Requirements Document

for

Hug the Rail IoT

Version 7.2

Prepared by Tristan Kensigner, Grace Mattern, Zachary Schuh, &
Christian Szablewski-Paz

Stevens Institute of Technology

May 3, 2021

Table of Contents

Revision History

Name(s)	Date	Reason For Changes	Version
Christian	2/15	Create document/begin section 1: Intro	1.0
Grace	2/16	Revise section 1: Intro	1.0
Zac, Christian, Trista	2/16	Organize section 1: Into paragraphs, add notes	1.0
Grace, Christian, Zac Tristan	2/22	Add section 2: Overview	1.0
Christian	2/23	Rewrite bullet points into paragraphs	1.0
Tristan	2/23	Add section 9: Cost	1.0
Grace	3/1	Revise/organize entire document, add graphics	1.0
Tristan	3/1	Elaborate section 2.4, add graphics	1.0
Grace and Christian	3/2	Add section 3: Requirements	1.1
Grace	3/4	Revise section 3: Requirements	1.1
Grace, Tristan, Christian	3/9	Update section 3: Requirements with CTO of HTR recommended features	1.1
Grace	3/11	Update timeline, slippage system diagram, section 3: Requirements	2.0
Grace	3/16	Incorporate TA 1.0 notes, add section 4: Requirements Modeling, fix formatting	2.1
Christian, Grace, Tristan	3/17	Add use cases in section 4: Requirements Modeling	2.1
Christian	3/20	Work on model index cards	2.2
Christian, Tristan, Grace, Zac	3/21	Finish section 4: Requirements Modeling	3.0

Grace, Christian, Tristan, Zac	3/28	Make changes as directed to section 4: Requirements Modeling	4.0
Tristan, Christian, Grace	4/5	Add section 5: Software Architecture	5.0
Grace, Christian, Tristan	4/10	Revise section 5: Software Architecture	6.0
Christian, Zac	4/18	Add section 6: Code	7.0
Christian, Grace, Tristan	4/22	Add GUI, simulations, and user input testing to section 6: Code	7.1
Grace	4/29	Fix requirements, remove section 8 and 9	7.2
Christain, Tristan, Grace, Zach	5/2	Update with horn feature, complete section 6 and 7	7.2

1. Introduction

1.1 Problem

Trains are a very common and popular form of transportation that require analysis of data derived from the weather conditions, fuel consumption, and time prediction, among others, in order to ensure safe and successful travel. Current trains retrieve data from servers to receive updates that are needed to make calculations for the Locomotive Control System (LCS). One major setback to this system is that trains need to be connected to servers to keep receiving updates and consequently provide safe travel. This leaves the trains vulnerable when a cellular wifi or internet connection is lost. No new data can be retrieved without this connection and thus the safety of the passengers and operator are jeopardized.

1.2 Purpose

The Internet of things (IoT) Hug the Rail (HTR) project is a system that will allow the LCS to make decisions locally in absence of cellular and wifi connectivity to back offices. For example, it will provide the operator with information regarding if the train is slipping or if there is an obstruction. Furthermore, the operator will be able to make informed decisions regarding the problems and cautions specified by the IoT HTR thereby maintaining safe operation and avoiding potentially dangerous situations. The overall purpose of this product would be to make HTR safer, less costly, and more efficient.

1.3 Team

Our dedicated team of developers are skilled in areas such as problem-solving, programming, and project management and are excited to participate in the project. Despite having little experience with embedded systems, GitLab, and IoT systems, we are driven to design a system that allows trains to efficiently operate without a dependence on a connection. Christian, Zach, and Tristan's main roles will be coders and problem solvers. Coders' responsibilities include coding, debugging and testing. Grace's role will be as a project manager due to her being highly organized and detail-oriented. The project manager's responsibilities include creating a schedule and timeline to execute each phase of the project. We hope to innovate and reimage the future of the locomotive industry. Our team will develop and implement IoT Hug the Rail from February 9th to May 11th of 2021.

1.4 Evolving Current Operations

Current operations on trains receive data from the departure station and cloud. During the travel, updated data is sent from servers in range and the operator utilizes the information to control the LCS. Once at the destination, the train, station, and cloud exchange data. The cloud communicates this information over the larger network. While the LCS has evolved over the years as it now can supply lots of information such as heat of the engine, fuel consumption, and generally offers a failsafe-like "software" to prevent the engine from operating outside of predefined safe limits.

1.5 Approach

The edge devices will capture data from other HTR locomotives and the environment instead of relying on data from servers. Many sensors will be implemented to provide crucial information that has now become unavailable (due to the lack of connection). Data from sensors will be sent to a time sensitive networking router (TSNR) which will act as a bridge between the raw sensor data and IoT HTR. IoT HTR will make any necessary calculations and display data from the sensors. IoT HTR's main goal is to make suggestions via the Display to the operator. The product will provide the capability for the operators to receive statuses and download the latest rules for operation from the Fog/Cloud onto the IoT software. Consequently, a finite state system will be created, and such features will help make the appropriate suggestions to the operator.

1.6 Timeline*

The timeline of this project will closely follow an agile methodology; throughout the development process changes will likely be made to both requirements and capabilities. Within each iteration of development, the team will work to recognize problems early and adapt to any changes while continuing to work efficiently. These include but are not limited to software design, technical requirements, deliverables, and the timeline itself. We find that this model helps provide a high-quality project that is attuned to the needs and desires of the consumers and stakeholders while also reducing the failure to analyze risks.

Week 1: 2/2 - **Communication phase** (constantly revisited, understand new desires of stakeholders)
Form teams
Decide roles and responsibilities
Week 2: 2/9 - **Planning phase** (revisited to address issues)
Understand the problem
Create documentation template
Start Section 1: Intro
Research existing systems
Week 3: 2/16 - Review/revise Section 1: Intro
Start Section 2: Overview
Research sensors that can be used to provide information to IoT HTR
Begin Section 10: Cost
Week 4: 2/23 - Familiarize team with GitLab/Git
Upload version 1.0 on GitLab
Research wheel sensors
Research front sensors
Research GPS interaction with wheel sensor
Research thermometer interaction with wheel sensor
Understand how sensors capture data from other HTR trains and surrounding environment
Week 5: 3/2 - **Requirement Analysis** (revisited if old model fails to fit new requirements)
Start Section 3: Requirements
Implement sensors that enable the IoT HTR to make suggestions
Add CTO of HTR features to Section 3
Research gate status sensors
Research hardware and OS for IoT HTR
Week 6: 3/9 - Continue Section 3
Use analytics engine to process info from the sensors to the LCS
Week 7: 3/16 - **Requirements Modeling**
Start Section 4: Use Cases
Enable operator to download the latest data from Fog/Cloud onto the IoT HTR when connected
Week 8: 3/23 - Continue Section 4
Week 9: 3/30 - Continue Section 4
Week 10: 4/1 - **Software Architecture**

Start Section 5
Week 11: 4/6 - Continue Section 5
Week 12: 4/13 - **Coding and Testing**
Start Section 6
Start Section 7
Week 13: 4/20 - Continue coding
Continue testing
Copy code to Section 6: Code
Complete Section 7: Testing
Week 14: 4/27 - Continue coding
Continue testing
Week 15: 5/4 - Continue coding
Continue testing
Week 16: 5/11- **First Iteration Completed**
Present

*Subject to change

2. Overview

2.1 Problem Statement

Current IoT technologies on trains are focused on live data received from the Central Operation Center Servers via WiFi/Cellular networks. This heavy dependence on WiFi/Cellular connectivity introduces a massive weakness regarding safety when a stable connection is lost. Therefore it is imperative that a reliable IoT system based on local data be developed to allow for the continuation of a safe trip. For a train to operate properly external data must be provided. In the case that wifi is unavailable to provide the necessary data, a backup solution must be put in place.

2.2 Data Required to Operate

IoT HTR will need to collect data regarding objects, their position relative to the train (front or back), and their movement (stationary or moving) . IoT HTR will alert the operator to slow down or stop. The train will also need information about the gate crossing's status (open or closed). If the gates are open, the train will need to slow down or halt because vehicles may be crossing the train crossing. We assume that the train will still have access to the GPS/satellite. The GPS data can coordinate with the wheel sensor data which will provide data on the wheels' spin rate (rpm). In a sense, these two data will act to verify each other since when they contradict each other it should alert the operator that the train is probably slipping.

2.3 Expected Output

The data will be accessible to IoT HTR via the TSNR, from which the operator can read the suggestions from IoT HTR Display and make the appropriate decisions. Specifically, IoT HTR will interface with sensors to collect data, analyze it, and use rule-based logic to issue recommendations or actions to the operator via the IoT HTR Display by using TSNR as a proxy. For example, if the front sensor detects an object is too close, it then will detect if it is moving. This information will be passed to TSNR which will then pass it to IoT HTR. IoT HTR will provide a hazard warning to the operator, who can then slow down or stop the train. Another example is the wheel sensor which will verify its data with the still working GPS. The GPS provides the speed the train is going while the wheel sensor will provide how fast the wheels are rotating (rpm) and thus if one of the data fails to corroborate the other, then a warning will appear on the IoT HTR Display. With the knowledge that the train is slipping from the warning message, the operator can slow down or brake.

2.4 Available Technologies

There are a multitude of different sensors available. A tachometer is the main sensor we intend to use to check for wheel slippage (along with the GPS data). Another important sensor is a radar/sonar/ultrasonic-like device that can map out the environment surrounding the train for the purpose of warning/alerting the operator of obstructions. Popular technologies include LED time of flight sensors and LiDAR sensors, the latter of which are not uncommon on cars and airplanes. A LiDAR sensor continually fires off beams of laser light and then measures how long it takes for the light to return to the sensor. In effect, a LiDAR sensor returns the distance of objects hit by the laser beam and thereby creating a 3D visualization of the environment. With continuous firing, it can also present the movement of objects surrounding the train just as it would notify a driver that a pedestrian, curb, or other vehicle is near. The GPS is a versatile technology that provides information about the position of the train. Not only is it highly reliable, but it also integrates well with the previous technologies.

The sensors for the IoT Hug the Rail Project would be mainly located in three spots. The tachometers will be located near the wheels of the train as these sensors require information from the wheel's rotation and friction. This data will provide the necessary information to IoT HTR about the train's speed, change of speed, and friction force. The front and back of the train will house sensors concerning 3D mapping for obstacle detection. Our IoT system will still have access to the GPS as it does not require a WiFi/cellular connection. The status (open/closed) of cross gates will be handled by object recognition cameras that continuously compare photos taken with a library of reference photos.

2.5 Systems

The IoT HTR system will collect data retrieved from the various sensors and interpret them accordingly. Results (i.e. weather, speed, obstructions, etc.) will be available via the IoT HTR Display and suggest operational changes to the operator based on its detections. Below are some conceptual architectures for our IoT. **Figure 1** and **Figure 2** communicate the general problem and solutions. **Figure 3** shows how IoT HTR would be used specifically regarding weather based slippage.

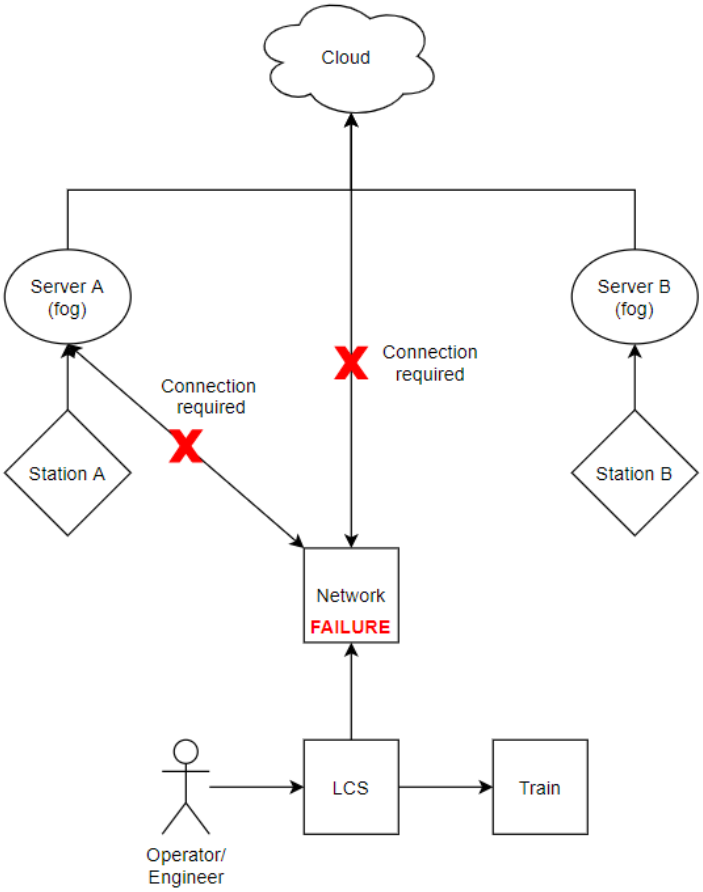


Figure 1: *shows the problems that occur if a connection is lost (represented by the red x's). Supposing the train starts in the range of station A, the LCS is connected to the network which receives data from server A and also the cloud. However when the connection is lost no new data is retrieved from server A or the cloud and the network fails.*

