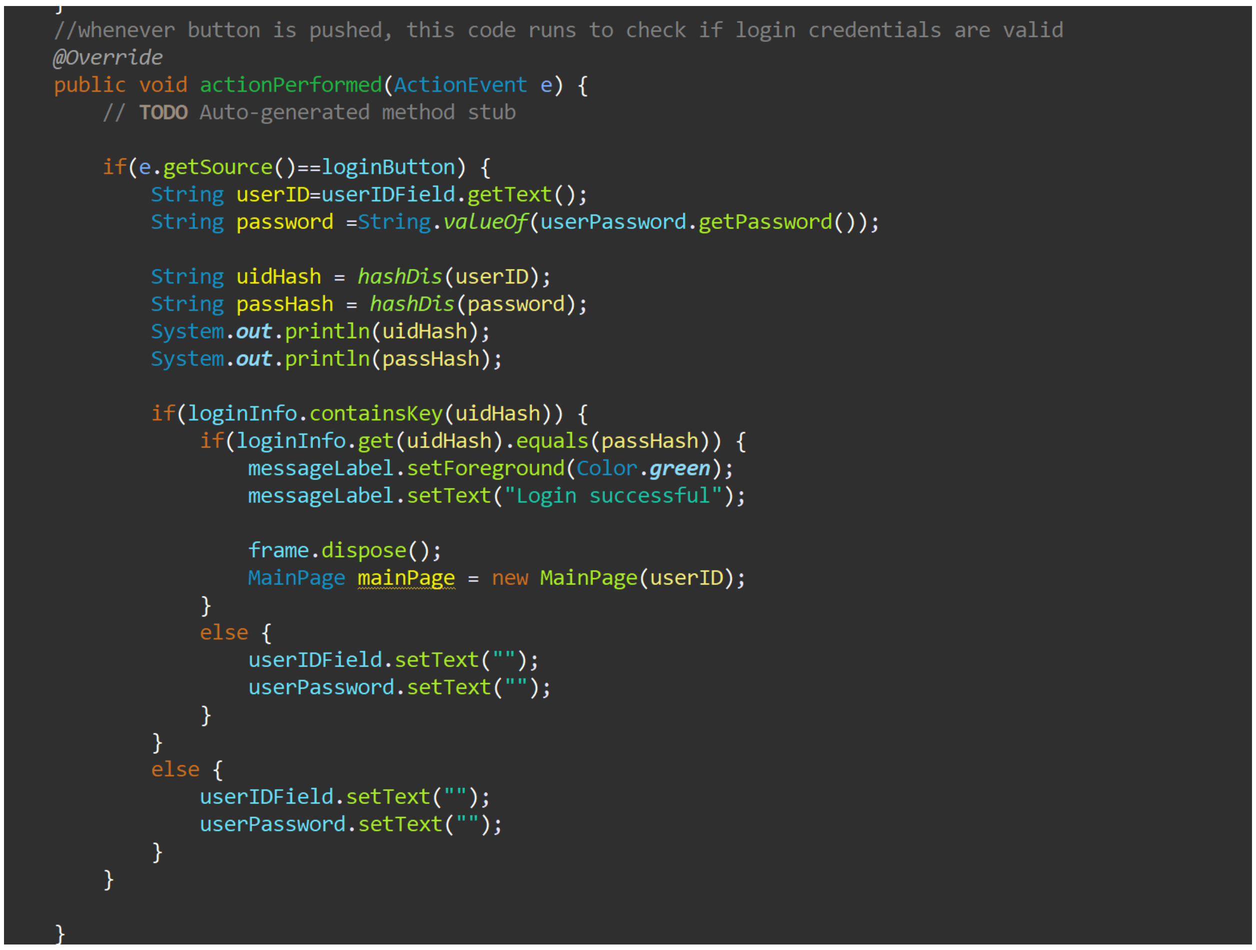
```
    //HORN ASSIGNMENT IS THIS FUNCTION
    //calculates warnings for each gate
    public String calculateWarningGate(Gate gate) {
        String s = "";
        //Gate Detection warning
        if(gate.isOpen() == true && gate.getDistance() > 0) {
            s = s.concat("OPEN GATE DETECTED: Please use horn and apply breaks.\n");
        }
        else if(gate.isOpen() == false && gate.getDistance() < 1650 && gate.getDistance() > 800) {
            s = s.concat("HONK HORN FOR 15 SECONDS: Closed gate is 1 mile ahead.\n");
        }
        else if(gate.isOpen() == false && gate.getDistance() <= 400) {
            s = s.concat("HONK HORN FOR 6 SECONDS: Passing through closed gate.\n");
        }
        return s;
    }
```



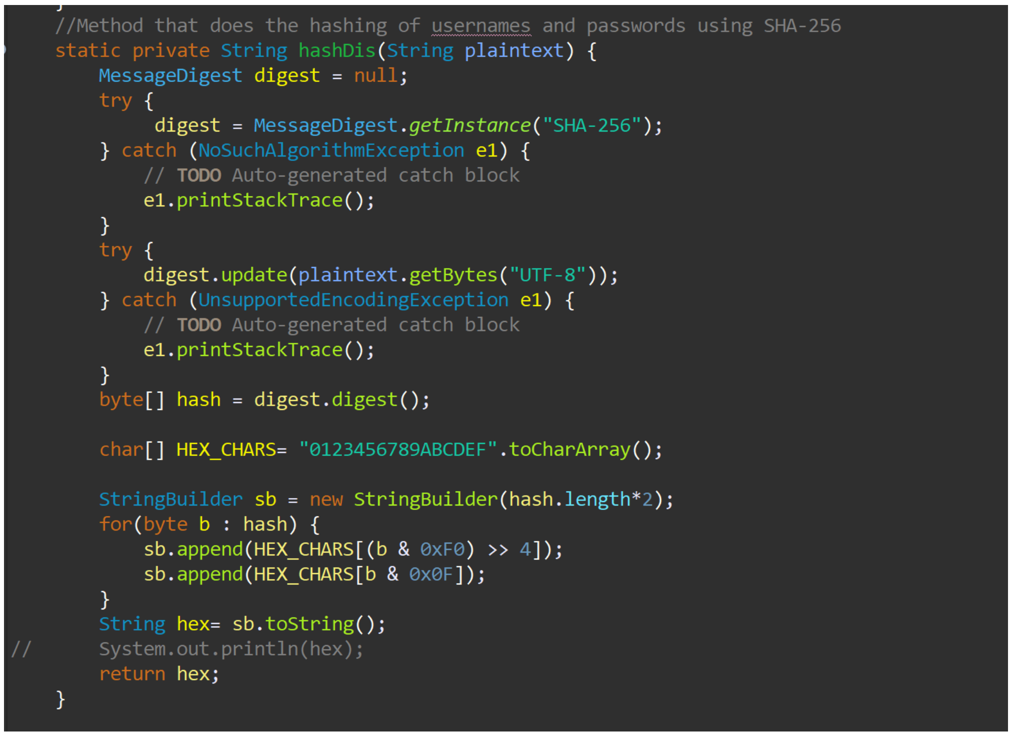
```java
public class LoginPage implements ActionListener{

    JFrame frame = new JFrame();
    JButton loginButton = new JButton("Login");
    JButton resetButton = new JButton("Reset");
    JTextField userIDField = new JTextField();
    JPasswordField userPassword = new JPasswordField();
    JLabel userIDLabel = new JLabel("userID:");
    JLabel userPassLabel = new JLabel("Password:");
    JLabel messageLabel = new JLabel();

    HashMap<String,String> loginInfo= new HashMap<String,String>();

    LoginPage(HashMap<String,String> loginInfoOG){


        //Setting up the GUI

        userIDLabel.setBounds(50,100,75,25);
        userPassLabel.setBounds(50,150,75,25);

        messageLabel.setBounds(125, 250, 250, 35);
        messageLabel.setFont(new Font(null,Font.ITALIC, 25));

        userIDField.setBounds(125,100,200,25);
        userPassword.setBounds(125,150,200,25);

        loginButton.setBounds(125,200,100,25);
        loginButton.setFocusable(false);
        loginButton.addActionListener(this);


        frame.add(userIDField);
        frame.add(userPassword);
        frame.add(userIDLabel);
        frame.add(userPassLabel);
        frame.add(messageLabel);
        frame.add(loginButton);
        loginInfo= loginInfoOG;
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(500,500);
        frame.setLayout(null);
        frame.setVisible(true);

```



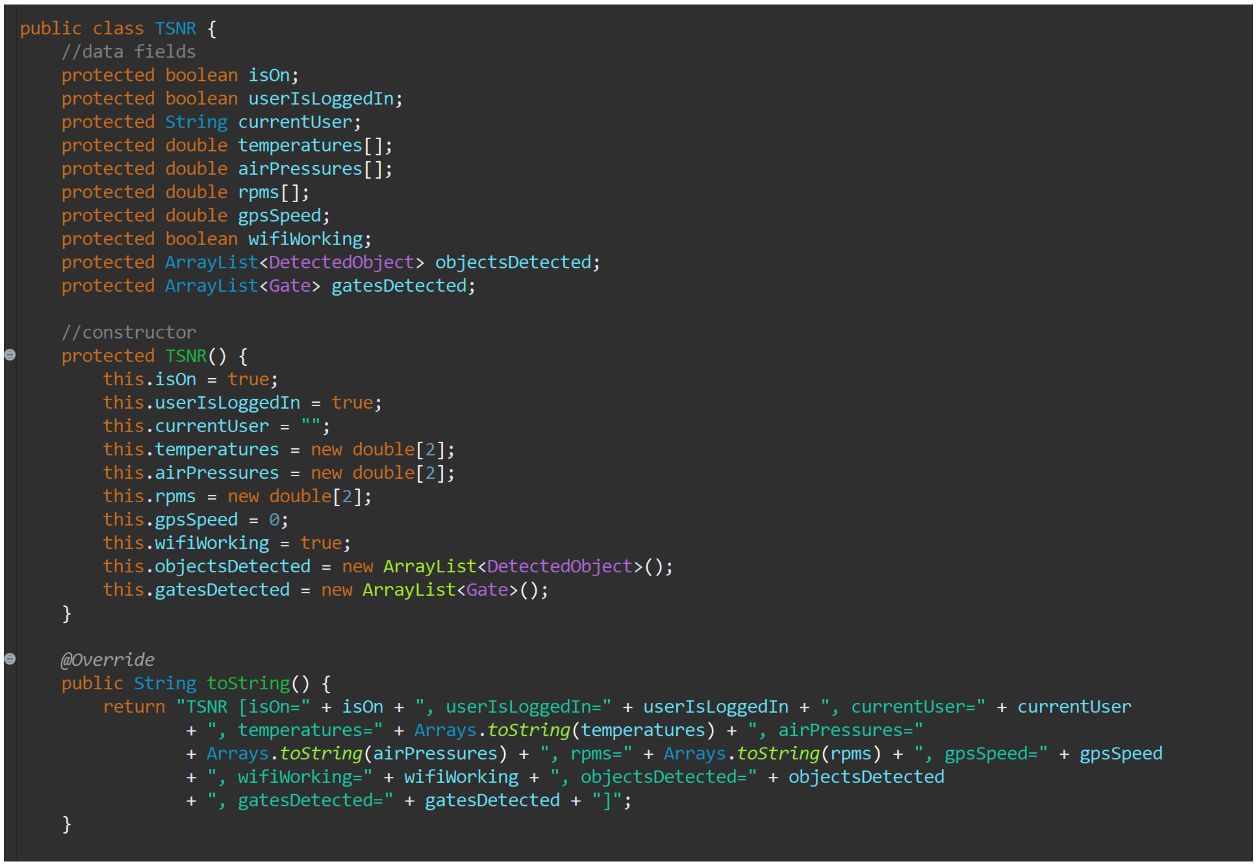
```
//Calculates warnings based on parameter values
public String calculateWarningOther(double[] rpms,  double gpsSpeed, double[] airPressures) {
    String s = "";

    //Slip-Ratio warning
    double slipRatio = calculateSlipRatio(calculateAverage(rpms), gpsSpeed);
    if(slipRatio < 10 && slipRatio > 7.5) {
        s = s.concat("Slip-Ratio < 10% : Put sand on tracks.\n");
    }
    else if(slipRatio <= 7.5 && slipRatio > 5) {
        s = s.concat("Slip-Ratio < 7.5% : Put sand on track & SLOW DOWN.\n");
    }
    else if(slipRatio <= 5) {
        s = s.concat("Slip-Ratio < 5% : Put sand on tracks & SLOW DOWN aggressively.\n");
    }

    //Weather Warning
    double avgHg = calculateAverage(airPressures);
    if(avgHg <= 29.8) {
        s = s.concat("Weather warning: Heavy rain / snow expected.\n");
    }
    else if((avgHg >= 29.8) && (avgHg < 30)) {
        s = s.concat("Weather warning: Light rain expected.\n");
    }
    else if(avgHg > 30.2) {
        s = s.concat("Weather warning: Rain is incoming.\n");
    }

    return s;
```