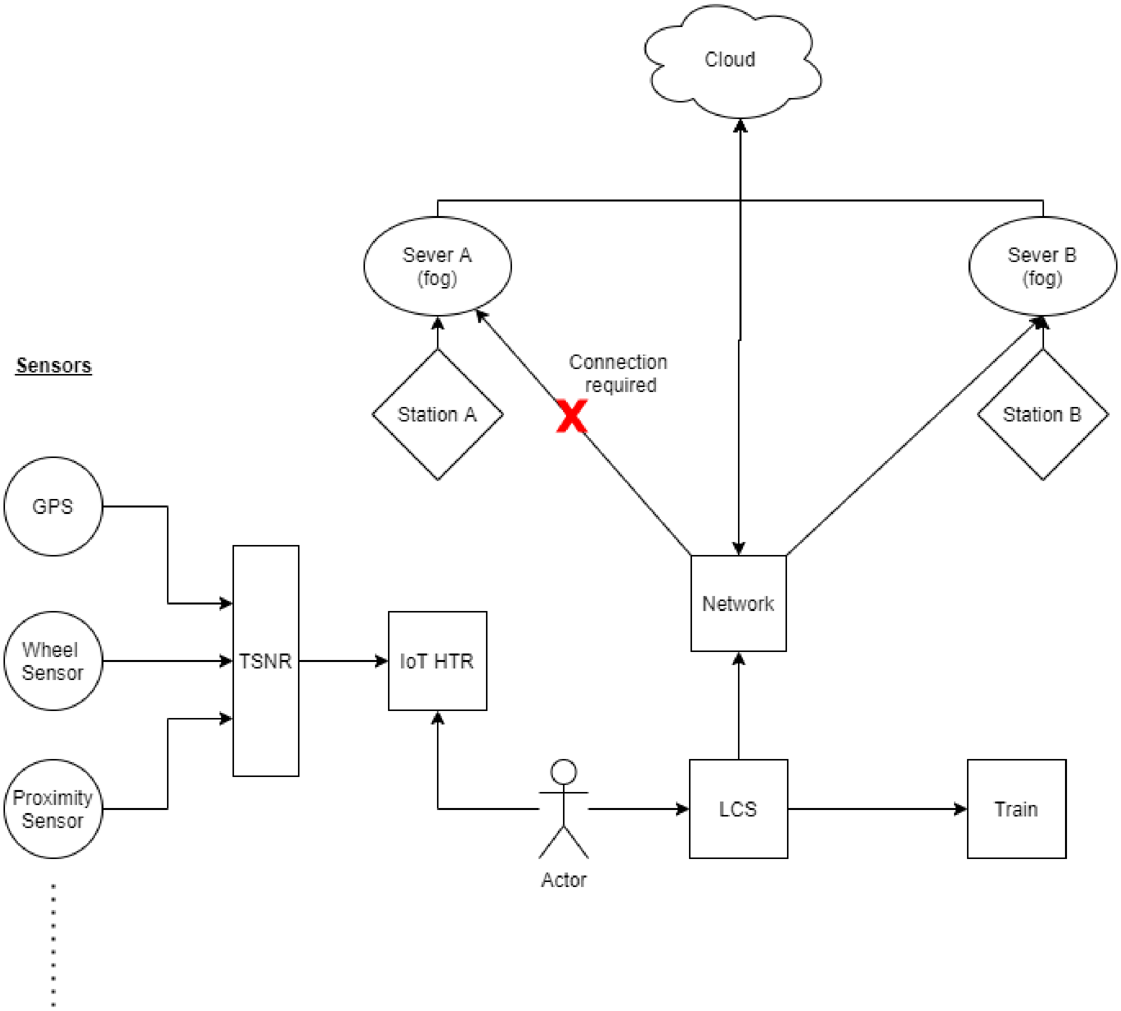


Figure 2: *shows the general solution IoT Hug the Rail Project will provide. Supposing the train is headed to station B without a connection, information from the GPS, proximity sensor, wheel sensor, and other sensors are passed through the TSNR and processed by the IoT HTR. IoT HTR displays the data and any warnings which are read by the operator who can then make the appropriate decisions. Once at station B, a connection is reestablished. The train, server B, and the cloud share data.*

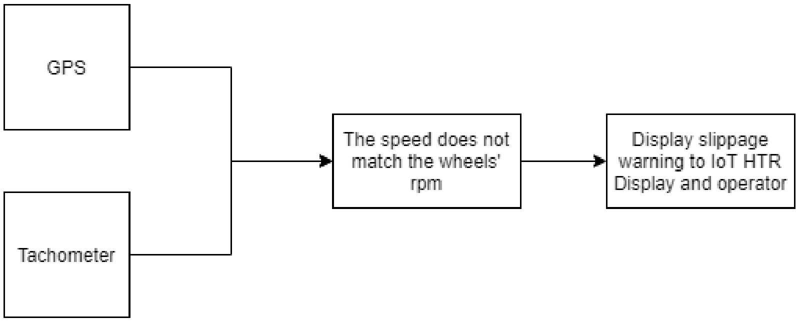

GPS
Tachometer
The speed does not match the wheels' rpm
Display slippage warning to IoT HTR Display and operator

Figure 3: *The GPS could report the speed of the train but the tachometer would contradict the data by showing that the wheels are rotating faster than the given speed. The following data will be available to the operator on the IoT HTR Display.*

3. Requirements

3.1 Non-Functional Requirements

3.1.1 Security

R-1: IoT HTR shall only be accessed via user ID and password.
R-2: Operators shall only see the operations of the train on the IoT HTR Display .
R-3: Only Admins shall have access to raw data and logs.
R-4: Only Admins shall have access to system configurations.
R-5: IoT HTR network shall be secured by a LoRaWan Protocol.
R-6: IoT HTR shall encode all user-supplied output.

3.1.2 Performance

R-7: IoT HTR sensors shall process an event within 0.5 seconds.
R-8: TSNR shall send sensor data to IoT HTR within 0.5 seconds.
R-9: IoT HTR shall process an event within 0.5 seconds.
R-10: IoT HTR shall process 1,000 events per second without delay.
R-11: IoT HTR shall display processed events within 1.0 seconds.
R-12. The Log will log events every 0.01 seconds.

3.1.3 Reliability

R-13: IoT HTR shall operate with no failures 99.99% of the time.
R-14: IoT HTR shall alert the Operator how fast obstructions are moving with 95% accuracy.
R-15: The precision of distance-based calculations shall be at least 0.000001.
R-16: IoT HTR defect rate shall be less than 1 failure per 1000 hours of standard operation.
R-17: IoT HTR shall activate within 100 ms after the power button is pressed.
R-18: IoT HTR shall deactivate within 100 ms after the power button is pressed.
R-19: IoT HTR shall meet or exceed 99.9% uptime.
R-20: The network of sensors shall be operational at least 99.99% of the time.

3.1.4 Other Non-Functional Requirements

R-21: IoT HTR shall be equipped with a display.
R-22: IoT HTR shall be equipped with a power button.
R-23: Operators and admins shall be able to start and stop the IoT HTR via a power button.
R-24: The Operator shall be prompted with a login screen after pressing on the power button.
R-25: The Display shall show recommendations from the IoT HTR.
R-26: The Display will display a continue normal operations message if no issues are presented by all the sensors.
R-27: The Log continuously logs data and sends it to a single log file to keep track of all events.
R-28: IoT HTR shall have a clock to attach timestamps to the log file entries.

3.2 Functional Requirements

3.2.1 Standing Objects

R-29: There shall be two LiDAR sensors, one in the front and one in the back of the train.
R-30: The LiDAR sensors shall continuously take pictures to map the surroundings.
R-31: The LiDAR sensors shall provide the distance and speed of obstructions to the TSNR every 0.05 seconds.
R-32: IoT HTR shall receive obstruction data from the TSNR.
R-33: IoT HTR shall display a warning message if an obstruction is detected via the Display.
R-34: IoT HTR shall display that the object is stationary via the Display.
R-35: IoT HTR shall suggest to the Operator to slow down if there is an object within a 2.0 mile radius of the train via the Display.
R-36: IoT HTR shall suggest to the Operator to brake if there is an object within a 1.0 mile radius of the train via the Display.
R-37: IoT HTR shall display normal operations when there is no object within a 2.0 mile radius of the train via the Display.

3.2.2 Moving Objects

R-38: There shall be two LiDAR sensors, one in the front and one in the back of the train.
R-39: The LiDAR sensors shall continuously take pictures and map the surroundings.
R-40: The LiDAR sensors shall provide the distance and speed of obstructions to the TSNR every 0.05 seconds
R-41: IoT HTR shall receive obstruction data from the TSNR.
R-42: IoT HTR shall display a warning message if an obstruction is detected via the Display.
R-43: IoT HTR shall suggest to the Operator to slow down if there is an object within a 2.0 mile radius of the train via the Display.
R-44: IoT HTR shall suggest to the Operator to brake if there is an object within a 1.0 mile radius of the train via the Display.
R-45: IoT HTR shall display normal operations when there is no object within a 2.0 mile radius of the train via the Display.

3.2.3 Gate Crossings

A closed gate status means cars should stop at the railroad crossing as a train is approaching.
An opened gate means that cars can drive over the railroad crossing as no train is approaching.

R-46: There shall be one radar sensor located in the front of the train.
R-47: The Radar sensors shall continuously emit radio waves and checks the change in frequency after omission.
R-48: The Radar sensors shall provide the distance and status of a gate crossing to the TSNR every 0.05 seconds.
R-49: IoT HTR shall receive gate crossing data from the TSNR.
R-50: IoT HTR shall display a warning message if the gate is open via the Display.
R-51: IoT HTR shall suggest to the Operator to slow down if the gate is open within a 2.0-mile Radius via the Display.
R-52: IoT HTR shall suggest to the Operator to brake if the gate is open within a 1.0-mile radius via the Display.
R-53: IoT HTR shall suggest to the Operator to honk the horn for 15 seconds when it is 1 mile away from the gate via the Display.
R-54: IoT HTR shall suggest to the Operator to honk the horn for 5 seconds when it reaches the gate via the Display.
R-55: IoT HTR shall display normal operations when the gate is closed via the Display.

3.2.4 Wheel Slippage

R-56: IoT HTR is equipped with four tachometers on the four outermost train wheels.
R-57: IoT HTR is equipped with one GPS.
R-58: The Tachometers shall continuously measure the rotation speed of the train wheel in rpm.
R-59: The GPS shall continuously send and receive signals to calculate the speed of the train.
R-60: The Tachometers shall provide the wheels' rpm to the TSNR every 0.05 seconds.
R-61: The GPS shall provide the speed of the HTR train to the TSNR every 0.01 seconds.
R-62: IoT HTR shall receive wheel slippage data from the TSNR.
R-63: IoT HTR shall calculate the speed of the HTR train from the tachometer data.
R-64: IoT HTR shall calculate the difference in the speeds from the GPS and tachometer.
R-65: IoT HTR shall display a warning message if the train's wheels are slipping via the Display.
R-66: IoT HTR shall suggest to the Operator to slow down if the train is minorly slipping (5mph < Δspeed < 10mph) via the Display.
R-67: IoT HTR shall suggest to the Operator to brake if the train is slipping severely (Δspeed > 10mph) via the Display.
R-68: IoT HTR shall display normal operations when slipping stops via the Display.

3.3 Hardware & Operating System

R-69: IoT HTR hardware shall be able to support at least 1,000 sensors.
R-70: IoT HTR shall support 5TB of data every day.
R-71: IoT HTR shall be able to import and export information obtained from operating.
R-72: IoT HTR shall be updated yearly with patches in between.

3.3.1 Operating System

R-73: IoT HTR shall be equipped with Microsoft Windows 10 IoT Enterprise
R-74: IoT HTR shall be equipped with Microsoft Windows 10, 64-bit

3.3.2 Processor

R-75: IoT HTR shall be equipped with ARM processor

3.3.3 RAM

R-76: IoT HTR shall be equipped with 2GB of RAM

3.3.4 Graphics Card

R-77: IoT HTR shall be equipped with a GPU that supports DirectX 9

3.3.5 Storage

R-78: IoT HTR shall be equipped with 32GB of storage

3.3.6 Sensors

R-79: IoT HTR shall be equipped with a Tachometer - measures the rpm of a wheel
R-80: IoT HTR shall be equipped with LiDAR sensors - maps surrounding environment

R-81: IoT HTR shall be equipped with Radar sensors - detect is the status of gate crossing
R-82: IoT HTR shall be equipped with a GPS - give global positioning

4. Requirements Modeling

4.1 Use Cases

Yellow warning messages mean slow down.
Red warning messages mean stop and apply brakes.
If no warnings are detected/displayed, normal operations will be displayed.

Use case 1: Initialize IoT HTR
Primary actor: Operator
Secondary actors: IoT HTR
Goal in context: Turn on IoT HTR
Preconditions: IoT HTR is off
Trigger: Operator decides to initialize IoT HTR, that is, to press the power button
Scenario:

1. Operator presses the power button
2. Display turns on
3. IoT HTR displays the login screen
4. Operator types in ID and password
5. IoT HTR displays operations of the HTR train

Exceptions:

1. Incorrect login: Warning message “Invalid Username or Password” is displayed, Operator re-enters correct ID/password
2. Fails to initialize: Display does not turn on, abort use case. Admin is required to fix the issue(s)

Use case 2: Initialized IoT HTR with admin privileges
Primary actor: Admin
Secondary actors: IoT HTR
Goal in context: Turn on IoT HTR
Preconditions: IoT HTR is off
Trigger: Admin decides to initialize IoT HTR, that is, to press the power button
Scenario:

1. Admin presses the power button
2. Display turns on
3. IoT HTR displays the login screen
4. Admin types in admin ID and password
5. IoT HTR displays system configurations (software os)

Exceptions:

1. Incorrect login: Warning message “Invalid Username or Password” is displayed, Admin re-enters correct ID/password
2. Fails to initialize: Display does not turn on, abort use case. Admin is required to fix the issue(s)

Use case 3: Obstruction detection
Primary actor: Proximity sensors
Secondary actors: IoT HTR

Goal in context: Notify Operator for any encounters of obstructions
Preconditions: IoT HTR is initialized
Trigger: Proximity sensors detect that there is an obstruction
Scenario:

1. LiDAR sensors detect obstructions and their speed that are within 2 miles and reports it to TSNR
2. TSNR sends collected data to IoT HTR.
3. The Display displays if there is an obstruction.
4. The Display displays if it is a moving or stationary obstruction.
5. The Display displays a yellow warning message when the obstruction's distance is between 1 mile and 2 miles (1 mile < distance < 2 miles).
6. The Display displays a red warning message when the obstruction's distance is less than 1 mile (distance < 1 mile).
7. Log records obstructions.

Use case 4: Gate crossing detection
Primary actor: Gate crossing sensors
Secondary actors: IoT HTR
Goal in context: Notify Operator of the gate status
Preconditions: IoT HTR is initialized
Trigger: Gate crossing gate based sensors detect the status of gate crossing
Scenario:

1. Radar sensors detect closed gates that are within 2 miles and reports it to the TSNR
2. TSNR sends collected data to IoT HTR.
3. The Display displays the status (opened/closed) of the gate crossing
4. The Display displays a yellow warning message when the open gate's distance is between 1 miles and 2 miles (1 miles < distance < 2 miles).
5. The Display recommends to the operator to honk the horn for 15 seconds when the train is 1 mile away from the gate (distance = 1 mile).
6. The Display displays a red warning message when the open gate's distance is less than 1 mile (distance < 1 mile).
7. The Display recommends to the operator to honk the horn for 5 seconds when the train is 0 miles away from the gate (distance = 0 miles).
8. Log records gate status.

Use case 5: Wheel slippage detection
Primary actor: Wheel sensors
Secondary actors: IoT HTR
Goal in context: Notify Operator for any encounters of wheel slippage
Preconditions: IoT HTR is initialized.
Trigger: Wheel based sensor detects wheel slippage
Scenario:

1. The tachometer detects the rpm of the HTR train wheels and reports it to TSNR.
2. GPS detects the speed of the HTR train and reports it to TSNR
3. TSNR sends collected data to IoT HTR.
4. IoT HTR calculates the speed of the HTR train from the tachometer data.
5. IoT HTR calculates the difference in speed given by the tachometer and GPS.