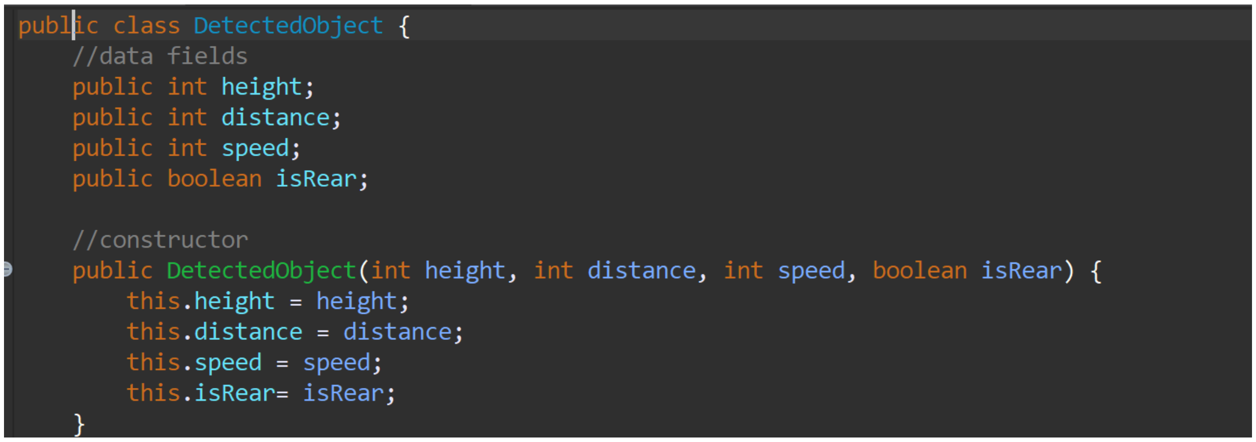
```
    //whenever button is pushed, this code runs to check if login credentials are valid
    @Override
    public void actionPerformed(ActionEvent e) {
        // TODO Auto-generated method stub

        if(e.getSource()==loginButton) {
            String userID=userIDField.getText();
            String password =String.valueOf(userPassword.getPassword());

            String uidHash = hashDis(userID);
            String passHash = hashDis(password);
            System.out.println(uidHash);
            System.out.println(passHash);

            if(loginInfo.containsKey(uidHash)) {
                if(loginInfo.get(uidHash).equals(passHash)) {
                    messageLabel.setForeground(Color.green);
                    messageLabel.setText("Login successful");

                    frame.dispose();
                    MainPage mainPage = new MainPage(userID);
                }
                else {
                    userIDField.setText("");
                    userPassword.setText("");
                }
            }
            else {
                userIDField.setText("");
                userPassword.setText("");
            }
        }

    }
```



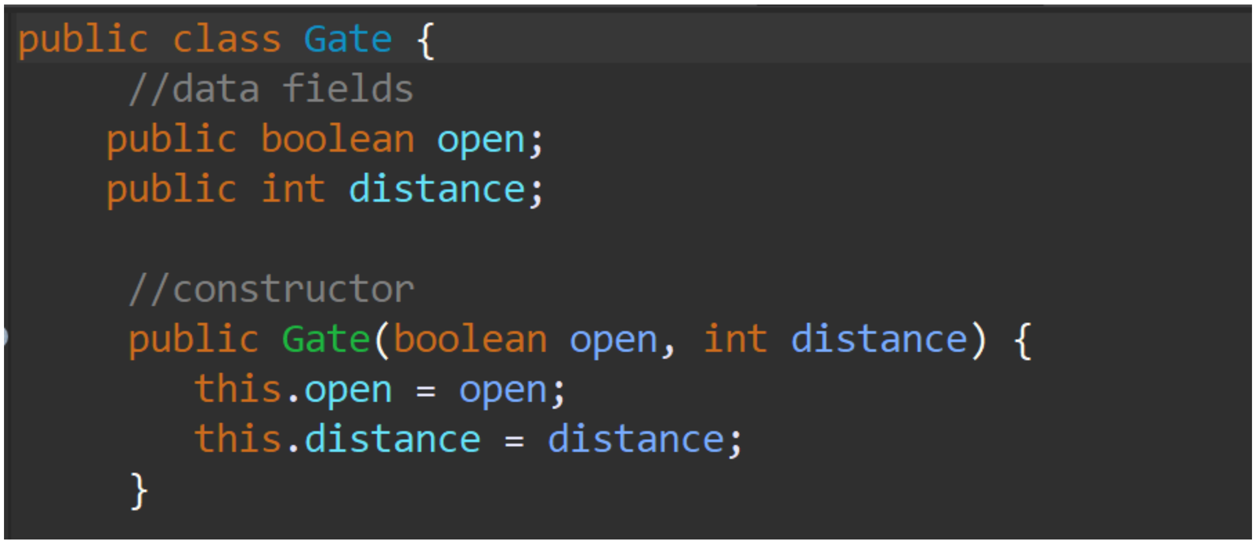
```
    //Method that does the hashing of usernames and passwords using SHA-256
    static private String hashDis(String plaintext) {
        MessageDigest digest = null;
        try {
             digest = MessageDigest.getInstance("SHA-256");
        } catch (NoSuchAlgorithmException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        }
        try {
            digest.update(plaintext.getBytes("UTF-8"));
        } catch (UnsupportedEncodingException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        }
        byte[] hash = digest.digest();

        char[] HEX_CHARS= "0123456789ABCDEF".toCharArray();

        StringBuilder sb = new StringBuilder(hash.length*2);
        for(byte b : hash) {
            sb.append(HEX_CHARS[(b & 0xF0) >> 4]);
            sb.append(HEX_CHARS[b & 0x0F]);
        }
        String hex= sb.toString();
//      System.out.println(hex);
        return hex;
    }
```



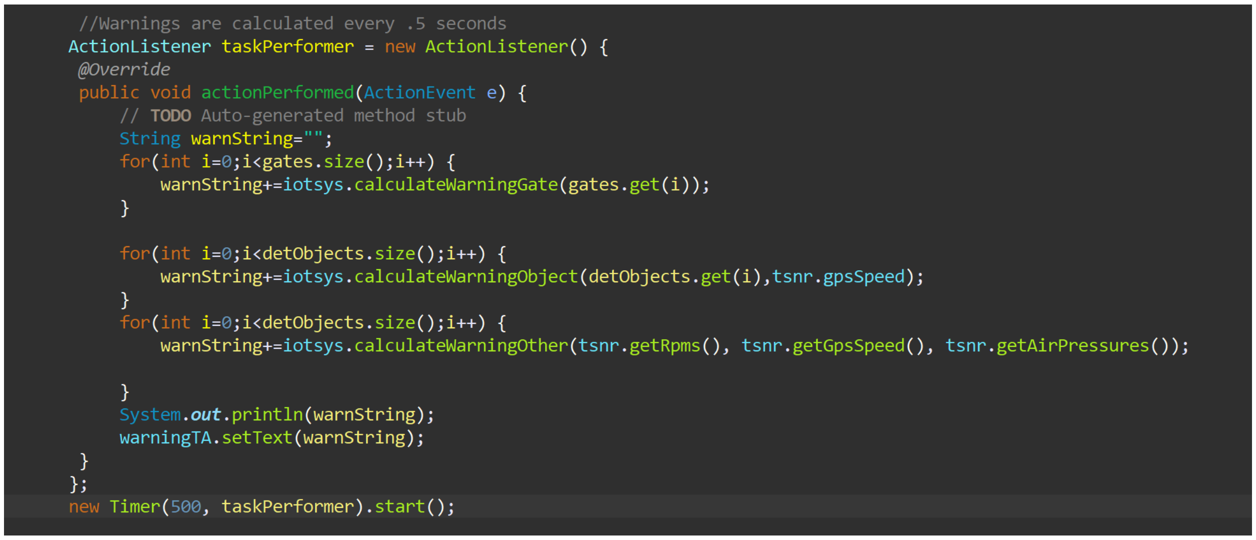
```
public class TSNR {
    //data fields
    protected boolean isOn;
    protected boolean userIsLoggedIn;
    protected String currentUser;
    protected double temperatures[];
    protected double airPressures[];
    protected double rpms[];
    protected double gpsSpeed;
    protected boolean wifiWorking;
    protected ArrayList<DetectedObject> objectsDetected;
    protected ArrayList<Gate> gatesDetected;

    //constructor
    protected TSNR() {
        this.isOn = true;
        this.userIsLoggedIn = true;
        this.currentUser = "";
        this.temperatures = new double[2];
        this.airPressures = new double[2];
        this.rpms = new double[2];
        this.gpsSpeed = 0;
        this.wifiWorking = true;
        this.objectsDetected = new ArrayList<DetectedObject>();
        this.gatesDetected = new ArrayList<Gate>();
    }

    @Override
    public String toString() {
        return "TSNR [isOn=" + isOn + ", userIsLoggedIn=" + userIsLoggedIn + ", currentUser=" + currentUser
                + ", temperatures=" + Arrays.toString(temperatures) + ", airPressures="
                + Arrays.toString(airPressures) + ", rpms=" + Arrays.toString(rpms) + ", gpsSpeed=" + gpsSpeed
                + ", wifiWorking=" + wifiWorking + ", objectsDetected=" + objectsDetected
                + ", gatesDetected=" + gatesDetected + "]";
    }
```



```
public class DetectedObject {
    //data fields
    public int height;
    public int distance;
    public int speed;
    public boolean isRear;

    //constructor
    public DetectedObject(int height, int distance, int speed, boolean isRear) {
        this.height = height;
        this.distance = distance;
        this.speed = speed;
        this.isRear= isRear;
    }
```



```
public class Gate {
     //data fields
    public boolean open;
    public int distance;

     //constructor
     public Gate(boolean open, int distance) {
        this.open = open;
        this.distance = distance;
     }
```



```
     //Warnings are calculated every .5 seconds
    ActionListener taskPerformer = new ActionListener() {
     @Override
     public void actionPerformed(ActionEvent e) {
         // TODO Auto-generated method stub
         String warnString="";
         for(int i=0;i<gates.size();i++) {
             warnString+=iotsys.calculateWarningGate(gates.get(i));
         }

         for(int i=0;i<detObjects.size();i++) {
             warnString+=iotsys.calculateWarningObject(detObjects.get(i),tsnr.gpsSpeed);
         }
         for(int i=0;i<detObjects.size();i++) {
             warnString+=iotsys.calculateWarningOther(tsnr.getRpms(), tsnr.getGpsSpeed(), tsnr.getAirPressures());

         }
         System.out.println(warnString);
         warningTA.setText(warnString);
     }
    };
    new Timer(500, taskPerformer).start();
```

Section 7: Testing

Test 1

Test 1:

- ➔ Description:
  - ◆ *User attempts to log in to IoT HTR System*
- ➔ Expected Result:
  - ◆ *IoT Display displays the appropriate page layout (Operator View or Administrator View) of IoT HTR System*

a. The user provides the correct credentials and is verified to be an administrator. Since the user is an administrator, the IoT Display displays the "Administrator View" of the IoT HTR System.

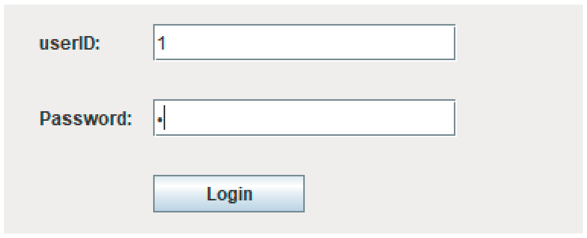
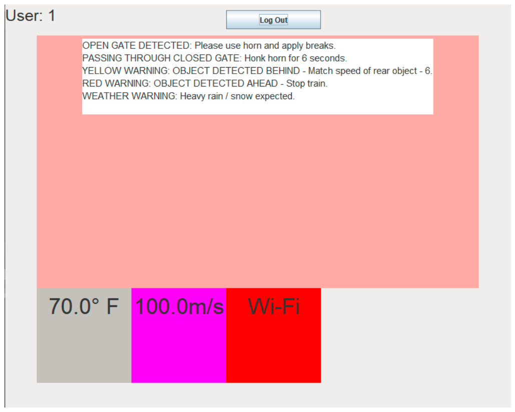
b. The user provides the correct credentials and is verified to be an operator. Since the user is an operator, the IoT Display displays the "Operator View" of the IoT HTR System.

c. The user provides the incorrect credentials when attempting to log in. Since the user is not recognized, an "incorrect credentials" message is displayed and the user is once again prompted to attempt to log into the IoT HTR System.

Test 1 Result

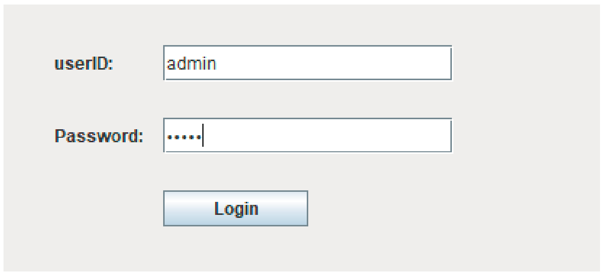
How to test:

- When starting up the program or after logging out of a user account, you will be met with a "Log-In" screen:
  - For "operator" credentials, enter:
    - Username: 1
    - Password: 1
    - **Result**: Program opens up IoT Operator View of our software

 userID: 1 Password: • *Operator Credentials entered (User presses "Login" button after entering credentials)*	 User: 1 OPEN GATE DETECTED: Please use horn and apply breaks. PASSING THROUGH CLOSED GATE: Honk horn for 6 seconds. YELLOW WARNING: OBJECT DETECTED BEHIND - Match speed of rear object - 6. RED WARNING: OBJECT DETECTED AHEAD - Stop train. WEATHER WARNING: Heavy rain / snow expected. 70.0° F 100.0m/s Wi-Fi *Operator View of software is displayed*

- Alternatively:

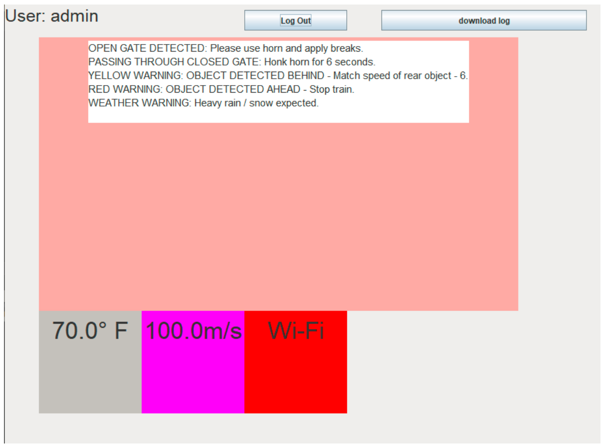
- For “administrator” credentials, enter:
    - Username: admin
    - Password: admin
    - **Result**: Program opens up to IoT Administrator View of our software



userID: admin

Password: •••••

Administrator Credentials entered (User presses “Login” button after entering credentials)



User: admin

```
OPEN GATE DETECTED: Please use horn and apply breaks.
PASSING THROUGH CLOSED GATE: Honk horn for 6 seconds.
YELLOW WARNING: OBJECT DETECTED BEHIND - Match speed of rear object - 6.
RED WARNING: OBJECT DETECTED AHEAD - Stop train.
WEATHER WARNING: Heavy rain / snow expected.
```

Administrator View of software is displayed (distinguishable by the “download log” button in the top right corner)

Test 2

Test 2:

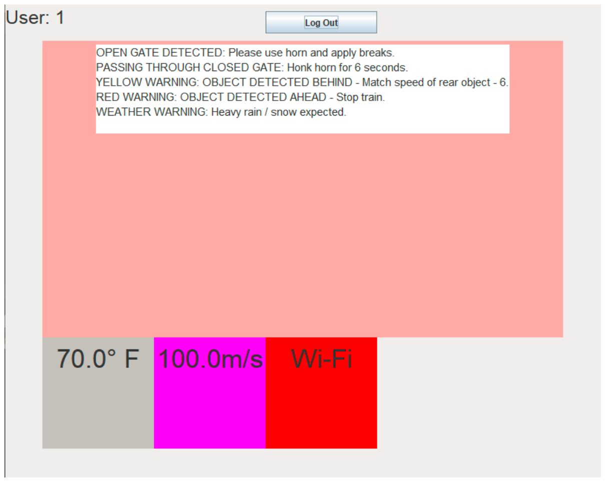
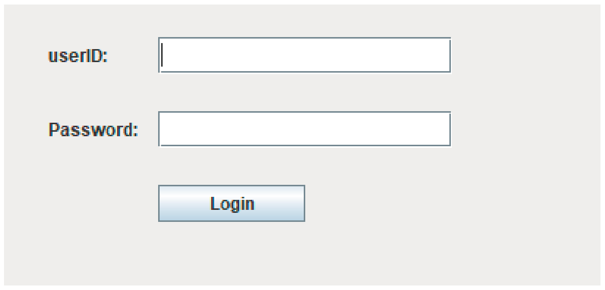
- ➔ Description:
    - ◆ *User attempts to log out of IoT HTR System*
- ➔ Expected Result:
    - ◆ *IoT Display displays the log-in prompt page of IoT HTR System*

a. The user has successfully logged in and now intends to log out. The user clicks the log-out button. Since the user is now logged out, the IoT Display displays the log-in prompt page of the IoT HTR System.

Test 2 Result

How to test:

- While in either the “Operator View” or the “Administrator View” (logged in as user):
    - Press “Log-Out” button
    - **Result**: Program returns user to “Log-In” prompt page

 OPEN GATE DETECTED: Please use horn and apply breaks. PASSING THROUGH CLOSED GATE: Honk horn for 6 seconds. YELLOW WARNING: OBJECT DETECTED BEHIND - Match speed of rear object - 6. RED WARNING: OBJECT DETECTED AHEAD - Stop train. WEATHER WARNING: Heavy rain / snow expected. *User is in either Operator View or Administrator View, and user presses "Log Out" button*	 *Program returns user to "Log-In" prompt page, both input boxes blank and awaiting input*

Test 3

Test 3:

- ➔ Description:
  - ◆ *IoT System processes array of data to get an average sensor reading*
- ➔ *Expected Result;*
  - ◆ *IoT System produces single float that is the average of an inputted array of floats*

a. The TSNR sends new sensor data to the IoT system.

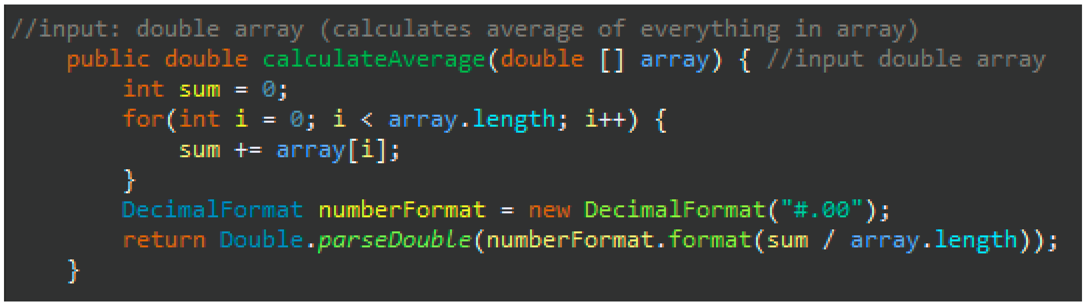
b. The IoT System then passes whatever data that is applicable to the calculateAverage() function, which would process the data and return the average reading.

c. This reading is then able to be used to determine necessary warnings and able to be stored in the Log.

Test 3 Result

How to test:

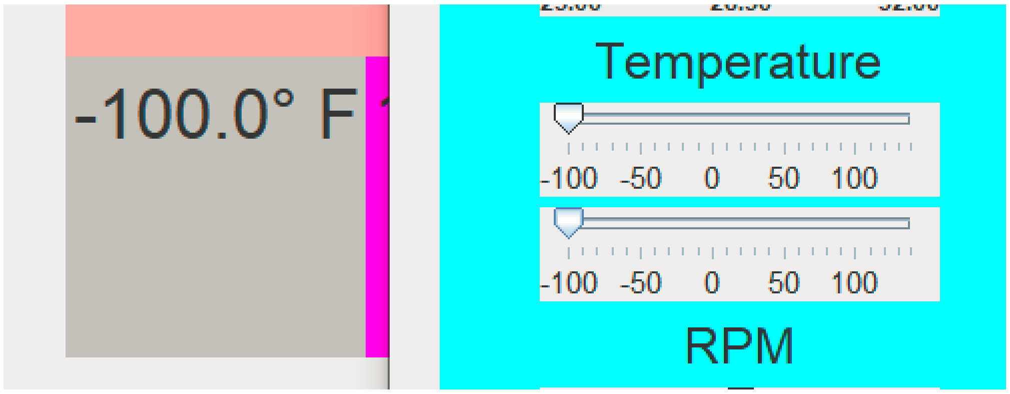
- Our software calculates the average of multiple sensor inputs, such as air pressure, temperature, and wheel rotations per minute (rpms)
- **Result**: Average from array of values is calculated to two decimal places



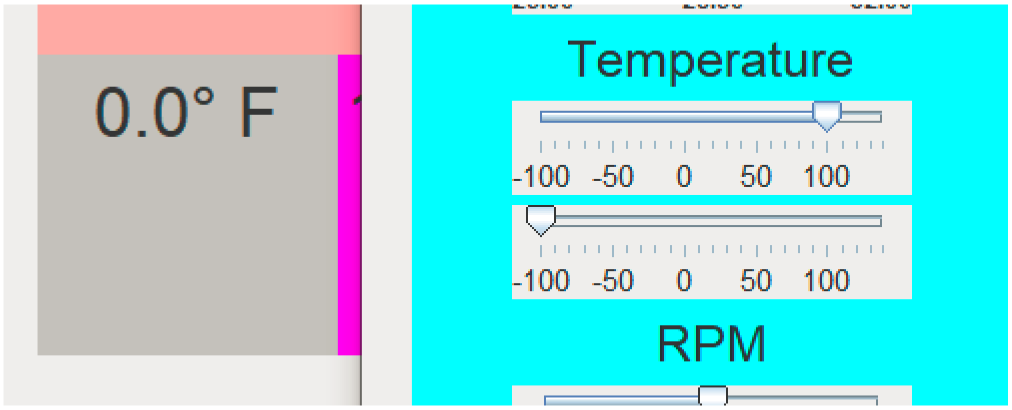
```
//input: double array (calculates average of everything in array)
    public double calculateAverage(double [] array) { //input double array
        int sum = 0;
        for(int i = 0; i < array.length; i++) {
            sum += array[i];
        }
        DecimalFormat numberFormat = new DecimalFormat("#.00");
        return Double.parseDouble(numberFormat.format(sum / array.length));
    }
```

calculateAverage() function shown above

- We can test this calculateAverage() function by modifying the two "Temperature" sliders in the controller panel, and observing if the Temperature displayed in the User's view is an average of the two values
    - The temperature displayed is the average value from the two temperature sliders within the Controller panel