6. The Display displays a yellow warning message when the difference is between 5mph and 10mph ($5\text{mph} < \Delta\text{speed} < 10\text{mph}$).
7. The Display displays a red warning message when the difference is between greater than 10mph ($\Delta\text{speed} > 10\text{mph}$).
8. Log records slippage

Use case 6: Uninitialize IoT HTR
Primary actor: Operator
Secondary actors: IoT HTR
Goal in context: Turn off IoT HTR
Preconditions: IoT HTR is on
Trigger: Operator decides to uninitialize IoT HTR, that is, to press the power button
Scenario:

1. Operator presses the power button
2. IoT HTR Display turns off

Use case 7: View logs
Primary actor: Admin
Secondary actors: IoT HTR
Goal in context: To check and download operational logs
Preconditions: There is data in the logs, that is, IoT HTR is been initialized at some point; IoT HTR is initialized
Trigger: Admin decides to download the operational logs to check data
Scenario:

1. Admin logs into IoT HTR with admin privileges
2. Admin open the file location of the logs
3. Admin reads the logs

Exceptions:

1. Incorrect admin login: Warning message “Invalid Username or Password” is displayed, Operator re-enters
2. Cannot find file location, abort use case

Use case 8: Change configurations
Primary actor: Admin
Secondary actors: IoT HTR
Goal in context: Change the configuration of the HTR train
Preconditions: HTR train has IoT HTR system installed, IoT HTR is initialized
Trigger: Admin decides to change the configurations of the HTR train
Scenario:

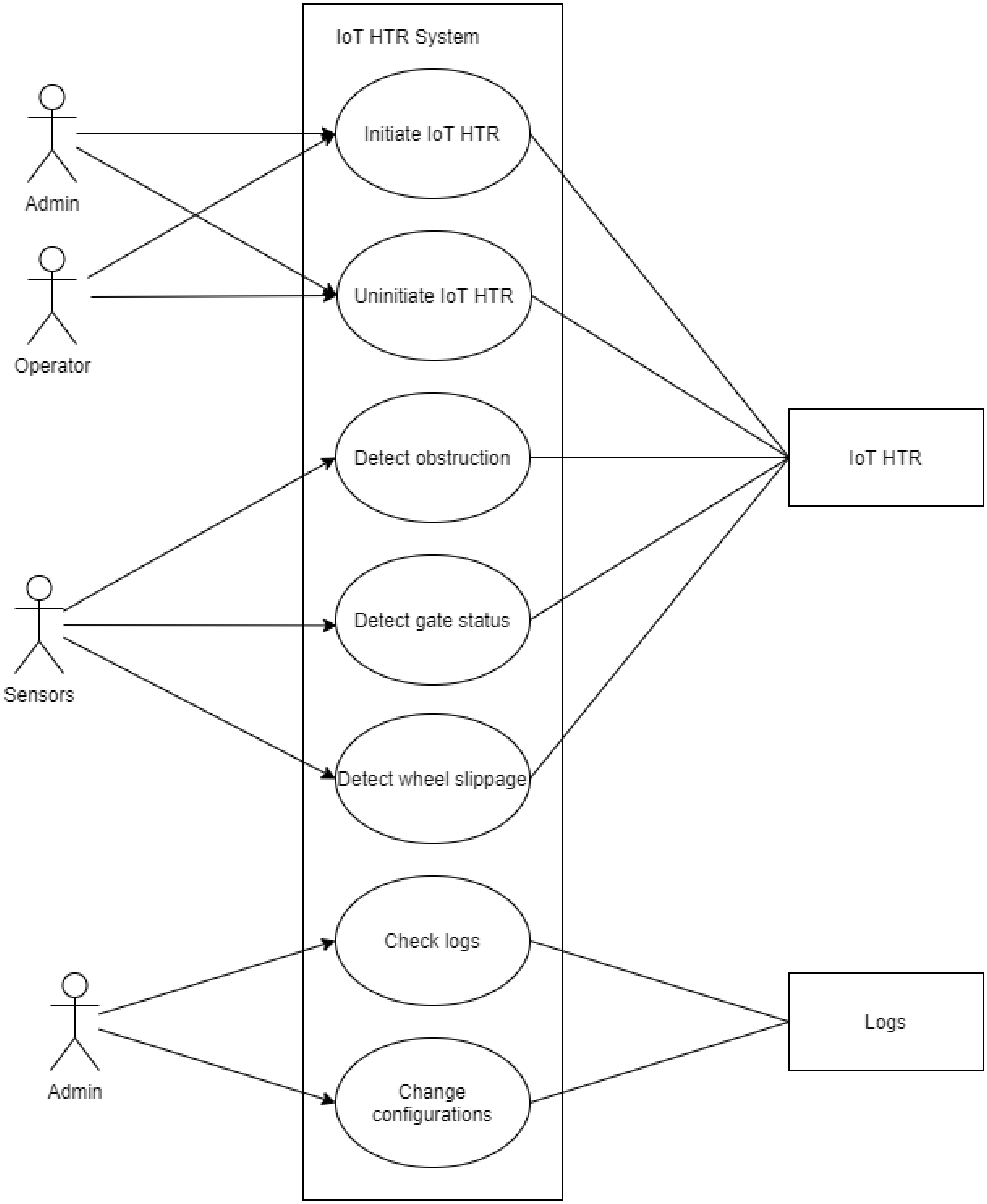
1. Admin supplies correct admin ID/password to IoT HTR
2. Admin makes appropriate changes to IoT HTR
3. Admin closes the configuration interface

Exceptions:

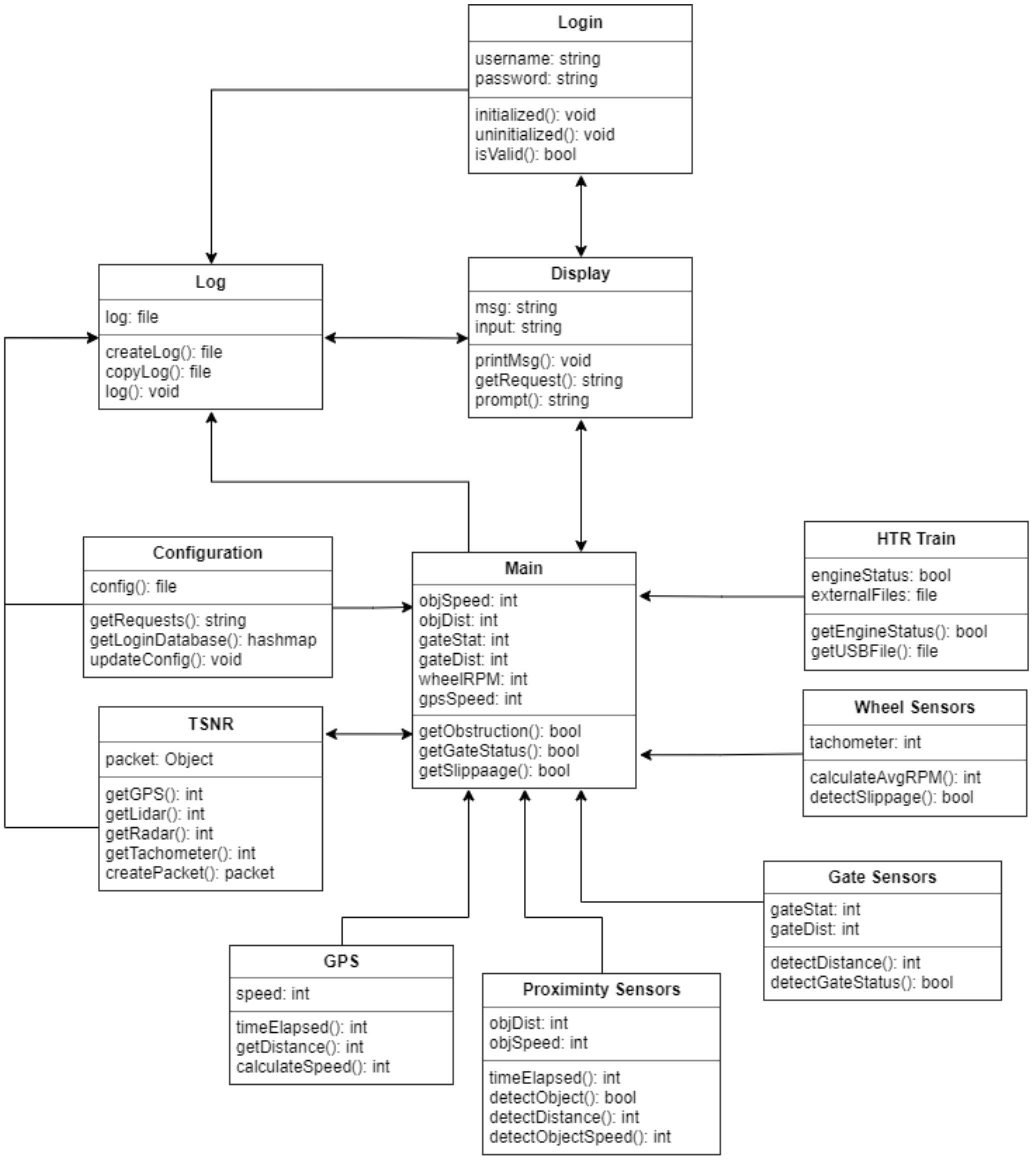
1. Incorrect login: Warning message “Invalid Username or Password” is displayed, Operator re-enters correct ID/password.

2. Fail to make configurations: Warning message "Failed to configure" is displayed, abort use case.

4.2 Use Case Diagram



4.3 Class Diagram



4.4 CRC Model Index Cards

Operator	
User who has limited access to IoT HTR	
Responsibility	Collaborator
Initiate IoT HTR	Main
Uninitiate IoT HTR	

Admin	
User who has complete access to IoT HTR	
Responsibility	Collaborator
Check logs	Log
Change configurations	Configurations

Main	
Takes in all the data from the TSNR, displays it, and determines if the operator needs to be alerted about hazards	
Responsibility	Collaborator
Continuously log	Log
Obstructions detection	Proximity sensors via TSNR
Slippage detection	GPS and tachometer via TSNR
Gate crossing detection	Gate sensors via TSNR

TSNR	
Takes in all the data from the sensors, processes it, and sends it to IoT HTR	
Responsibility	Collaborator
Send proximity (obstruction) data	Proximity sensors
Send global positioning data	GPS
Send wheel slippage data	Wheel sensors
Send gate crossing data	Gate sensors

Proximity sensors	

Sensors that can detect objects (stationary and moving) within a 2-mile radius of the train	
Responsibility	Collaborator
Check if there are stationary obstructions	Main
Gets the distance of stationary objects	
Check if there are moving obstructions	
Gets the distance of moving objects	
Gets the speed of obstructions	
Display warning message for obstructions and their position relative to the HTR train	
Display suggestion	

Wheel sensors	
Sensors that can detect if the train's wheels are slipping	
Responsibility	Collaborator
Get the rpm of the train's wheels	Main
Determine the speed based on the wheel's rpm	
Compare wheel sensor's speed with GPS's speed to check for slippage	
Display warning message for slippage	
Display suggestion	

Gate sensors	
Sensors that can detect a gate crossing' status within a 3-mile radius of the train	
Responsibility	Collaborator
Determines if gate crossings are opened/closed	Main
Gets the distance of a gate crossing	

Display warning message for open gate crossing	
Display suggestion	

GPS	
Gets the global positioning of the HTR train	
Responsibility	Collaborator
Get the speed of the HTR train	Main

Log	
Logs all events while IoT HTR is on	
Responsibility	Collaborator
Get current time Log events	Main

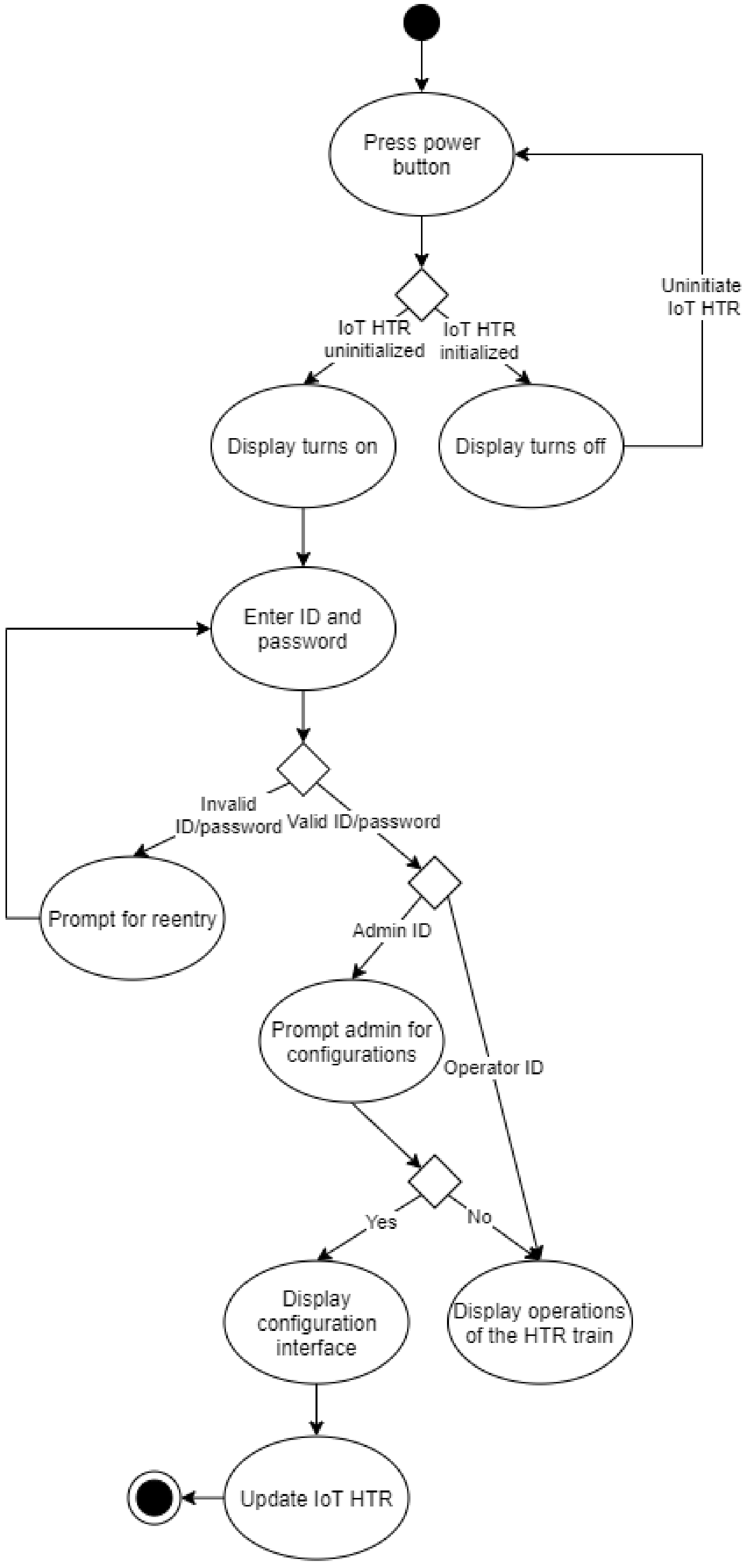
Configurations	
User configurations for IoT HTR	
Responsibility	Collaborator
Read admin input to change/update system settings	Main

Display	
Display outputs from main	
Responsibility	Collaborator
Read main outputs and print to GUI	Main

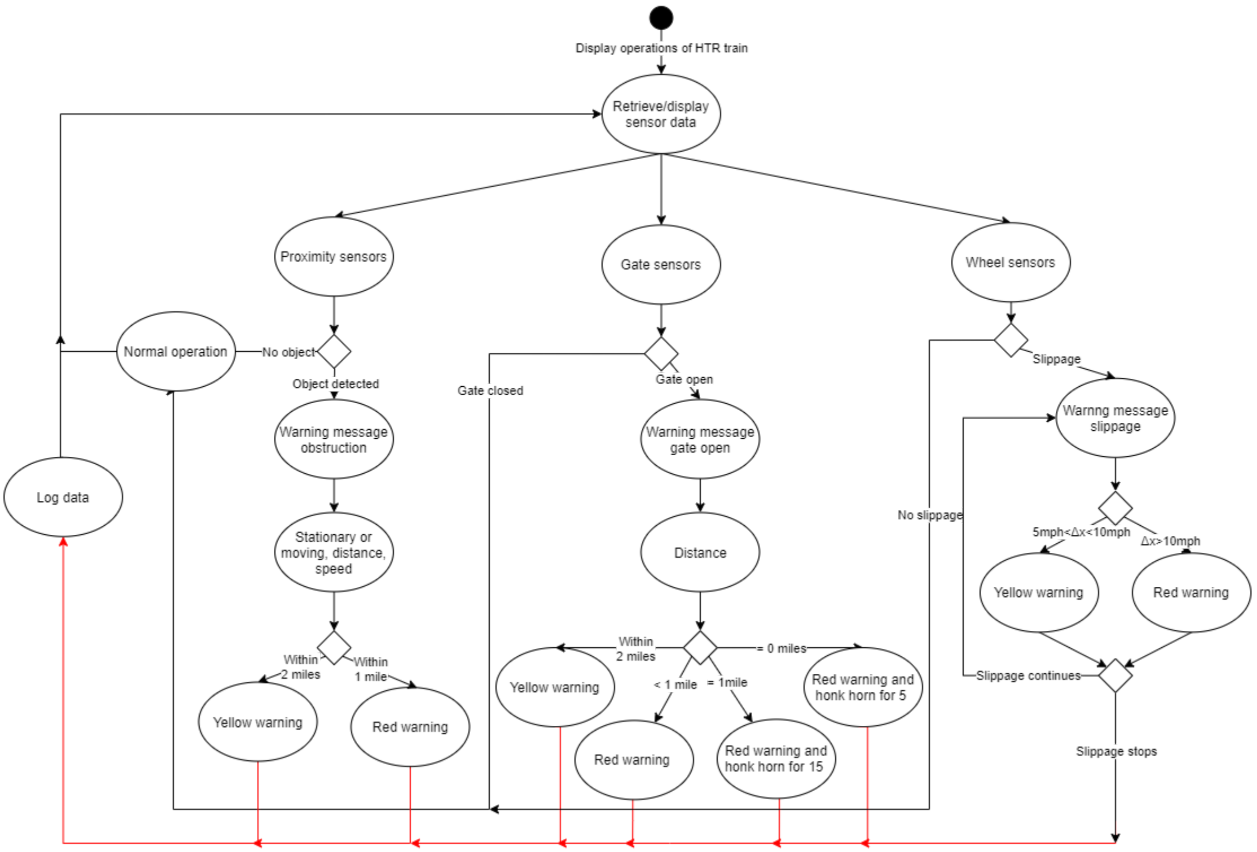
Login	
Login system	
Responsibility	Collaborator
Checks usernames and password before allowing a user into the software	Main

HTR Train	
All external inputs from the HTR train	
Responsibility	Collaborator
Sends to the Main that status of the Engine of the HTR train	Main
Detects external USB storage I/O	

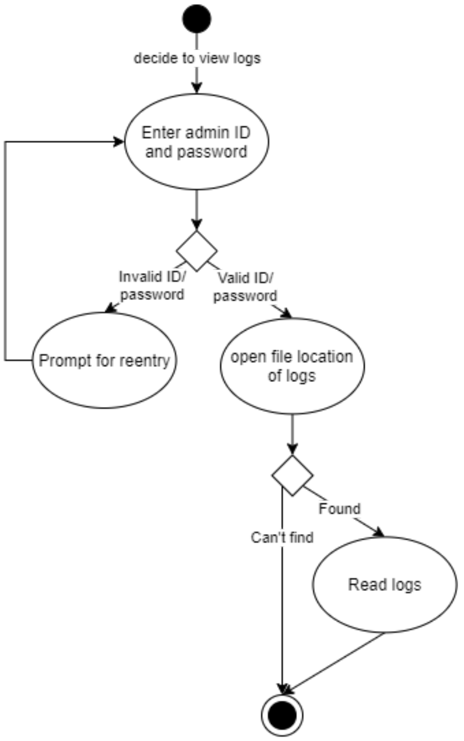
4.5 Activity Diagram



Precondition: IoT HTR has already been initialized

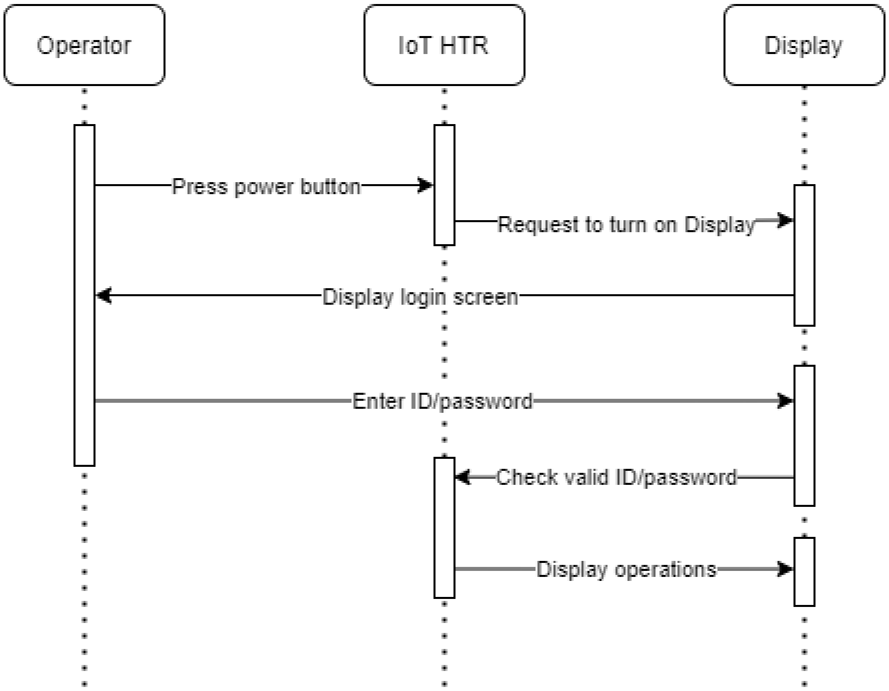


Precondition: IoT HTR has already been initialized

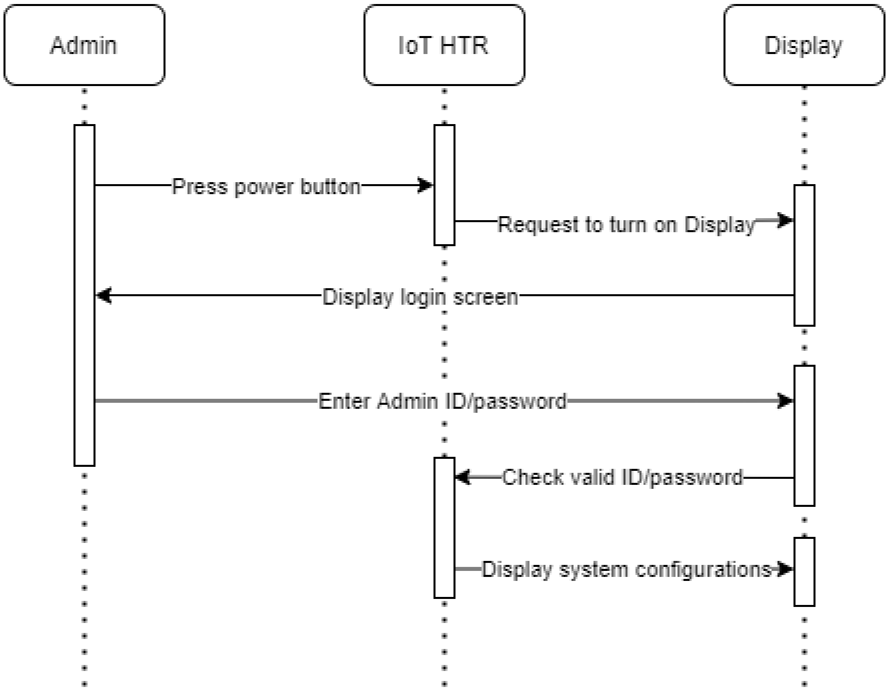


4.6 Sequence Diagrams

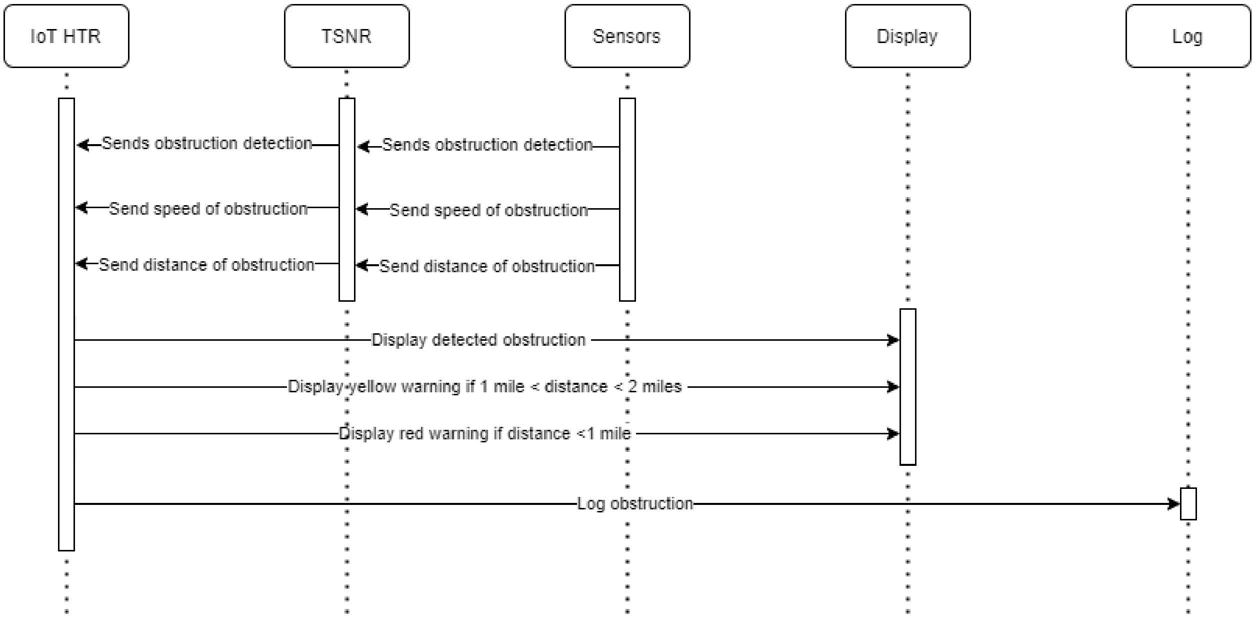
Use case 1:



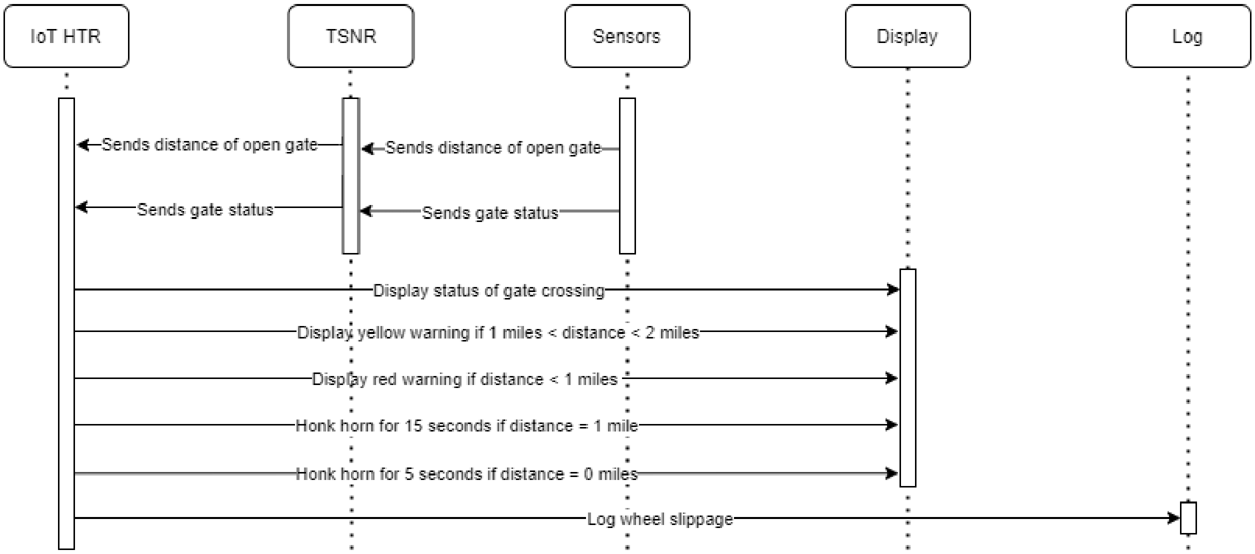
Use case 2:



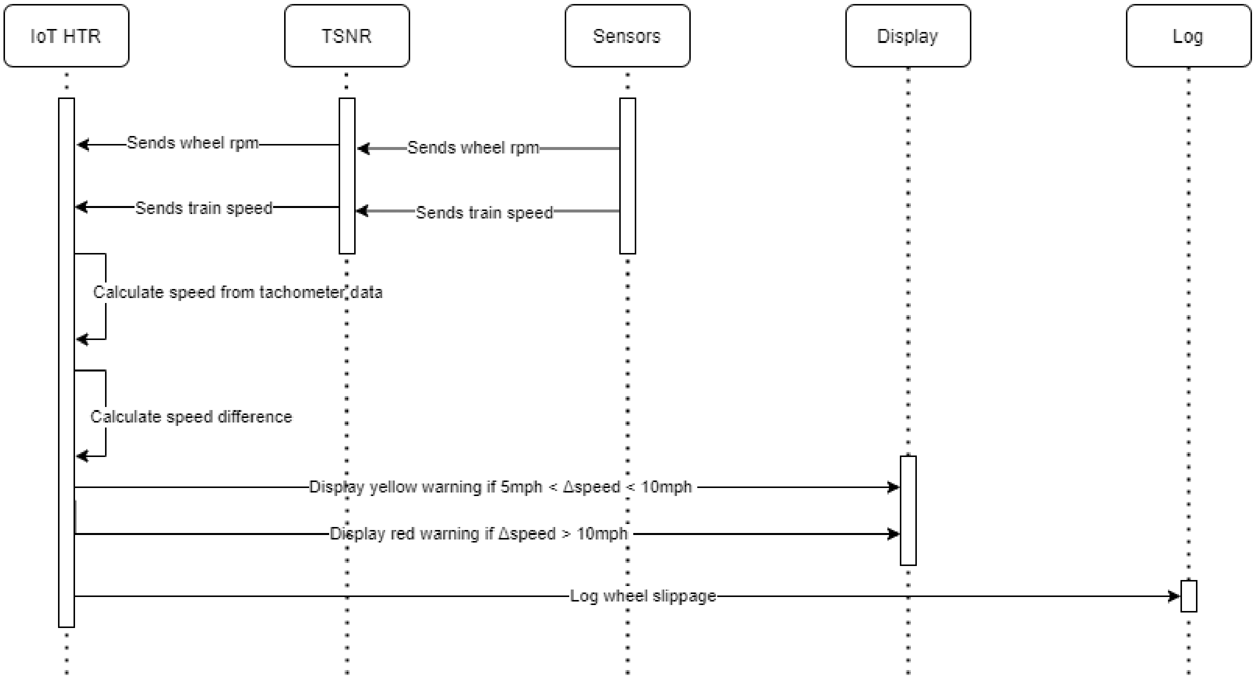
Use case 3:


IoT HTR
TSNR
Sensors
Display
Log
Sends obstruction detection
Sends obstruction detection
Send speed of obstruction
Send speed of obstruction
Send distance of obstruction
Send distance of obstruction
Display detected obstruction
Display yellow warning if 1 mile < distance < 2 miles
Display red warning if distance <1 mile
Log obstruction

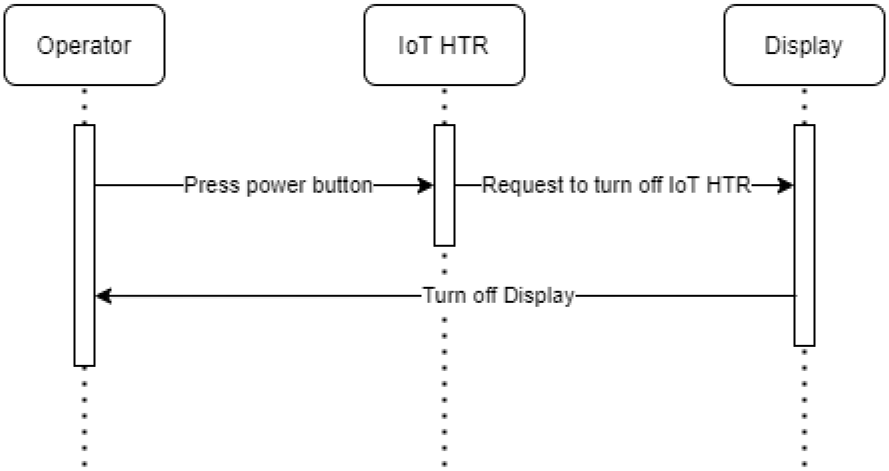
Use case 4:


IoT HTR
TSNR
Sensors
Display
Log
Sends distance of open gate
Sends distance of open gate
Sends gate status
Sends gate status
Display status of gate crossing
Display yellow warning if 1 miles < distance < 2 miles
Display red warning if distance < 1 miles
Honk horn for 15 seconds if distance = 1 mile
Honk horn for 5 seconds if distance = 0 miles
Log wheel slippage

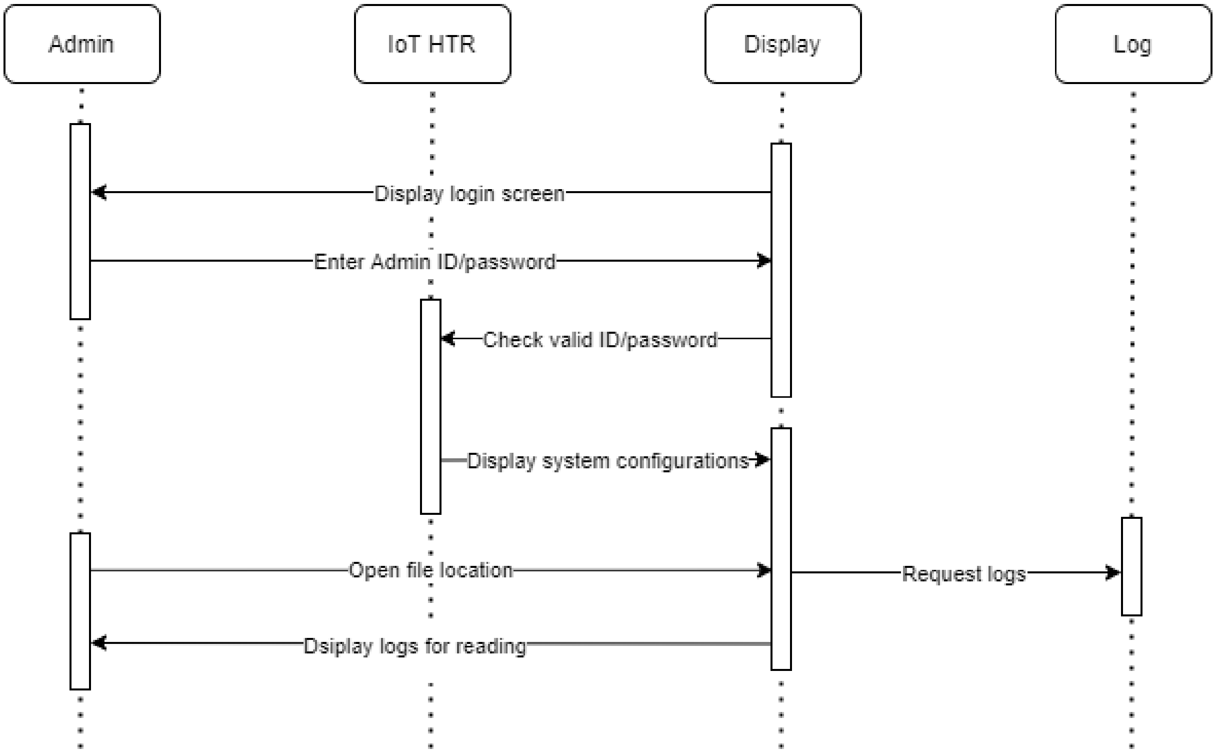
Use case 5:


IoT HTR
TSNR
Sensors
Display
Log
Sends wheel rpm
Sends wheel rpm
Sends train speed
Sends train speed
Calculate speed from tachometer data
Calculate speed difference
Display yellow warning if 5mph < Δspeed < 10mph
Display red warning if Δspeed > 10mph
Log wheel slippage

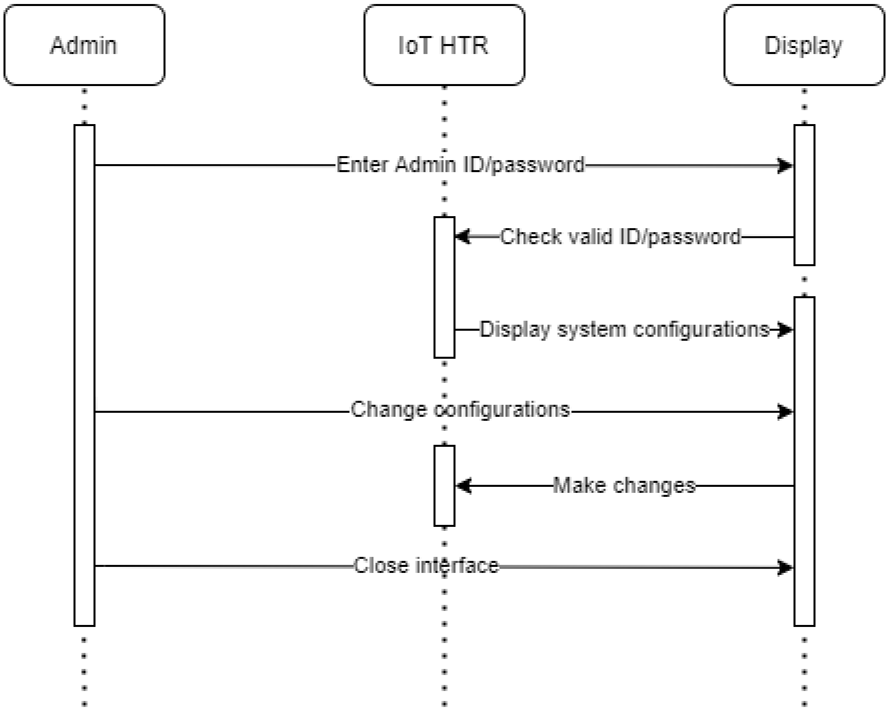
Use case 6:



Use case 7:

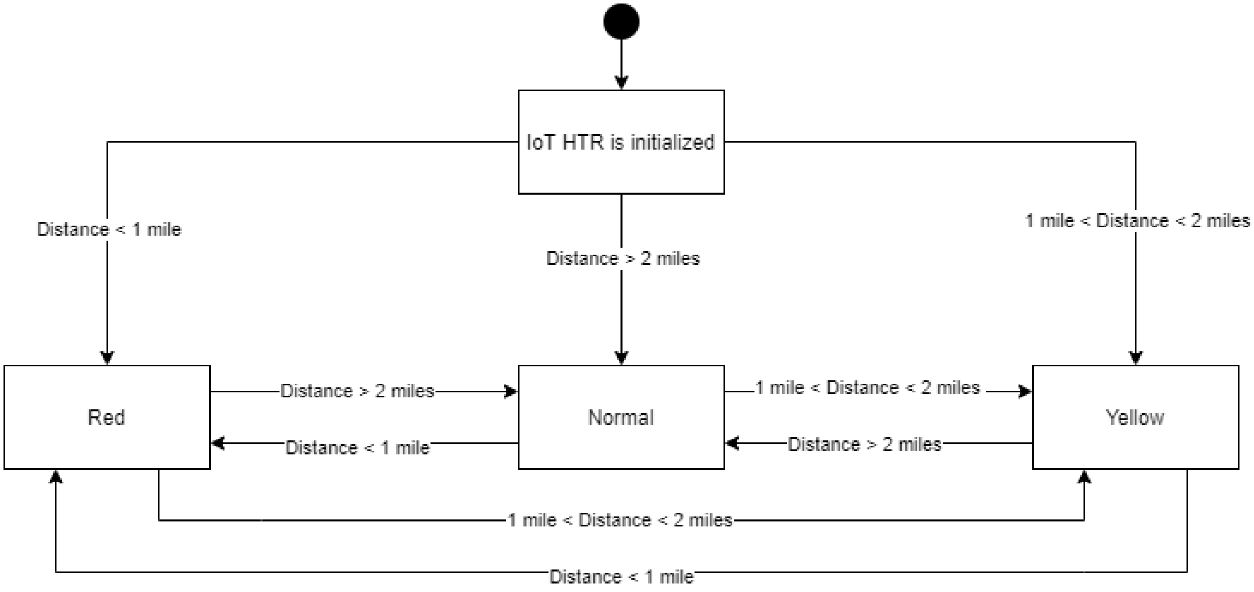


Use case 8:

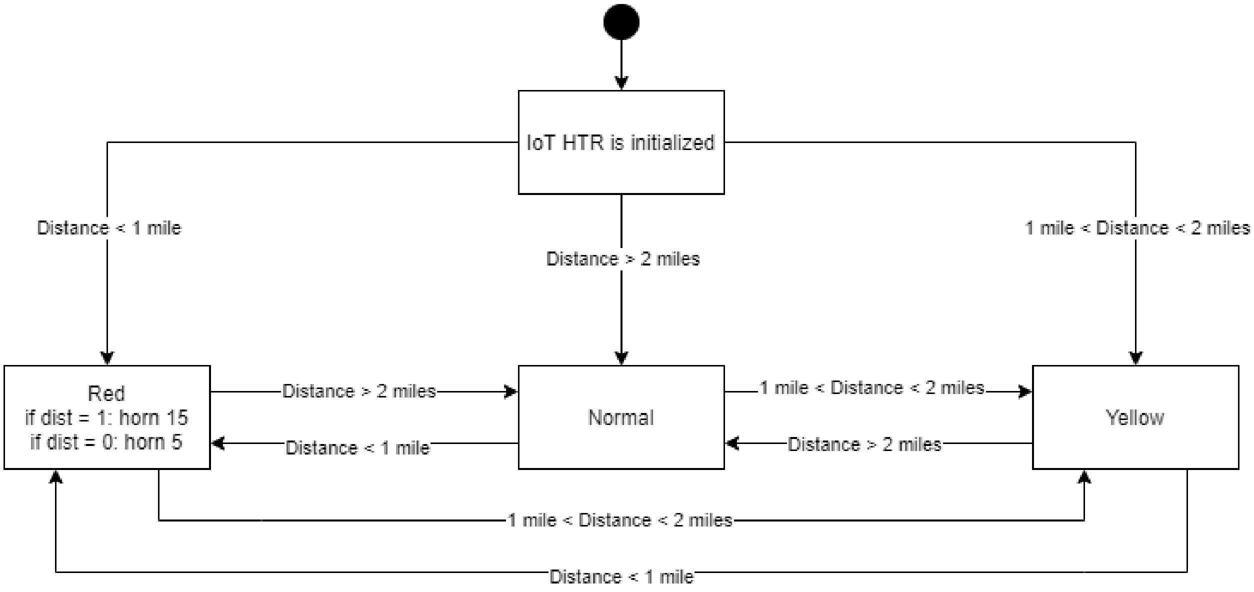


4.7 State Diagram

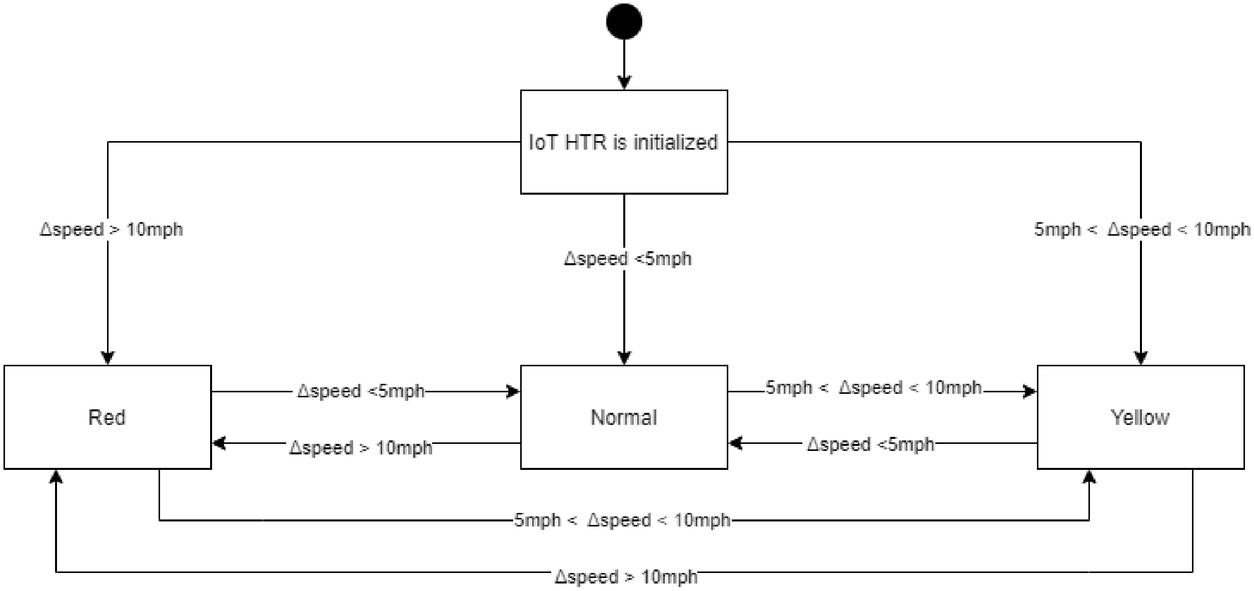
Object detection state diagram:



Gate detection state diagram:



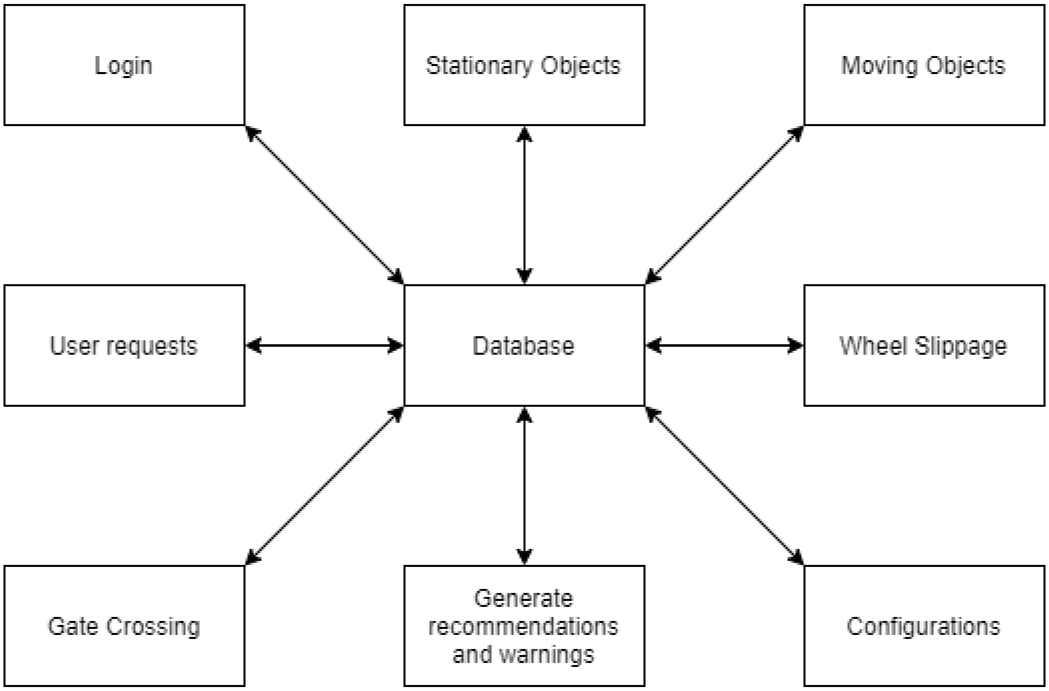
Wheel slippage detection:



5. Software Architecture

5.1 Data Centered Architecture

Arrows from the client software into the database illustrate the movement of sensor data going into the database via TSNR while arrows from the database into the client software represent sending data inputs into the respective process (e.g. stationary object detection) and displaying it on the Display.



Pros:

- All IoT data is logged into a central system
- The log is easily accessed from a single location
- Very modular and scalable in adding more software modules

Cons:

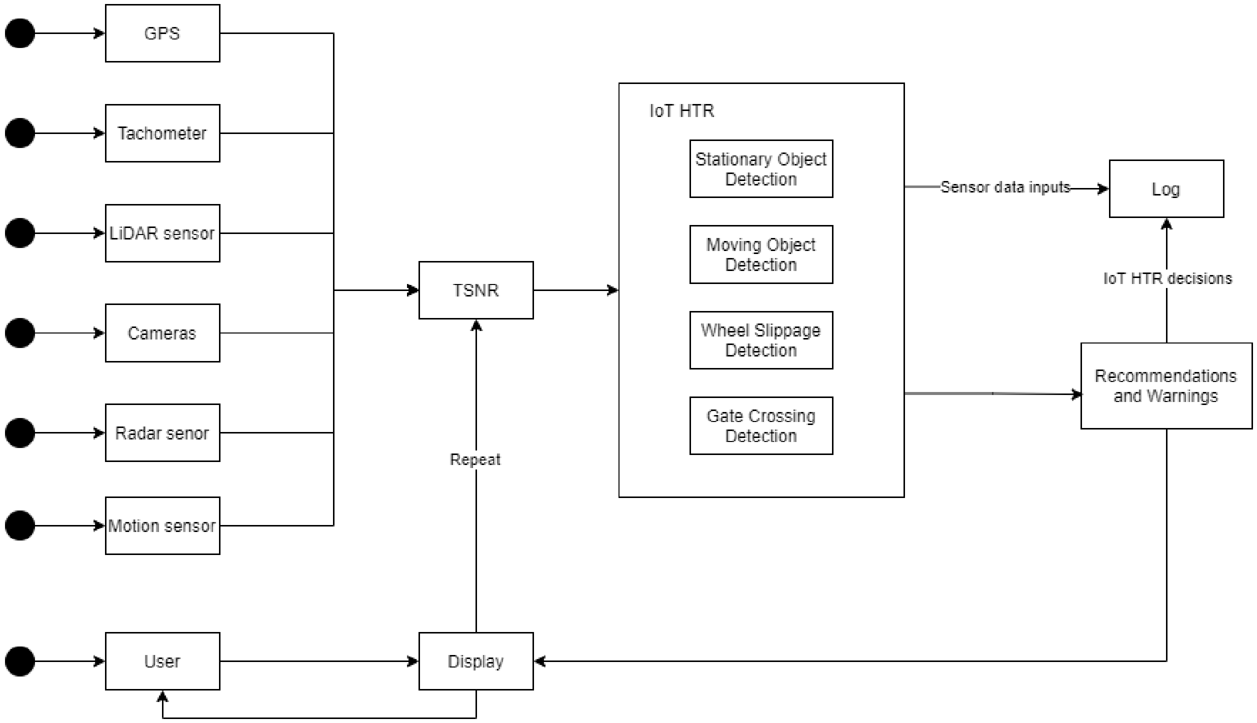
- High dependency between the central system and individual software
- Failure of central system propagates into a failure of the whole log
- Data structure of central system must be consistent for all software
- The log must manage high traffic requests
- It is vulnerable to failure and data replication/duplication
- Partitioning of data/links slows down how quickly the data can be accessed

The Data Centered Architecture does not suit IoT's dynamic nature. The main issue of a Data Centered Architecture implementation into IoT is that the data center is too slow. Our system is a real time system therefore there is a time constraint for all processes. In a Data Centered Architecture implementation of IoT it would need to get the data from the sensors and store it into the database. Then IoT would have to get that data to analyze/process it. The decisions from IoT would also need to be stored into the database. At this point nothing has even been displayed on the Display. As illustrated, this is much too slow for a mission critical system.

Being primarily a storage-based architecture, the only IoT module that somewhat fits this role is the log. Even then, the log is not a suitable choice for utilizing the Data Centered Architecture. Each log file stores IoT's data from inilizationg to uninitialization, which does not require massive storage space typically found in a large database. Additionally, the log constantly updates and changes its stored values, which is not a feature of long-term database storage. Our IoT operates on current (most recent) data which is dynamic, therefore a file with older values serves no purpose to the calculations made by the software.

5.2 Data Flow Architecture

The following evaluation refers to a Pipes and Filters Approach to Data Flow Architecture.



Pros:
- Allows IoT to have high throughput of data processing
- Simplifies IoT's software execution into block chains
- Each block chain can run sequentially or independently in parallel (flexible architecture)
- Individual software filters are easy to maintain and modify

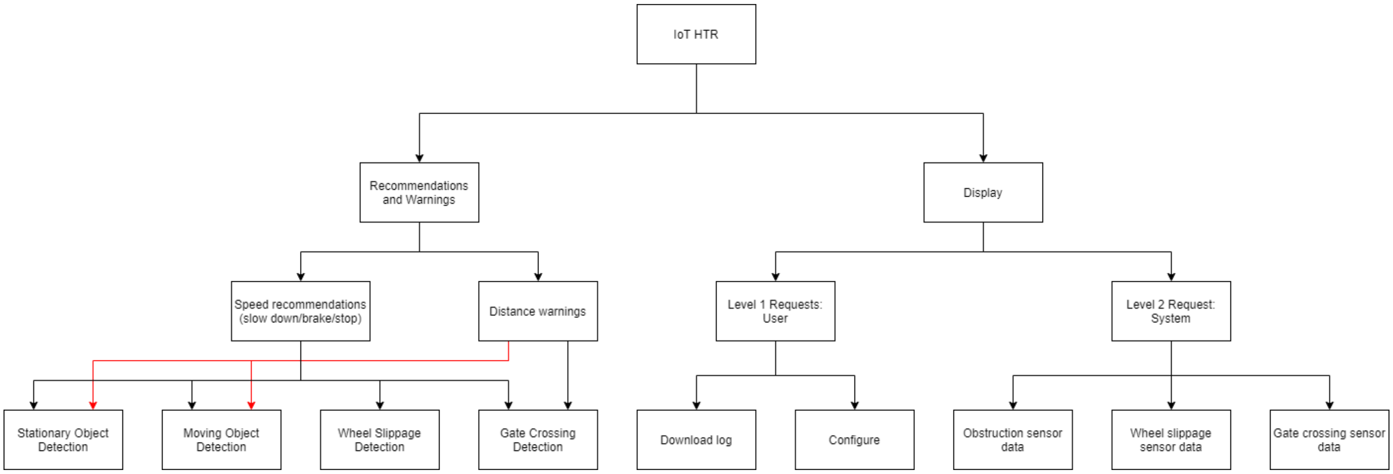
Cons:
- More sequential software increases IoT's processing time
- Does not allow IoT to modify/redirect existing filters dynamically
- Failure in a software filter will create bottlenecks in subsequent filters
- Software filters have a hard time working on a large problem
- Software filters cannot effectively store calculations
- Not suitable for dynamic interactions and dynamic change

The Data Flow Architecture is a decent option for IoT. One of the highlights of this architecture is the ability for IoT to run data processes in parallel. If all six types of sensors were to haphazardly enter IoT's four main detection processes, IoT would likely hang. However, by running these processes separately in *parallel and with TSNR, IoT would have a higher throughput of data. Additionally, chains of parallel filters create distinct blocks for the data to flow, simplifying the readability and development of IoT's software.*

However, the biggest drawback of having predefined filter blocks is that IoT cannot make decisions outside those predefined filters. In this sense there is no dynamic decision making. For example if the rails were icy, IoT HTR would not be able to alert the operator to proceed with caution thereby putting the passengers and operator at risk. By having a fixed number of filters, there can only be a finite number of possibilities that the data can flow. This inhibits IoT from making decisions to larger problems outside the scope of what was naively intended. Furthermore, having a finite number of filters means that IoT cannot effectively store calculations and values, a key component in generating recommendations. Adding more filters to account for these variations just increases disorganization. Thus, although the Data Flow Architecture is promising in its parallel filters, its fixed nature prevents IoT from running effectively.

5.3 Call Return Architecture

The data enters the model in the following manner: Sensor Data → TSNR → IoT HTR



Pros:

- Reduces complexity of IoT into smaller and simpler subprograms (separation of concerns)
- Subprograms can be hidden from one another
- Groups of subprograms create easy to understand hierarchy of IoT
- Modular in adding new subprograms to IoT

Cons:

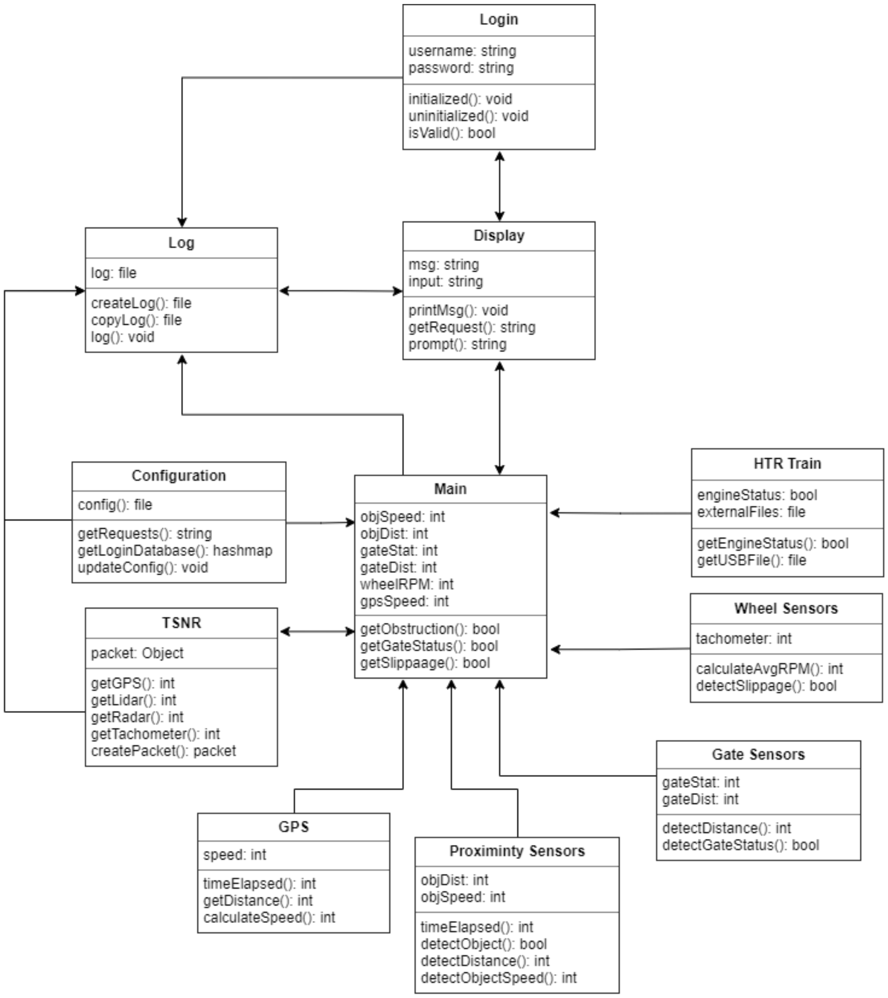
- High dependency between output of one subprogram and the input of another subprogram
- Failure of subprogram will propagate up into IoT's main process
- Does not scale well as IoT's complexity increases
- More subprograms increase process time of IoT

The Call Return Architecture is a strong candidate for IoT because of its hierarchical system. Through visualizing IoT from a hierarchical perspective, complex processes such as recommendation and warning generation can be further broken down into much simpler and easier to manage subproblems. For example, all terminal requests in IoT can be grouped and divided into Level 1 and Level 2 requests, and distance warnings can be derived from object detection and Gate Crossing detection.