Temperature displayed in bottom left corner of User View is constantly updated to match average value of Temperature sliders



With the given temperature values of "-100" and "100", IoT System calculates the correct average to be 0 degrees Fahrenheit

Test 4

Test 4:

- ➔ Description:
  - ◆ *IoT System calculates Slip Ratio*
- ➔ Expected Result:
  - ◆ *IoT System produces a percentage value for slip ratio*

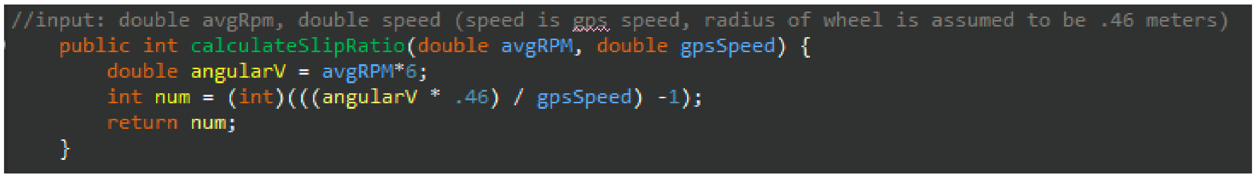
a. GPS sensor sends speed of locomotive to IoT System.
b. All Wheel Sensors send RPM to the TSNR.
c. The IoT System pulls Wheel Sensor data from the TSNR.
d. The IoT System calculates the average RPM.
e. The IoT System reads the diameter of the wheel from the "Rules of Operation" file.
f. The IoT System calculates the speed of the locomotive using the GPS speed data.
g. The IoT System calculates angular velocity of wheels using average of all wheel's RPMs.
   i. Angular velocity = avg. wheel RPM * 6
h. The IoT System calculates the slip ratio using the SAE J670 slip ratio equation.
   i. $Slip\ Ratio\ \% \ = \ (\frac{Angular\ Velocity * Wheel\ Radius}{Speed\ of\ Train} - 1) \ * \ 100\%$
i. Calculated slip ratio is returned for use in other functions which require it

Test 4 Result

How to test:

- The IoT System calculates the slip ratio using the SAE J670 slip ratio equation:

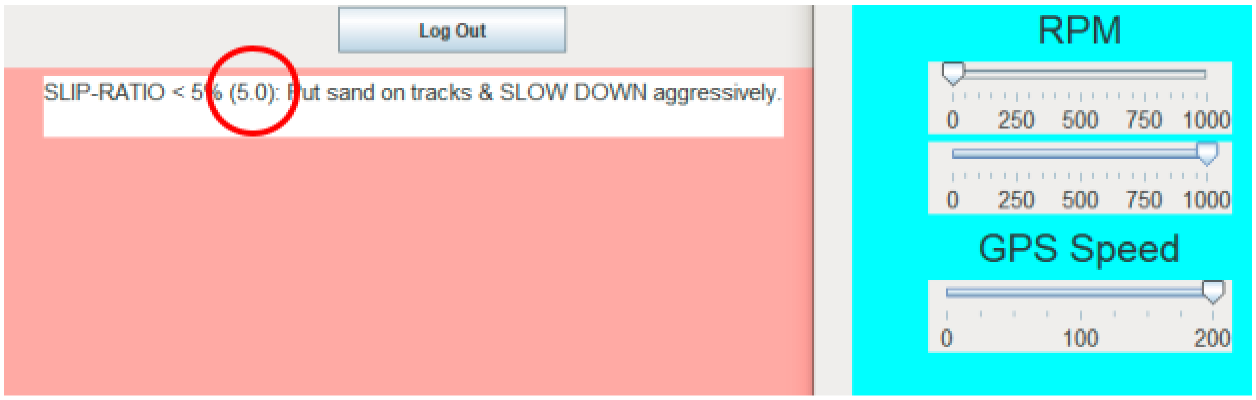
$$(\,Slip\ Ratio\ \% \ = \ (\frac{Angular\ Velocity * Wheel\ Radius}{Speed\ of\ Train} - 1) \ * \ 100\%\,)$$



```
//input: double avgRpm, double speed (speed is gps speed, radius of wheel is assumed to be .46 meters)
    public int calculateSlipRatio(double avgRPM, double gpsSpeed) {
        double angularV = avgRPM*6;
        int num = (int)(((angularV * .46) / gpsSpeed) -1);
        return num;
    }
```

calculateSlipRatio() function shown above

- To test this,
  - Set first RPM slider to 0, and set second RPM slider to 1000.
  - Set the GPS Speed to 200
    - Slip ratio should be "(500*6*.46 / 200) - 1" (equal to 5.0)
      - <u>VALUE IS ROUNDED DOWN to the nearest integer</u>, ensures maximum safety in variable conditions
  - **Result**: IoT System correctly calculates and displays slip ratio


SLIP-RATIO < 5% (5.0): Put sand on tracks & SLOW DOWN aggressively.

IoT System correctly calculates the slip ratio to be "5.0"

Test 5

Test 5:

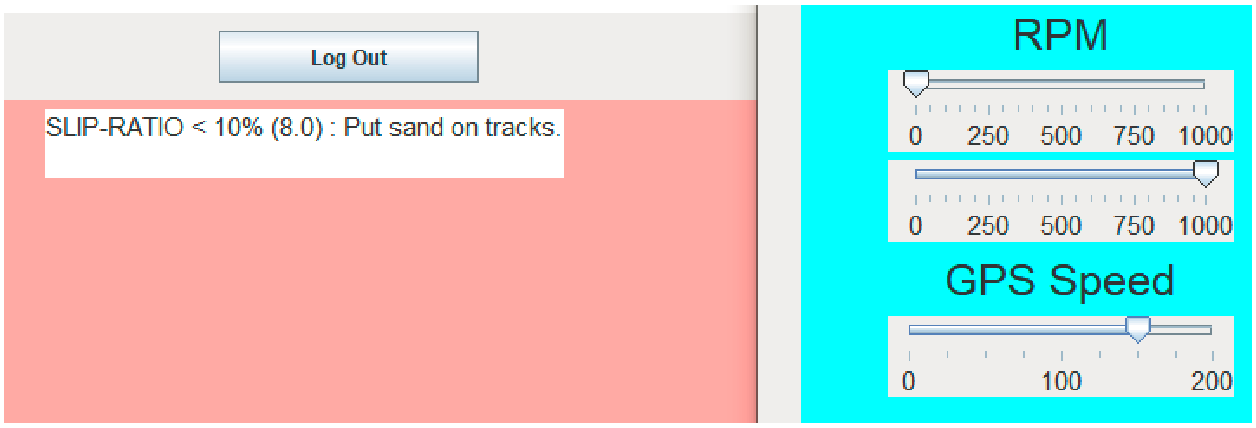
➔ Description:
  ◆ *IoT System processes data from TSNR and outputs appropriate warnings for Slip Ratio*
➔ Expected Result:
  ◆ *IoT System produces a string that is presented on IoT Display*

a. The IoT System calculates the slip ratio
b. The IoT System will generate a string based on the slip ratio and the recommended action for that slip ratio.
   i. If the slip ratio is <10% , notify the conductor to put sand on tracks.
   ii. If the slip ratio is <7.5%, notify the conductor to put sand on tracks and slow down.
   iii. If the slip ratio is <5%, notify the conductor to put sand on tracks and slow down aggressively.
c. The string is added to a single string, which is then displayed on the IoT Display

Test 5 Result

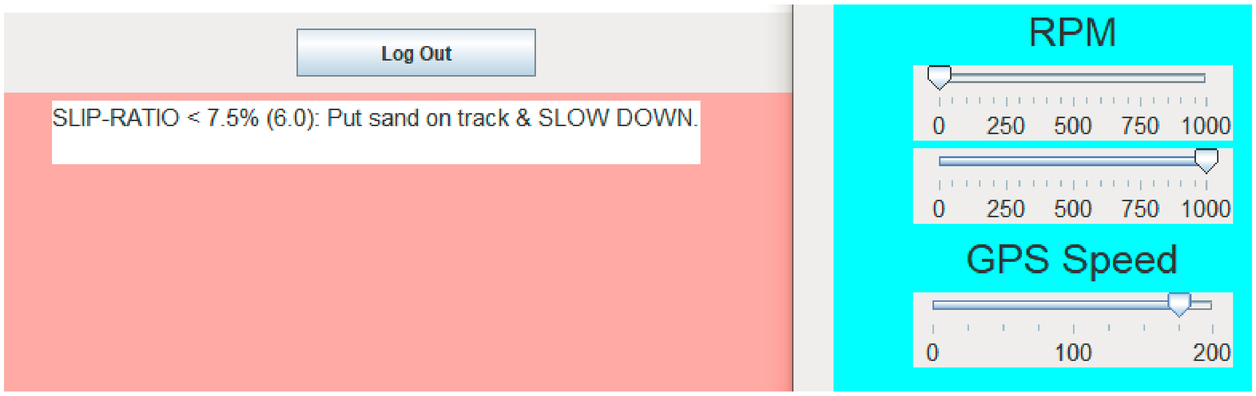
How to test:

- The IoT System will generate a string based on the slip ratio and the recommended action for that slip ratio.
  - If the slip ratio is <10% , notify the conductor to put sand on tracks.
    - Set first RPM slider to 0, and set second RPM slider to 1000.
    - Set the GPS Speed to 150
    - **Result**:



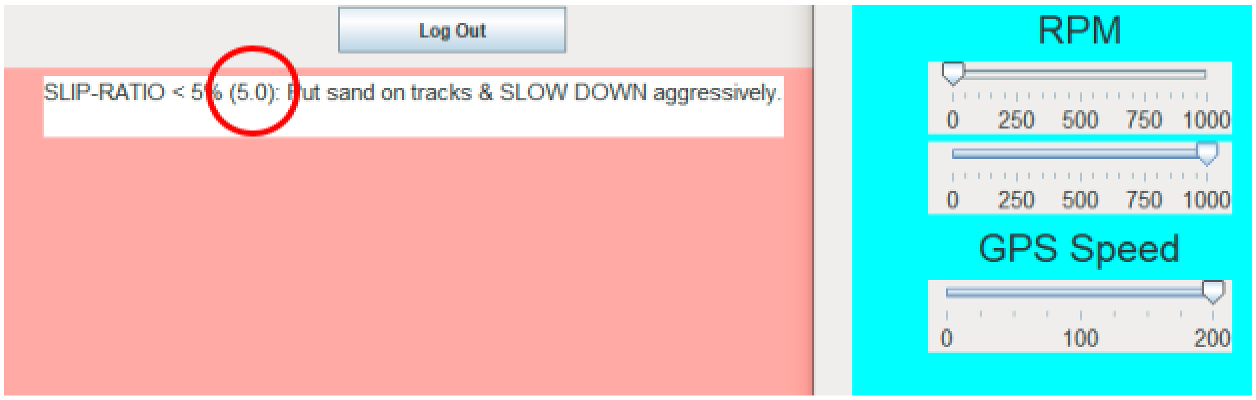
SLIP-RATIO < 10% (8.0) : Put sand on tracks.

- If the slip ratio is <7.5%, notify the conductor to put sand on tracks and slow down.
  - Set first RPM slider to 0, and set second RPM slider to 1000.
  - Set the GPS Speed to 175
  - **Result**:



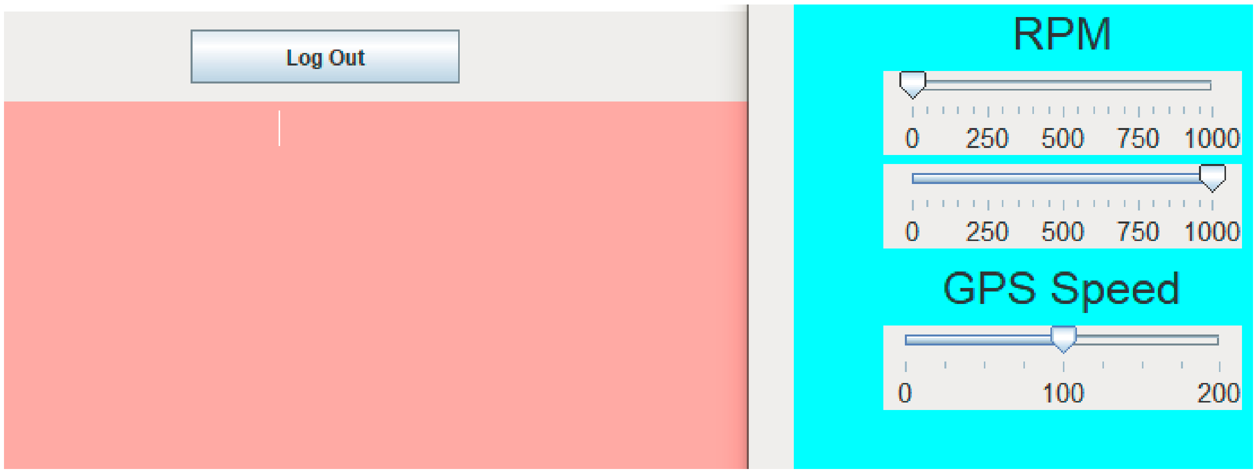
SLIP-RATIO < 7.5% (6.0): Put sand on track & SLOW DOWN.

- If the slip ratio is <5%, notify the conductor to put sand on tracks and slow down aggressively.
  - Set first RPM slider to 0, and set second RPM slider to 1000.
  - Set the GPS Speed to 200
  - **Result**:



SLIP-RATIO < 5% (5.0): Put sand on tracks & SLOW DOWN aggressively.

- If the slip ratio is >10%, no warning is shown
    - If RPM average is 500 (as it is in the previous test cases), setting the GPS speed to anything below 125 will result in no Slip-Ratio Warning being displayed
    - **Result:**



Test 6

Test 6:

➔ Description:
- ◆ *IoT System calculates Time until collision when dangerous object on track is detected*

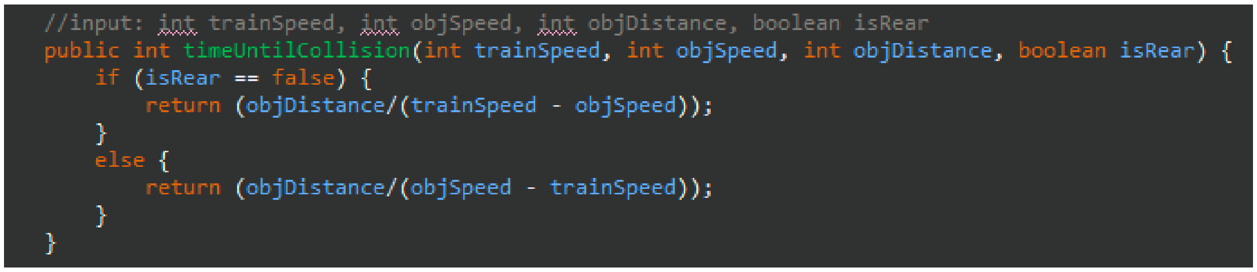
➔ Expected Result:
- ◆ *IoT System produces a number that represents Time Until Collision between object on tracks and train*

a. Radar Sensor sends a frame to ODS.
b. ODS detects an object larger than 1.5 ft. tall.
c. ODS sends the distance and speed of the object to the TSNR.
d. The IoT System pulls ODS data from the TSNR.

e. The IoT System calculates Time Until Collision using Speed of the locomotive, distance to the object, and speed of the object
f. Calculated TimeUntilCollision is used in generating warnings

Test 6 Result

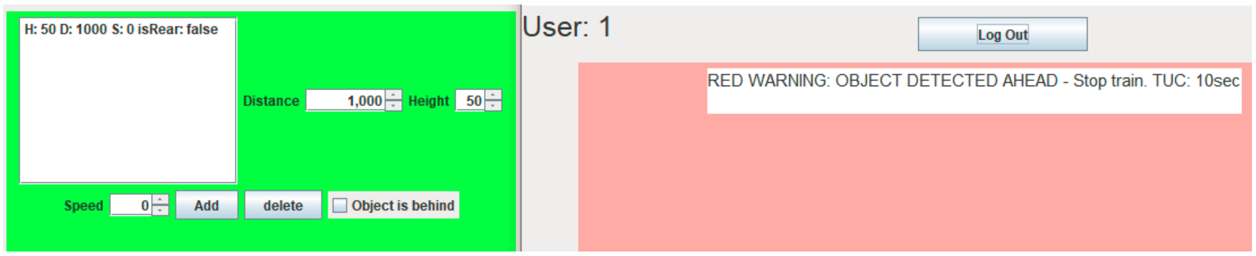
How to test:



```
//input: int trainSpeed, int objSpeed, int objDistance, boolean isRear
public int timeUntilCollision(int trainSpeed, int objSpeed, int objDistance, boolean isRear) {
    if (isRear == false) {
        return (objDistance/(trainSpeed - objSpeed));
    }
    else {
        return (objDistance/(objSpeed - trainSpeed));
    }
}
```

timeUntilCollision() function shown above

- During a “Red Warning”, when an object in front of the train is less than 200 seconds from collision, the Time Until Collision (or TUC) is displayed within the warning message
    - To test this case, create an new Object with the parameters
        - D (distance) : 1000
        - H (height): Anything value above 1.5
        - S (speed): 0
        - IsRear: leave unchecked
    - And set the GPS Speed Slider to 100 m/s (default value)
- **Result**: Warning message should indicate that the Time Until Collision is 10 seconds



H: 50 D: 1000 S: 0 isRear: false

Distance 1,000 Height 50

Speed 0

User: 1

RED WARNING: OBJECT DETECTED AHEAD - Stop train. TUC: 10sec

Correct TUC (Time Until Collision) displayed at the end of warning message

Test 7

Test 7:

➔ Description:
    ◆ *IoT System processes data from TSNR and outputs appropriate warnings for on track Objects Detected*

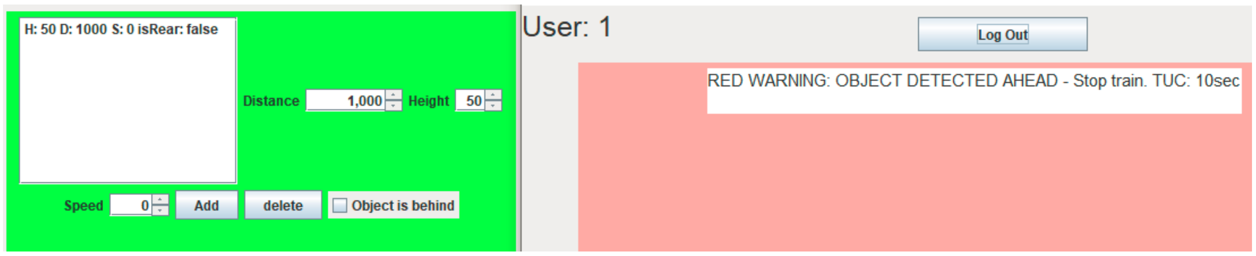
➔ Expected Result:
    ◆ *IoT System produces a string that is presented on IoT Display*

a. The IoT System calculates the Time Until Collision for an object on the track
b. IoT System will display information about the object to the Conductor and will give appropriate warning
   i. If the Time Until Collision is less than 200 seconds, the object is bigger than 1.5 feet tall, and the object is in front, the IoT will display "RED WARNING: OBJECT DETECTED AHEAD - Stop train."
   ii. If the Time Until Collision is less than 260 seconds but greater than or equal to 200 seconds, the object is in front, and the object is bigger than 1.5 feet tall, the IoT will display "ORANGE WARNING: OBJECT DETECTED AHEAD - Apply break to half-speed."
   iii. If the Time Until Collision is less than 260 seconds, the object is behind, and the object is bigger than 1.5 feet tall, the IoT will display "YELLOW WARNING: OBJECT DETECTED BEHIND - Match speed of rear object - obj.getSpeed()." where obj.getSpeed() is a call to the given objects speed parameter value.
c. The string is added to a single string, which is then displayed on the IoT Display

Test 7 Result

How to test:

- IoT System will display information about the object to the Conductor and will give appropriate warning
  - If the Time Until Collision is less than 200 seconds, the object is bigger than 1.5 feet tall, and the object is in front
    - **Result**: IoT Displays "RED WARNING: OBJECT DETECTED AHEAD - Stop train. TUC: *insert time until collision*"
    - To test this case, create an new Object with the parameters
      - D (distance) : 1000
      - H (height): Anything value above 1.5
      - S (speed): 0
      - IsRear: leave unchecked
    - And set the GPS Speed Slider to 100 m/s (default value)



H: 50 D: 1000 S: 0 isRear: false

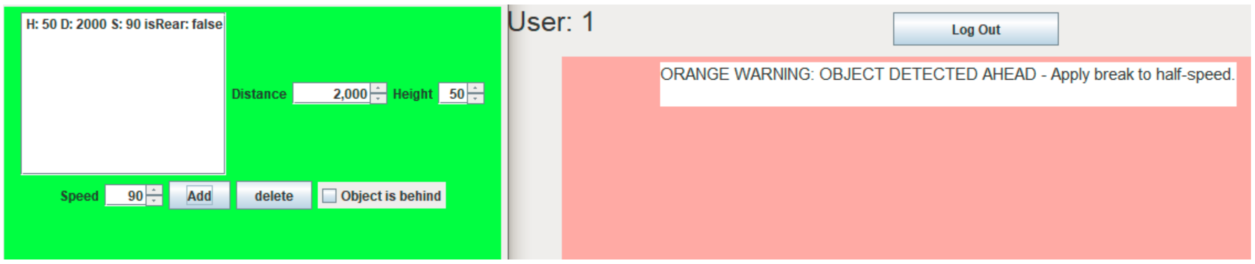
Distance 1,000 Height 50

Speed 0

RED WARNING: OBJECT DETECTED AHEAD - Stop train. TUC: 10sec

  - If the Time Until Collision is less than 260 seconds but greater than or equal to 200 seconds, the object is in front, and the object is bigger than 1.5 feet tall

- **Result**: IoT displays "ORANGE WARNING: OBJECT DETECTED AHEAD - Apply break to half-speed."
- To test this case, create an new Object with the parameters
    - D (distance) : 2000
    - H (height): Anything value above 1.5
    - S (speed): 90
    - IsRear: leave unchecked
- And set the GPS Speed Slider to 100 m/s (default value)


H: 50 D: 2000 S: 90 isRear: false

Distance 2,000 Height 50

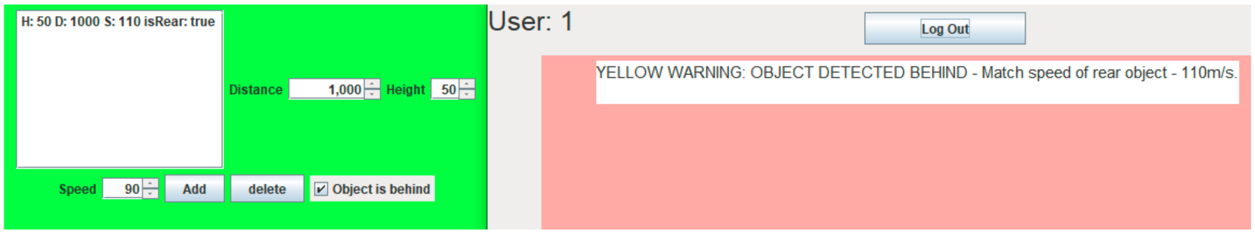
Speed 90

☐ Object is behind

User: 1

ORANGE WARNING: OBJECT DETECTED AHEAD - Apply break to half-speed.