Additionally, the hierarchical approach in the Call Return Architecture satisfies IoT's computation requirement. At every second, IoT must process TSNR's packet which includes data from six different types of *sensors. Then, IoT HTR must complete calculations for each individual sensor data. This requires IoT to be quick in computing, storing, and retrieving values to anticipate the next TSNR packet a moment later. Because this architecture is built on sharing and returning data between subprograms, this scenario is good for supporting IoT's calculation process.*

The one fault in this architecture is complications arising from scaling up IoT. When IoT becomes bigger, there must be exponentially more subprograms in the hierarchy, which will slow down development and computation time. Fortunately, IoT at this moment is not big enough to pose scalability issues while using the Call Return Architecture.

5.4 Object Oriented Architecture



Pros:

- Uses abstraction to reduce complexity of IoT
- Objects and methods are reusable, modular, and dynamic
- Implementation of IoT objects are hidden from one another
- IoT objects can be constructed and executed independently
- Classes can be reused without having to change them fundamentally for IoT (e.g. sensors)
- Easy to implement the IoT login process

Cons:

- Inefficient for high performance calculations in IoT

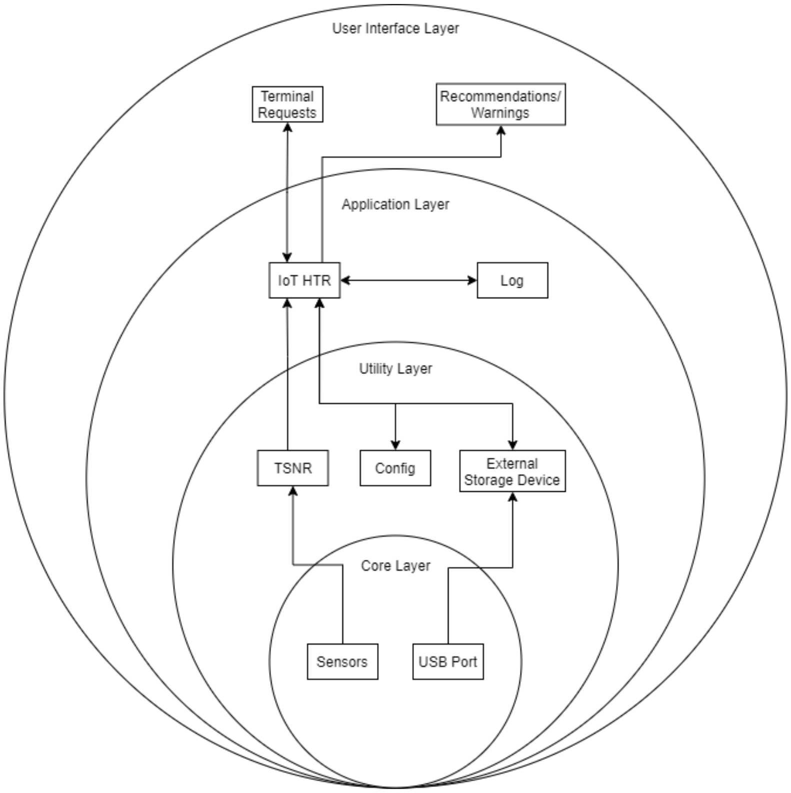
- Prone to excess and large amounts of code (cost of size and performance)
- Difficult implementation without a high level plan of IoT

The Object Oriented Architecture is an excellent option for IoT. The main advantage of utilizing this architecture is using abstraction to minimize and simplify IoT's software, By only specifying required classes and methods in IoT's objects, abstraction enables only the bare minimum and necessary amount of code to sufficiently run IoT. Thus, the run time of IoT can be reduced while simultaneously increasing overall efficiency, readability, and performance of IoT. Furthermore, by creating distinct classes such as the Config or Log, IoT's objects can run independently without the risk of one object's dependencies clashing with another object.

Another major benefit of using an Object Oriented Architecture is that classes can be reused without having to write them again. For example, a sensor class can be reused with little modification as most sensors are similar. Furthermore the logic for making recommendations/warnings are generally the same. They all have the same output essentially and require similar inputs. The only main difference in each logic is how to process the data. For instance, in the stationary object, IoT has to receive from the sensor that the object is moving at 0mph while for the moving object it needs the speed to be greater than 0mph. Another reason why an Object Oriented Architecture is easy to implement is because sensors can be represented as objects and its attributes will be the sensor data.

However, the Object Oriented Architecture assumes that IoT's abstracted code is well optimized for performance and thoroughly planned out. Objects require an abundance of space to store methods and stored data. When the code is unoptimized, duplicate and unneeded objects may still linger after its initial use, taking up valuable space in IoT. Additionally, it is difficult to optimize IoT's code without a thorough understanding and high level software plan of IoT. There needs to be a balance between having the bare minimum amount of code and a sufficient amount to successfully execute IoT.

5.5 Layered Architecture



Pros:

- Distinct layers simplifies complexity and understanding of IoT
- Divides IoT into four unique hardware, software, and interface layers
- Changes in one layer do no significantly impact development of other layers
- Robust for grouping and customizing IoT's functionality, specs, and interface

Cons:

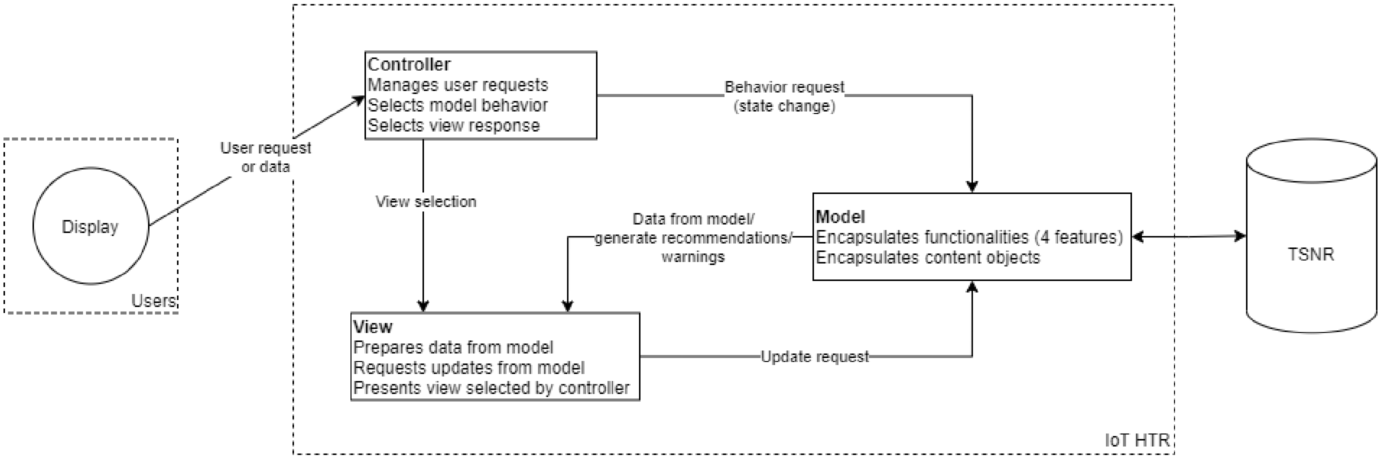
- More independent layers make it harder to maintain and modify IoT
- Not effective for higher modularity and scalability of IoT
- Moving through multiple layers lowers performance and increases process time for IoT
- Layers that are far apart have a hard time working together
- Not necessarily the fastest as we are pushing sensor data through multiple layers

The Layered Architecture is a decent fit for IoT due to its powerful organization of IoT's design layout. By separating IoT's hardware, software, and interface into four unique layers, each layer will have a designated purpose in IoT that makes customization and maintenance of the software easier. Furthermore, having unique layers enables independence during the software development process. For example, modifying existing or adding new Terminal requests in the User Interface Layer will not have a significant impact on the Utility and Core Layers, as these layers are far away from the user interface Layer. This ensures faster development of IoT without severely altering development of other layers.

Similar to the Call Return Architecture, the only major flaw of the Layered Architecture is when scaling up IoT. Because there are only four unique layers, adding new modules to IoT will overflow their respective layers until it will become hard to organize and maintain each individual layer. However, as stated in the Call Return Architecture, IoT at this moment is not large enough to encounter this scenario. Thus, the Layered Architecture is a powerful organizer of IoT's layout

However, one of the biggest challenges the Layered Architecture faces when implemented in our IoT system is that sensor data has to be pushed through multiple layers. This slows down processing time and presents performance and real time issues. As stated before, IoT runs in real time and is a mission critical system therefore a Layered Architecture would pose a threat to the safety of passengers due to its slowness.

5.6 Model View Controller Architecture



Pros:

- Oriented for IoT's user interface
- Splits IoT into Controller, Model, and View units enables for easier maintenance and readability

- Changes in one unit do not significantly impact the other two units
- Three separate units ensures independence

Cons:

- Not expandable outside of user interface
- Difficult implementation without a high level plan of IoT's interface
- IoT data follows a strict flow in this architecture
- Model unit becomes more complex as functionalities are added to IoT
- View is dependent on the Controller and Model
- Model oftentime does much of the work

The Model View Controller (MVC) Architecture is a good option for IoT. For IoT, the MVC Architecture fits quite nicely when implemented properly. Similar to how the Layered Architecture improves organization of IoT by separating the hardware, software, and interface into four different layers, the MVC architecture separates IoT's terminal interface into a Controller, View, and Model unit. Each unit is independent and serves its designated role in IoT. Many of the features and logic of IoT can easily be translated into a MVC Architecture. For example, the Controller unit is responsible for maintaining user requests, while the View unit is responsible for displaying recommendations and warnings onto the Display for the user. In addition to representing the structure of IoT, it also can represent a simple and straightforward login wherein the user requests from the controller to login, the controller requests the model to log into IoT, and the view displays the UI and success of logging in.

The biggest problem with this architecture is that it is hard to implement the three units without a thorough understanding and high level plan for IoT's interface. Although only having three units may seem simple to implement, each unit is unique and has its own data flow. Interactions between each unit requires knowledge of how the two units function separately and the triggers that cause the sharing of information in the first place.

5.7 Conclusion

The best architecture for our implementation of IoT is an Object Oriented Architecture. The Object Oriented Architecture will provide an excellent avenue to use abstraction to minimize and simplify IoT's software. An Object Oriented Architecture substantially allows for a streamlined program with only the most essential code. Thus, the run time of IoT can be reduced while simultaneously increasing overall efficiency, readability, and performance of IoT. These qualities are extremely important to a real time and mission critical system. Many of IoT's functionalities behave in comparable ways and therefore classes and objects can be utilized to reduce rewriting unnecessary code. Sensors can easily be represented as objects and its attributes will be the sensor data. Additionally, because we are all familiar with this architecture the team will be able more easily implement it effectively and therefore have a faster software development process.

5.8 Implementation

5.8.1 Model

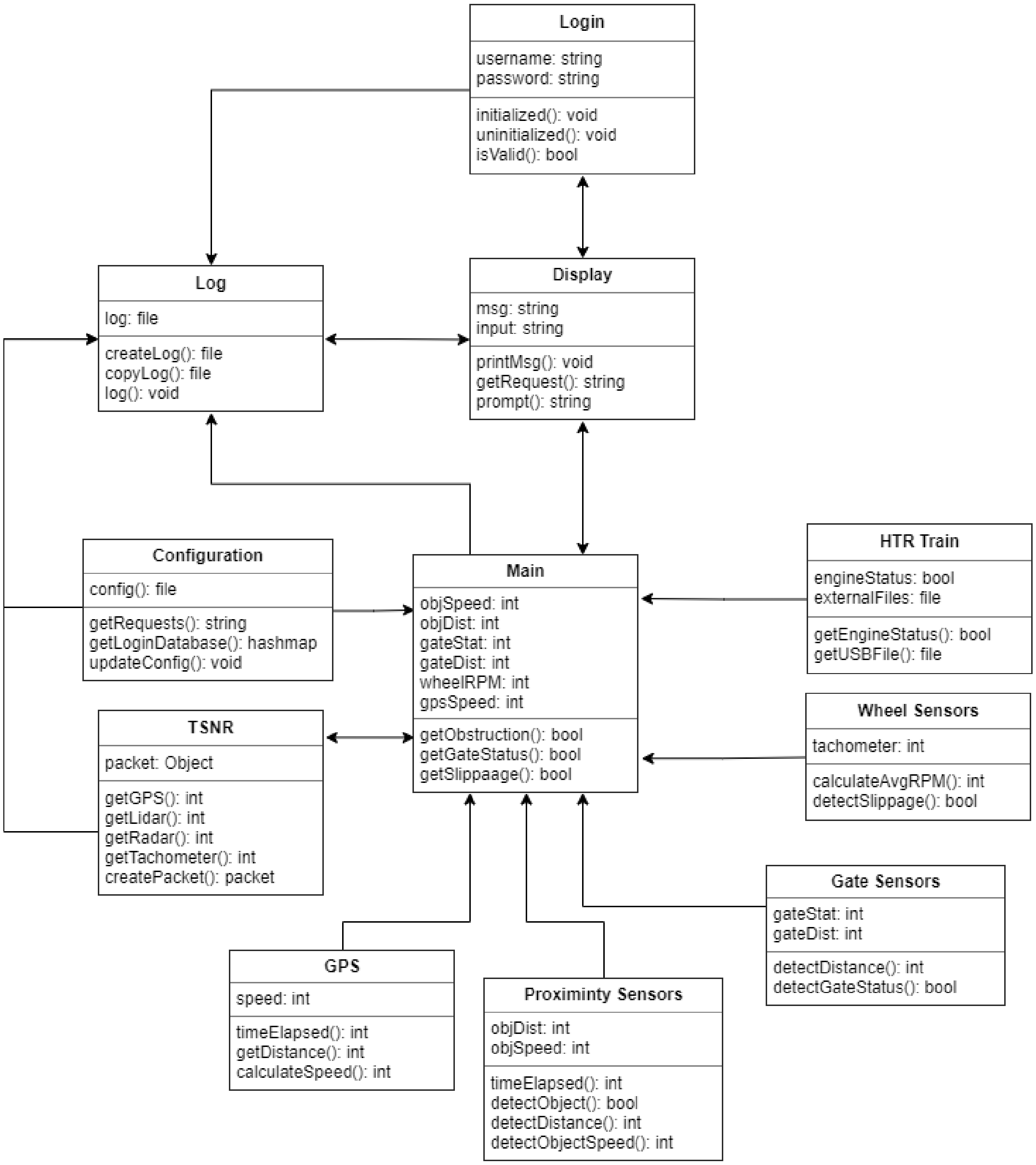


Figure 4: *Data will be fed into the software by means of the TSNR. The physical sensors (not shown in the figure) send data into the TSNR which is then processed and sent to IoT HTR's main function. The main calls on sensor classes which handles further calculations that need to be made. For example, calculating the average RPM of the wheels from the four tachometer sensor data will be handled in the Wheel Sensors Class. The main logs the data and sends recommendations and warnings to the Display.*

5.8.2 Classes

Login: *Takes a username and password, check if they are valid and if so allows the user to login*

1. username: a string passed through by the user to be used in authentication to the software
2. password: a secret string passed through by the user to be used in authentication to the software
3. initialized(): a void method that turns on the IoT HTR by logging the initialization and turning on the Display
4. unitialized(): a void method that turns off the IoT HTR by logging the uninitialization and turning off the Display
5. isValid(): a method that returns boolean based on the username and password being a valid pair. Works by checking the supplied username/password with the login database housed in the configurations class.