- If the Time Until Collision is less than 260 seconds, the object is behind the train, and the object is bigger than 1.5 feet tall, and the object is going faster approaching the train (going faster than train),
    - **Result**: IoT displays "YELLOW WARNING: OBJECT DETECTED BEHIND - Match speed of rear object - obj.getSpeed()." where obj.getSpeed() is a call to the given objects speed parameter value.
    - To test this case, create an new Object with the parameters
        - D (distance) : 2000
        - H (height): Anything value above 1.5
        - S (speed): 90
        - IsRear: leave unchecked
    - And set the GPS Speed Slider to 100 m/s (default value)


H: 50 D: 1000 S: 110 isRear: true

Distance 1,000 Height 50

Speed 90

☑ Object is behind

User: 1

YELLOW WARNING: OBJECT DETECTED BEHIND - Match speed of rear object - 110m/s.

Test 8

Test 8:

➔ Description:
    - ◆ *Status of Gate objects and on-track Object objects is gathered*

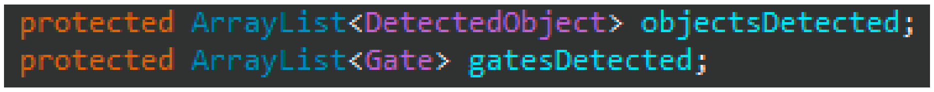
- ➔ Expected Result:
  - ◆ *Gate object and on-track Object object information is accessible for calculation*

1. Gate
   a. Radar sends a frame to TSNR.
   b. ODS pulls the frame from TSNR.
   c. ODS detects that there is a gate in front of the train.
   d. ODS determines if the gate is open or closed.
   e. ODS determines distance from train to gate
   f. Gate information is passed from TSNR to IoT System where calculations are made
2. Object
   a. Radar sends a frame to TSNR.
   b. ODS pulls the frame from TSNR.
   c. ODS detects that there is an object in the path of train.
   d. ODS determines distance from train to object, object height, object speed, and object location relative to train (front, back)
   e. Gate information is passed from TSNR to IoT System where calculations are made

Test 8 Result

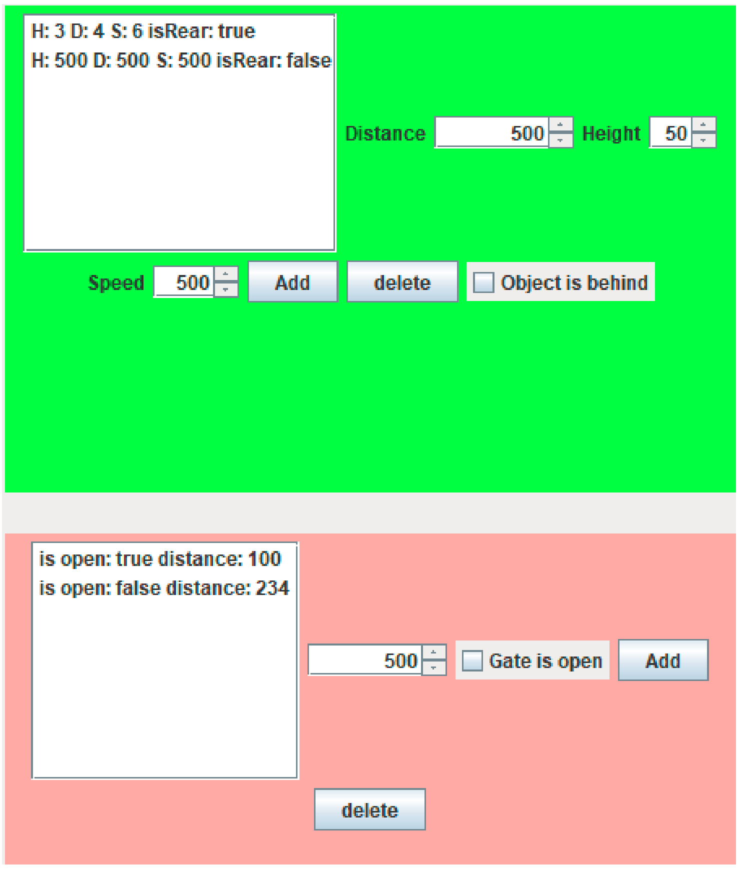
How to test:

- All of our detected Objects and Gates are stored in their own respective arrays, shown below



```
protected ArrayList<DetectedObject> objectsDetected;
protected ArrayList<Gate> gatesDetected;
```

- We can confirm that the status of our Gate objects and on-track Object objects are gathered and stored because the stored Objects and Gates are displayed within the Controller panel (slider panel), and calculations (warning messages) are created using the values they store.
- When testing, any Object or Gate can be deleted or added to their respective arrays



```
H: 3 D: 4 S: 6 isRear: true
H: 500 D: 500 S: 500 isRear: false
```

```
is open: true distance: 100
is open: false distance: 234
```

Test 9

Test 9:

- ➔ Description:
  - ◆ *IoT System processes data from TSNR and outputs appropriate warnings for a detected gate*
- ➔ Expected Result:
  - ◆ *IoT System produces a string that is presented on IoT Display*

a. Gate information is pulled from TSNR

b. IoT System determines necessary warning

   i. If the gate is open, ODS warns the Conductor that the locomotive is approaching a gate that is open and recommends him an action.

      1. Display warning “OPEN GATE DETECTED: Please use horn and apply breaks.”

   ii. If the gate is closed, ODS warns the Conductor that the locomotive is approaching a gate that is closed and recommends him an action.

      1. When the train is less than a mile away from the gate, display the warning “HONK HORN FOR 15 SECONDS: Closed gate is 1 mile ahead.”

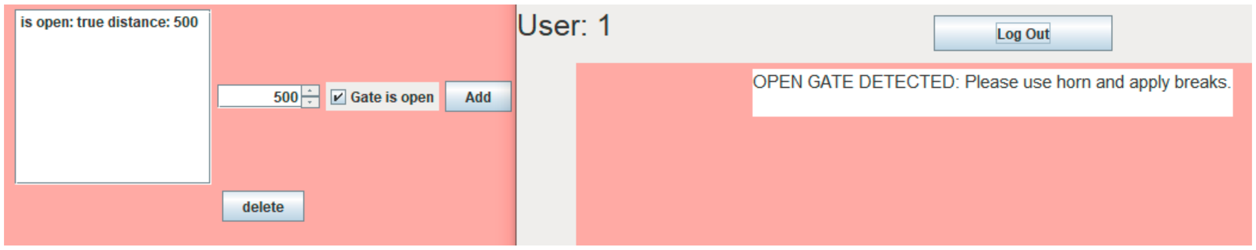
2. When the train is approaching the gate (less than or equal to 400 meters away from gate), display the warning "HONK HORN FOR 6 SECONDS: Passing through closed gate."

c. The string is added to a single string, which is then displayed on the IoT Display

Test 9 Result

How to test:

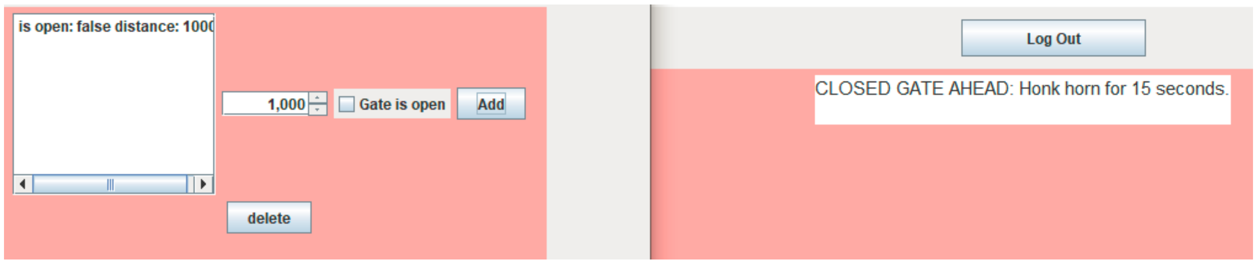
- If the gate is open,
    - **Result:** Display warning "OPEN GATE DETECTED: Please use horn and apply breaks."



```
is open: true distance: 500
```

OPEN GATE DETECTED: Please use horn and apply breaks.

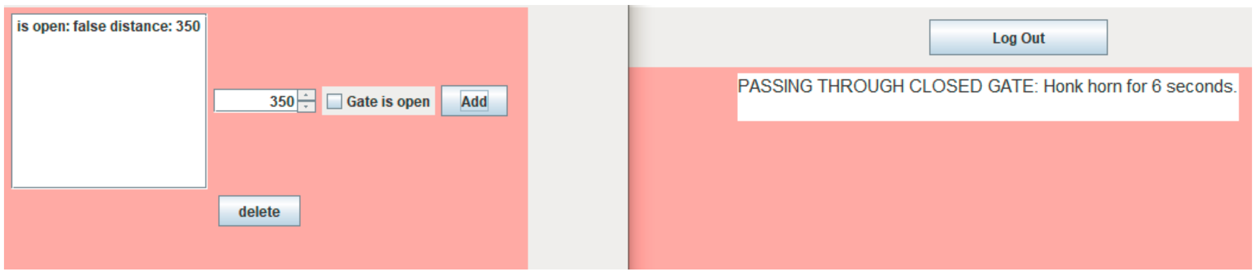
- If the gate is closed
    - When the train is less than a mile (~1650 meters) but greater than half a mile away (~800 meters) from the gate and gate is closed,
    - **Result:** IoT Display shows warning "HONK HORN FOR 15 SECONDS: Closed gate is 1 mile ahead."



```
is open: false distance: 1000
```

CLOSED GATE AHEAD: Honk horn for 15 seconds.

- When the train is approaching the gate (less than or equal to 400 meters away from gate) and gate is closed,
- **Result:** IoT Display shows warning "HONK HORN FOR 6 SECONDS: Passing through closed gate."



is open: false distance: 350

PASSING THROUGH CLOSED GATE: Honk horn for 6 seconds.

Test 10

Test 10:

- ➔ Description:
  - ◆ *IoT System processes data from TSNR and outputs appropriate warnings for predicted bad weather conditions.*
- ➔ Expected Result:
  - ◆ *IoT System produces a string that is presented on IoT Display*

a. The two Weather Sensors send an air pressure reading to the TSNR

b. The IoT System pulls the air pressure readings from the TSNR.

c. The IoT System calculates the average in Hg (Inches of Mercury).

d. The IoT System determines the appropriate string to return if necessary.

   i. If average Hg is 29.8 or below, warn the Conductor that rain storm / snow is incoming.

   ii. If average Hg is above 29.8 and below 30.2; Send that rain is incoming

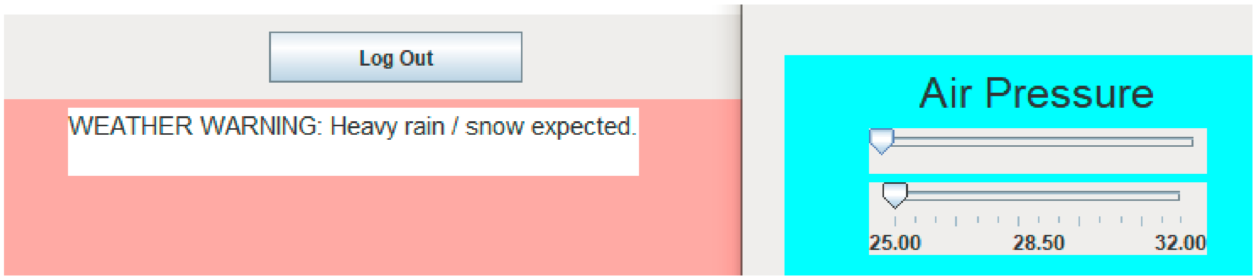
   iii. If average Hg is above 30.2 , Nothing is displayed as a warning display.

e. The string is added to a single string, which is then displayed on the IoT Display.

Test 10 Result

How to test:

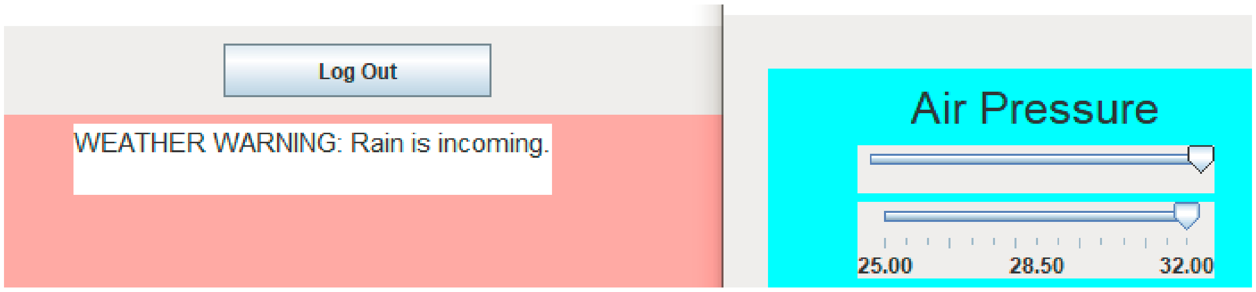
- If average Hg is 29.8 or below,
- **Result:** IoT Display warns the Conductor that rain storm / snow is incoming.



WEATHER WARNING: Heavy rain / snow expected.

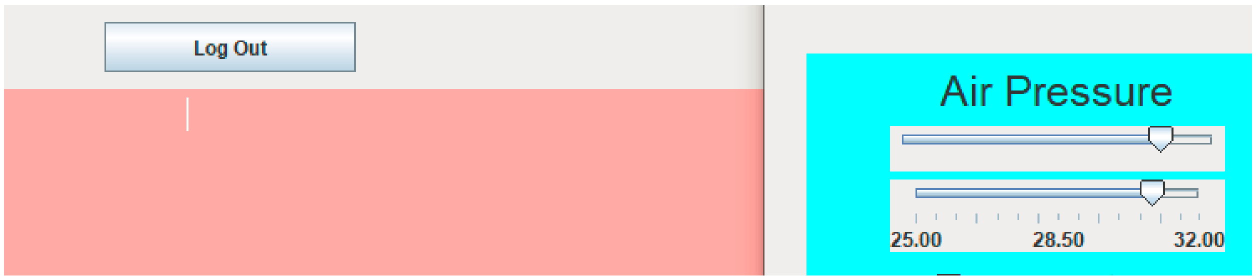
- If average Hg is above or equal to 29.8 and less than 30.2 ,

- **Result:** oT Display warns that rain is incoming



WEATHER WARNING: Rain is incoming.

- If average Hg is above 30.2,
- **Result:** Nothing is displayed as a warning display.



Test 11

Test 11:

➔ Description:
   - ◆ *Temperature surrounding the train is being displayed*

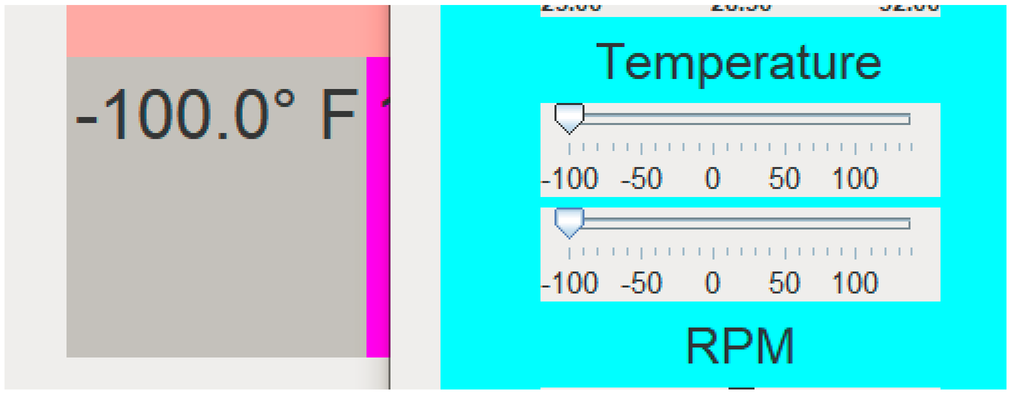
➔ Expected Result:
   - ◆ *The average of the temperature readings is calculated and displayed*

a. TSNR receives readings from sensors
b. IoT System receives readings from TSNR
c. IoT System calculates the average temperature
d. IoT System displays the average temperature

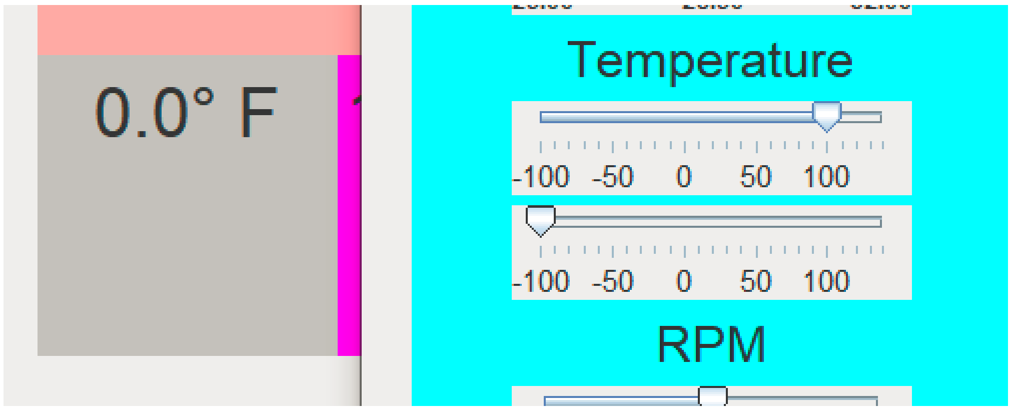
Test 11 Result

How to test:

- We can test by modifying the two “Temperature” sliders in the controller panel, and observing if the Temperature displayed in the User’s view is an average of the two values
   - Data Flow:
      - TSNR receives readings from sensors
      - IoT System receives readings from TSNR
      - IoT System calculates the average temperature
      - IoT System displays the average temperature



Temperature displayed in bottom left corner of User View is constantly updated to match average value of Temperature sliders



With the given temperature values of "-100" and "100", IoT System calculates the correct average to be 0 degrees Fahrenheit and displays value on IoT display

Test 12

Test 12:

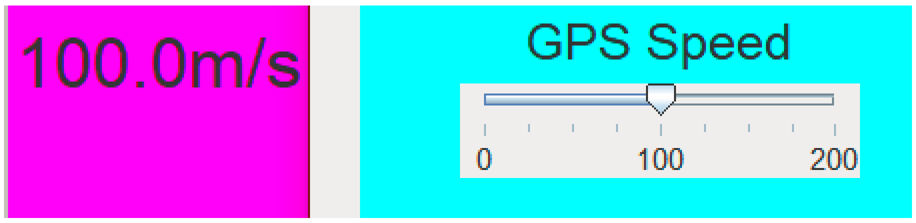
- ➔ Description:
  - ◆ *Speed of the train is being displayed*
- ➔ Expected Result:
  - ◆ *The current speed of the train is consistently updated*

a. The GPS Sensor sends speed value to TSNR
b. TSNR sends speed value to IoT System
c. IoT System displays the speed

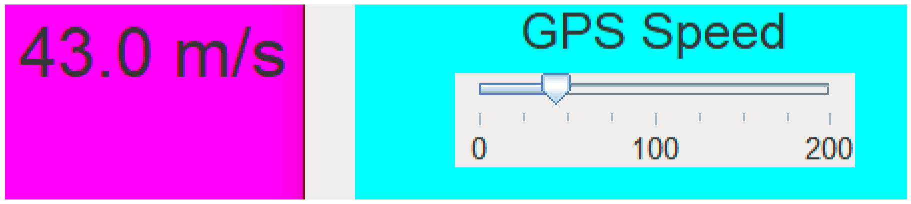
Test 12 Result

How to test:

- Change the GPS Speed slider value within the controller panel, and the change should be reflected on Displayed speed portion of the User View Panel
- **Result:** The current speed of the train is consistently updated



GPS Speed is set to 100 as default value



Displayed GPS Speed changes almost instantaneously for ever slider value change, meaning displayed value is constantly updated

Test 13

Test 13:

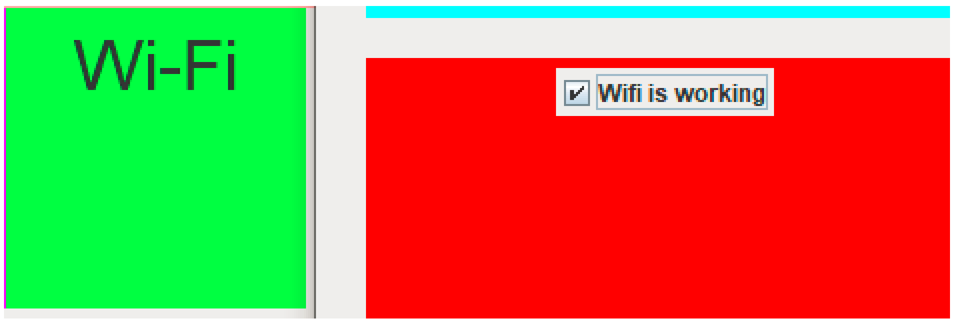
- ➔ Description:
  - ◆ *Display shows whether wifi is working or not*
- ➔ Expected Result:
  - ◆ *Display shows wifi panel as green if wifi is working and red if it is not working*

a. WiFi sensor sends WiFi status to TSNR
b. TSNR sends WiFI status to IoT System
c. IoT System decides what color the WiFi panel on the display should be based on the status

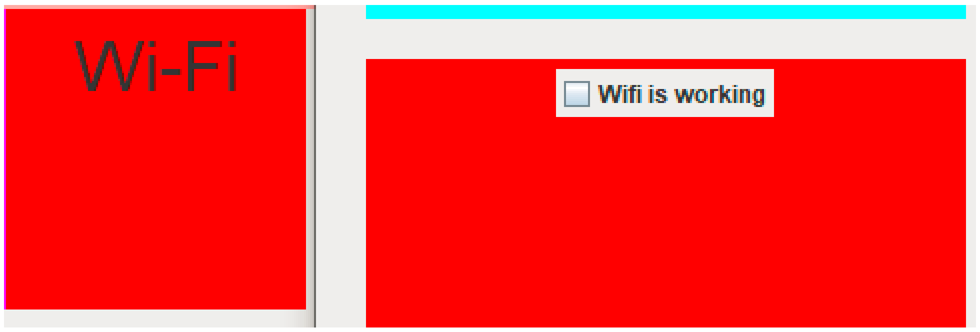
Test 13 Result

How to test:

- Log into system using Operator Credentials
- Click the “Wifi is working” checkbox on the controller panel
  - Check box is filled:
    - **Result:** “Wi-Fi” square in Operator View Page is colored Green



- Check box is NOT filled:
    - **Result:** "Wi-Fi" square in Operator View Page is colored Red



Test 14

Test 14:

➔ Description:
    - *Determine if password is run through a Sha-256 hashing algorithm when validating log in*

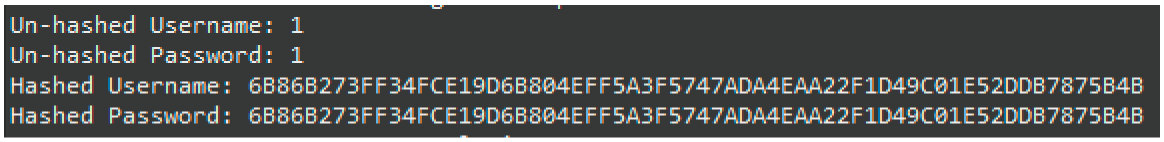
➔ Expected Result:
    - *Password is hashed via Sha-256 algorithm*

a. Case: Password is being input
    i. Password input string is hashed via Sha-256 algorithm
    ii. Compare hashed input to stored value associated with inputted user name
    iii. If they match, success

Test 14 Result

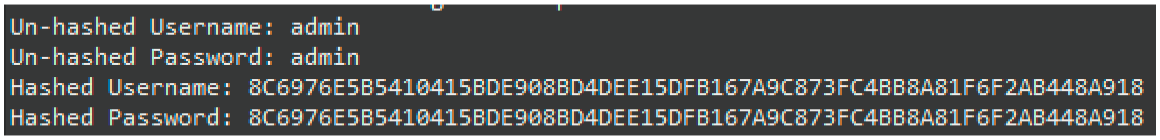
How to test:

- Our program actually encrypts both the username and password of every user when stored.
- For demonstrational purposes, we have outputted both the plaintext values as well as the hashed values of User credentials into the console of our Java application
  Result: Password is hashed via Sha-256 algorithm



```
Un-hashed Username: 1
Un-hashed Password: 1
Hashed Username: 6B86B273FF34FCE19D6B804EFF5A3F5747ADA4EAA22F1D49C01E52DDB7875B4B
Hashed Password: 6B86B273FF34FCE19D6B804EFF5A3F5747ADA4EAA22F1D49C01E52DDB7875B4B
```

Hashed credentials for User account with name and password "1" (output to Java console)



```
Un-hashed Username: admin
Un-hashed Password: admin
Hashed Username: 8C6976E5B5410415BDE908BD4DEE15DFB167A9C873FC4BB8A81F6F2AB448A918
Hashed Password: 8C6976E5B5410415BDE908BD4DEE15DFB167A9C873FC4BB8A81F6F2AB448A918
```

Hashed credentials for Administrator account with name and password "admin" (output to Java console)

- You can check that we do not store plaintext credentials by opening our IDandPasswords.java class
    - Usernames and passwords entered by users are hashed, compared against what we have stored, then stored in local variables (for displaying user name on IoT Display) that are cleared when the User logs out

Test 15

Test 15:

➔ Description:
  ◆ *TSNR is currently holding all raw data from sensors*

➔ Expected Result:
  ◆ *All data fields within TSNR output non-null values via their get method*

a. IoT system calls TSNR get method for a specific data field when executing a calculation method
b. IoT system is able to return an output for method called
c. This is done for all methods within IoT system
    i. For example, If an average value is needed for a function, call calculateAverage(get data from TSNR)

Test 15 Result

How to test:

- Our "iot_system" Java class doesn't hold any of the sensor input data, and instead accesses this data by using "get" functions implemented in the "TSNR" class. If the TSNR wasn't holding the "raw data" (in our programs case, the Controller panel inputs), our program wouldn't work.
    - Flow of "raw data":
        - IoT system calls TSNR get method for a specific data field when executing a calculation method

        - IoT system is able to return an output for method called
            - This is done for all methods within our IoT system. For example, If an average value is needed for a function, call calculateAverage(get data from TSNR)
- It can be assumed that our TSNR is always holding the most recent sensor data values, because our program wouldn't function otherwise.

Test 16

Test 16:

➔ Description:
    ◆ *IoT System is operation within 2 seconds of successful user log-in*

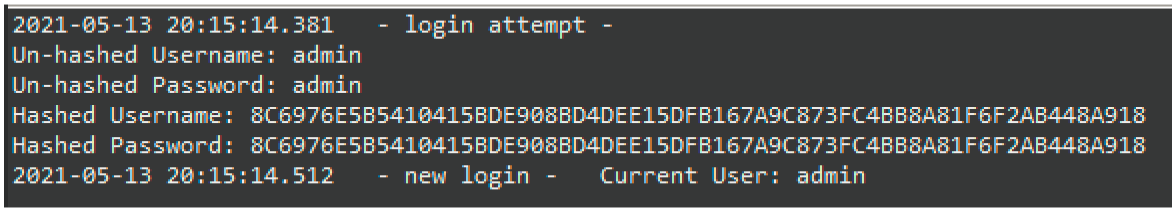
➔ Expected Result:
    ◆ *IoT System Display is turned on, and User view is being displayed*

a. Operator successfully logs in to IoT System
b. By time timer reaches 2 seconds Operator View is displayed

Test 16 Result

How to test:

- For demonstrational purposes, we output the time of a login attempt (time when login button is pressed) as well as the time when our IoT Display opens up the appropriate User View to the Java console.



```
2021-05-13 20:15:14.381   - login attempt -
Un-hashed Username: admin
Un-hashed Password: admin
Hashed Username: 8C6976E5B5410415BDE908BD4DEE15DFB167A9C873FC4BB8A81F6F2AB448A918
Hashed Password: 8C6976E5B5410415BDE908BD4DEE15DFB167A9C873FC4BB8A81F6F2AB448A918
2021-05-13 20:15:14.512   - new login -   Current User: admin
```

First and last lines of Java console output are timestamps of login attempts and successful new logins

- We can see here that
    - The login attempt was at 20:15:14.381
    - The login was successful at 20:15:14.512
- The last three numbers represent milliseconds
    - 512ms - 381ms = 131ms

- **Result:** IoT System displays User view in less than 2 seconds of successful user log in
    - Our program is actually operational in less than 200ms of successful user log-in, far faster than our required 2 seconds

Test 17

Test 17:

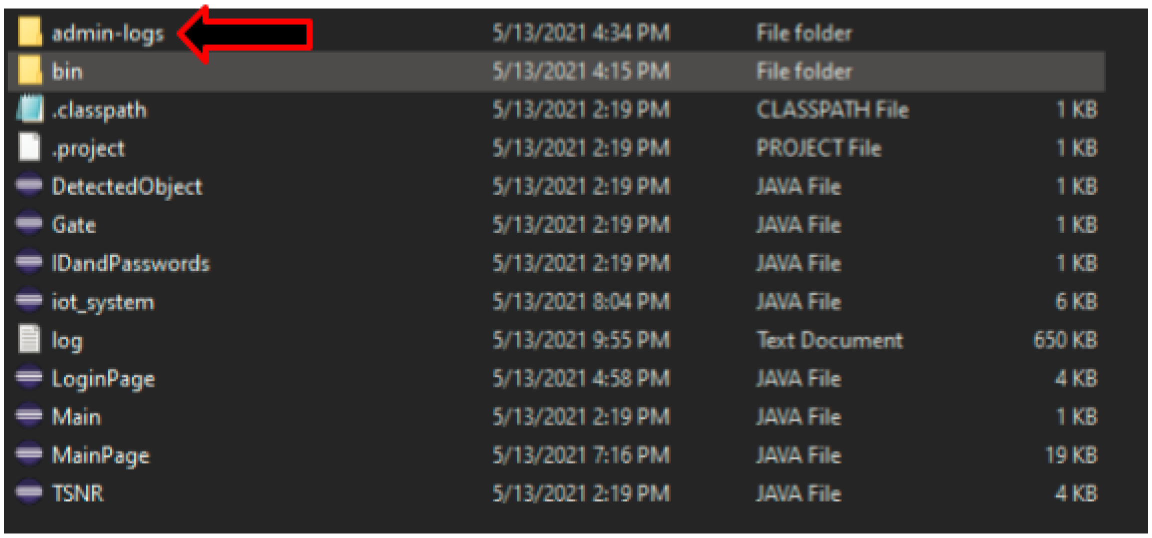
- ➔ Description:
  - ◆ *Administrator presses the "download log" button*
- ➔ Expected Result:
  - ◆ *Current log file is copied into new timestamped file and new empty log file is created*

a. Administrator logs in to IoT System
b. Administrator requests current log by pressing "download log" button in top right corner of administrator view
c. Current log file (log.txt) is copied into an admin log file located in the "admin-logs" folder within the iot program, distinguished by a timestamped filename
d. Current log file is deleted and replaced by new, empty log.txt file

Test 17 Result

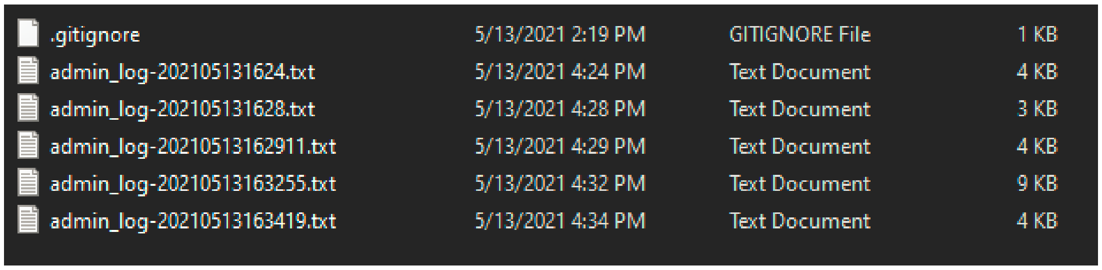
How to test:

- Log in as an administrator using the username "admin" and the password "admin"
- Press the "download log" button in the top right corner
- The log file is now copied into a folder called "admin-logs" within the top level of the iot file folder



```
admin-logs        5/13/2021 4:34 PM   File folder
bin               5/13/2021 4:15 PM   File folder
.classpath        5/13/2021 2:19 PM   CLASSPATH File      1 KB
.project          5/13/2021 2:19 PM   PROJECT File        1 KB
DetectedObject    5/13/2021 2:19 PM   JAVA File           1 KB
Gate              5/13/2021 2:19 PM   JAVA File           1 KB
IDandPasswords    5/13/2021 2:19 PM   JAVA File           1 KB
iot_system        5/13/2021 8:04 PM   JAVA File           6 KB
log               5/13/2021 9:55 PM   Text Document     650 KB
LoginPage         5/13/2021 4:58 PM   JAVA File           4 KB
Main              5/13/2021 2:19 PM   JAVA File           1 KB
MainPage          5/13/2021 7:16 PM   JAVA File          19 KB
TSNR              5/13/2021 2:19 PM   JAVA File           4 KB
```

- Within the admin-logs folder the Admin can find the log they requested
- Admin logs are time stamped with format "year|month|day|hour|minute|second"



```
.gitignore                         5/13/2021 2:19 PM    GITIGNORE File    1 KB
admin_log-202105131624.txt         5/13/2021 4:24 PM    Text Document     4 KB
admin_log-202105131628.txt         5/13/2021 4:28 PM    Text Document     3 KB
admin_log-20210513162911.txt       5/13/2021 4:29 PM    Text Document     4 KB
admin_log-20210513163255.txt       5/13/2021 4:32 PM    Text Document     9 KB
admin_log-20210513163419.txt       5/13/2021 4:34 PM    Text Document     4 KB
```

External Sources

1. https://www.britannica.com/technology/railroad/Railroad-operations-and-control
2. https://www.baumhedlundlaw.com/blog/2016/september/the-most-common-train-accident-causes/
3. https://www.scientificamerican.com/article/broken-rails-are-leading-cause-of-train-derailments/
4. https://railroads.dot.gov/research-development/program-areas/program-areas
5. https://www.cyient.com/blog/rail-transportation/five-smart-ways-how-iot-is-transforming-the-railways
6. https://www.ibm.com/blogs/internet-of-things/connected-trains-rail-travel/
7. https://en.wikipedia.org/wiki/List_of_sensors
8. https://uwnthesis.wordpress.com/2020/07/01/brute-force-password-how-long-will-it-take-to-brute-force-a-password/#:~:text=Brute%20force%20attacks%2C%20will%20normally,close%20to%20a%20months%20reset).
9. https://simple.wikipedia.org/wiki/RSA_algorithm#:~:text=RSA%20
10. https://learn.sparkfun.com/tutorials/mpl3115a2-pressure-sensor-hookup-guide/all
11. https://www.railfreight.com/intermodal/2020/09/09/when-average-speed-dips-below-40km-h-railways-should-be-free/?gdpr=deny
12. https://www.minnesotasafetycouncil.org/ol/stop.cfm#:~:text=The%20average%20freight%20train%20is,about%20a%20mile%20to%20stop
13. https://sciencing.com/predict-weather-barometer-5767204.html
14. https://medium.datadriveninvestor.com/measuring-traffic-speed-with-deep-learning-object-detection-efc0bb9a3c57
15. https://www.tesla.com/autopilotAI
16. https://www.nwengineeringllc.com/article/computer-vision-in-embedded-systems-and-ai-platforms.php