Log: *creates log file and stores text (in the form of warnings) into said log file*

1. log: a file consisting of the entire log of events that were encountered during the initialization of IoT HTR
2. createLog(): A function that creates a log file with current data
3. copyLog(): A function that returns a copy of the current log by retrieving it and copying it onto an external storage device
4. log(): A void function that updates the log by taking data passed into it (e.g. sensor data, object detection, recommendations, etc) from the main and writing it into the log file

Display: *essentially an enhanced print function*

1. msg: a string that is displayed to the user on the front end
2. input: a string that is taken in by the user and used to request a certain action by the system (e.g. log in by entering a username/password)
3. printMsg(): void function that prints message to screen
4. getRequest(): function that calls for an action as directed by the user (e.g. requesting to display HTR operations by supplying a valid username/password)
5. prompt(): function that displays what is requested

Configuration: *the area of code regarding the configuration e.g. the wheel circumference*

1. config(): returns a file consisting of the configurations for the IoT
2. getRequest(): function that returns a string that access to the configurations interface has been granted
3. getLoginDatabase(): returns a hashmap of all of the users on the system with each of their corresponding usernames/passwords and privileges
4. updateConfig(): updates the configuration file

HTR Train: *relates to all external input not feed by sensors/TSNR e.g. auxiliary storage device and power button inputs*

1. engineStatus: a boolean variable that is true if the engine is turned on and working correctly and is set to false otherwise
2. externalFiles: a file variable consisting of any external files concerning the IoT
3. getEngineStatus(): returns the value of the engineStatus boolean
4. getUSBFile(): returns a copy of the logs onto the USB (external storage device)

Main: *the "coordinator" of all the TSNR sensor data and calling warnings to be made to the Display()*

1. objSpeed: an int value representing the speed of objects
2. objDist: an int value representing the distance of objects
3. gateStat: an int value representing the status of the gate (0 for closed, 1 for open)
4. gateDist: an int value representing the distance of the gate

5. wheelRPM: an int value representing the rpm of the wheels
6. gpsSpeed: an int value presenting the speed of the train according to the GPS
7. getObstruction(): a boolean method that communicates that a warning and recommendation need to be displayed. True means there is an object otherwise false is returned. To see how an obstruction is detected go to the Proximity Sensor Class
8. getGateStatus(): a boolean method that communicates that a warning and recommendation need to be displayed. True means there is an open gate crossing otherwise false is returned. To see how a gate crossing is detected go to the Gate Sensor Class
9. getSlippage(): a boolean method that communicates that a warning and recommendation need to be displayed. True means there is wheel slippage otherwise false is returned. To see how wheel slippage is detected go to the Wheel Sensor Class

GPS: *handles the calculation of the speed of the train from the GPS data*

1. speed: an int to represent the speed of the HTR train based on the GPS data
2. timeElapsed(): is a method that uses the timer to determine the time elapsed since the last packet of GPS data was received from TSNR
3. getDistance(): is a method that calculates the distance traveled based on the GPS data
4. calculateSpeed(): returns an int that represents the speed of the train by dividing the getDistance() by the timeElapsed()

Proximity Sensors*: *handles the calculation of obstruction data; Detects if there is an obstruction by mapping the surround with the LiDAR and camera sensor data; Calculates the speed of the object by dividing the distance over the time elapsed; Outputs a warning flag if there is an obstruction as defined in the above two statements; Outputs if it is a stationary object by checking if the speed is 0mph otherwise the object is moving; Outputs a recommendation flag based on the speed, distance, and location of the object*

1. objDist: an int representing the distance of objects
2. objSpeed: an int representing the speed of objects
3. timeElapsed(): is a method that uses the timer to determine the time elapsed since the last packet of proximity data was received from TSNR
4. detectObject(): a boolean method that returns true if there is an object and false otherwise
5. detectDistance(): an int method that gets the distance of the object
6. detectObjectSpeed(): an int method that returns the speed of the object

Gate Sensors*: *handles the calculation of gate sensor data; Outputs a warning flag to the main if the gate is opened; Outputs a recommendation flag to the main based on the distance of the gate crossing (3 mile radius)*

1. gateStat: an int representing the status of the gate (0 for closed, 1 for open)
2. gateDist: an int representing the distance of the gate
3. detectDistance(): returns an int that represents the distance of the gate crossing
4. detectGateStatus(): returns a boolean true if the gate is open otherwise false.

Wheel Sensors: *handles the calculation of wheel sensor data; Outputs a warning and recommendation flag to the main if slippage is present (Δspeed >5mph) by comparing the difference in the speed from the GPS and the speed from the tachometer data*

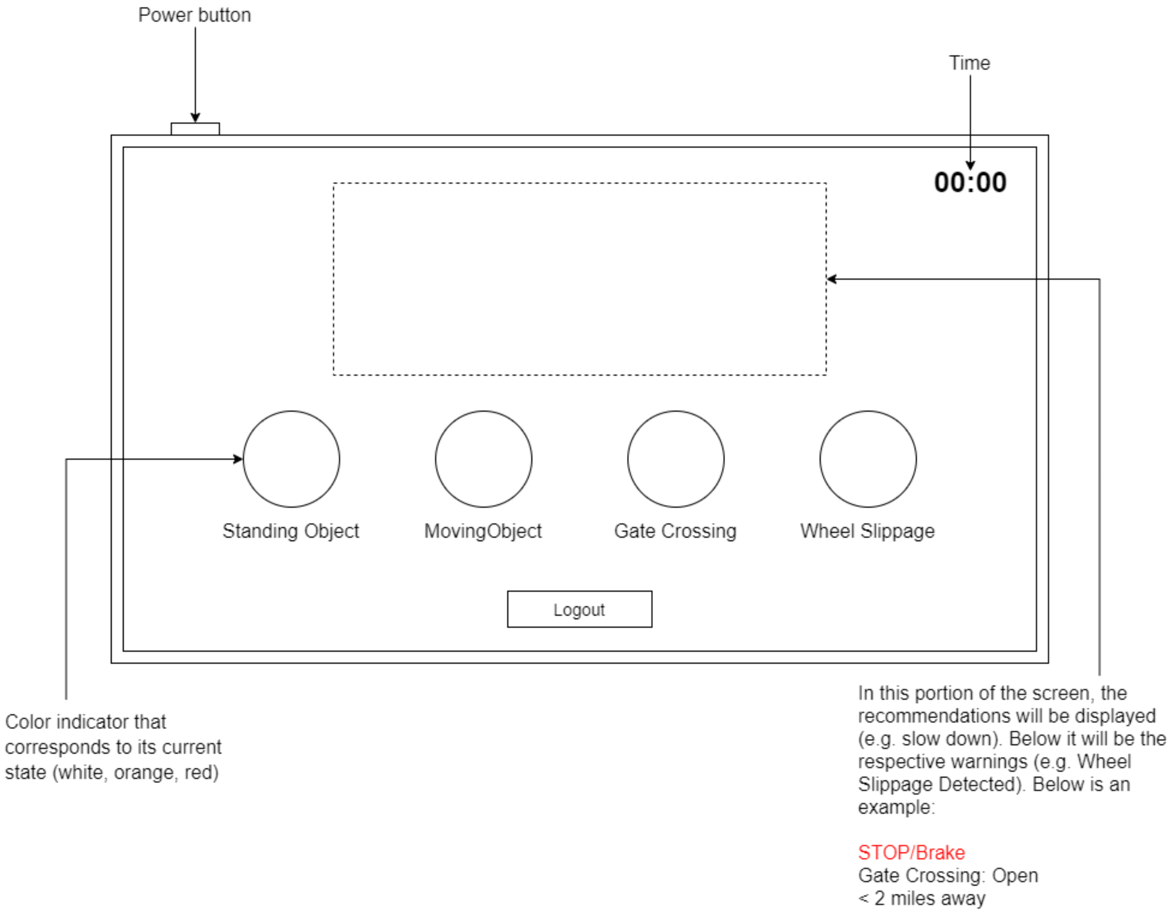
1. tachometer: an int that represents the RPM of the wheel it is attached to
2. calculateAvgRPM(): a method that returns an int by adding all four wheel RPM data and dividing it by four

3. detectSlippage(): a method that returns true if the difference between the speed from the GPS and the speed from the tachometer data is greater than 5mph

TSNR: *gets raw data from the physical sensors and sends them to the Main() as packets of information*

1. packet: an object that contains sensor data
2. Each getX() method receives data from the respective sensor
3. createPacket(): receives all sensor data by using the getX() methods and creates packets of data to be sent to IoT HTR.

5.8.3 Display UI



Color indicator that corresponds to its current state (white, orange, red)

In this portion of the screen, the recommendations will be displayed (e.g. slow down). Below it will be the respective warnings (e.g. Wheel Slippage Detected). Below is an example:

STOP/Brake
Gate Crossing: Open
< 2 miles away

6. Code

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.GridLayout;
import javax.swing.BoxLayout;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JTextField;
import java.math.*;
import java.io.FileWriter;
import java.io.IOException;
import java.sql.Time;
// https://www.tutorialsfield.com/login-form-in-java-swing-with-source-code/

/**
 * Display.java is the GUI of the IOT HTR project and allows for manual
input testing.
 * It displays a login where the
 *       username is admin or user
 *       password is pass
 * The display will then prompt the user if the login credidential were
correct.
 * The admin has the ability to test the software while the user can see
operations.
 *
 * The display will then prompt the admin to input values for the 3 main
functionalities.
 * Then the software will report the respective warnings if the same data
was recieved from the sensors.
 *
 * To run Display.java, simply run Display.java
 */
public class Display extends JFrame implements ActionListener {
    // Create the login GUI
    Container container = getContentPane();
    JLabel userLabel = new JLabel("USERNAME");
    JLabel passwordLabel = new JLabel("PASSWORD");
    JTextField userTextField = new JTextField();
    JPasswordField passwordField = new JPasswordField();
    JButton loginButton = new JButton("LOGIN");
    JCheckBox showPassword = new JCheckBox("Show Password");
    // GUI for manual user input
    private JFrame frame;
    private JPanel pane;
    private JTextField objSpeedField;
    private JTextField objDistField;
    private JTextField gateDistField;
    private JTextField gateStatusField;
    private JTextField wheelRPMField;
    private JTextField gpsSpeedField;
    private int objSpeed = 0;
    private int objDist = 0;
    private double gateDist = 0;
    private int gateStat = 0; // true 1 means its open, false 0 means its closed and safe to cross
    private int wheelRPM = 0;
    private int gpsSpeed = 0;

    Display() {
        setLayoutManager();
        setLocationAndSize();
        addComponentsToContainer();
        addActionEvent();
    }

    public void setLayoutManager() {
        container.setLayout(null);
    }

    public void setLocationAndSize() {
        userLabel.setBounds(50, 150, 100, 30);
        passwordLabel.setBounds(50, 220, 100, 30);
        userTextField.setBounds(150, 150, 150, 30);
        passwordField.setBounds(150, 220, 150, 30);
        showPassword.setBounds(150, 250, 150, 30);
        loginButton.setBounds(100, 300, 100, 30);
    }

    public void addComponentsToContainer() {
        container.add(userLabel);
        container.add(passwordLabel);
        container.add(userTextField);
        container.add(passwordField);
        container.add(showPassword);
        container.add(loginButton);
    }

    public void addActionEvent() {
        loginButton.addActionListener(this);
        showPassword.addActionListener(this);
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        // checks to see if the correct login credientials were provided
        if (e.getSource() == loginButton) {
            String userText;
            String pwdText;
            userText = userTextField.getText();
            pwdText = passwordField.getText();
            if (userText.equalsIgnoreCase("admin") &&
pwdText.equalsIgnoreCase("pass")) {
                JOptionPane.showMessageDialog(this, "Login Successful");

                // if correct admin login, ask for manual user input

                pane = new JPanel();
                pane.setLayout(new GridLayout(0, 2, 2, 2));

                objSpeedField= new JTextField(5);;
                objDistField= new JTextField(5);;
                gateDistField= new JTextField(5);;
                gateStatusField= new JTextField(5);;
                wheelRPMField= new JTextField(5);;
                gpsSpeedField= new JTextField(5);;
                pane.add(new JLabel("what is the obstruction speed 
(mph)?"));
                pane.add(objSpeedField);

                pane.add(new JLabel("how far is the obstruction (mile)?"));
                pane.add(objDistField);

                pane.add(new JLabel("how far is the gate (mile)?"));
                pane.add(gateDistField);

                pane.add(new JLabel("What is the status of the gate (int: 
open=1, close=0)?"));
                pane.add(gateStatusField);

                pane.add(new JLabel("What is the wheel rpm (int)?"));
                pane.add( wheelRPMField);

                pane.add(new JLabel("What is the speed as given by the GPS 
(mph)?"));
                pane.add(gpsSpeedField);

                int option = JOptionPane.showConfirmDialog(frame, pane, 
"Please fill all the fields and press yes when completed", 
JOptionPane.YES_NO_OPTION, JOptionPane.INFORMATION_MESSAGE);

                if (option == JOptionPane.YES_OPTION) {
                    String objSpeedInput = objSpeedField.getText();
                    String objDistInput = objDistField.getText();
                    String gateDistInput = gateDistField.getText();
                    String gateStatusInput = gateStatusField.getText();
                    String wheelRPMInput = wheelRPMField.getText();
                    String gpsSpeedInput = gpsSpeedField.getText();

                    try {
                        objSpeed = Integer.parseInt(objSpeedInput);
                        objDist = Integer.parseInt(objDistInput);
                        gateDist = Double.parseDouble(gateDistInput);
                        gateStat = Integer.parseInt(gateStatusInput);
                        wheelRPM = Integer.parseInt(wheelRPMInput);
                        gpsSpeed = Integer.parseInt(gpsSpeedInput);
                    } catch (NumberFormatException nfe) {
                        nfe.printStackTrace();
                    }

                    pane = new JPanel();
                    pane.setLayout(new BoxLayout(pane,
BoxLayout.PAGE_AXIS));

                    // Warning/recommendation generation

                    if((gateStat == 1) && (gateDist <= 2)) {
                        if(Math.abs(gateDist) == 1) {
                            JOptionPane.showMessageDialog(null ,"<html><div
color=red>"+"Open Gate! Break immediately and blow horn for 15 seconds.",
"Warning" , JOptionPane.ERROR_MESSAGE);
                        } else if(Math.abs(gateDist) == 0) {
                            JOptionPane.showMessageDialog(null ,"<html><div
color=red>"+"Open Gate! Break immediately and blow horn for 5 seconds.",
"Warning" , JOptionPane.ERROR_MESSAGE);
                        } else if(Math.abs(gateDist) < 1) {
                            JOptionPane.showMessageDialog(null ,"<html><div
color=red>"+"Open Gate! Break immediately.", "Warning" ,
JOptionPane.ERROR_MESSAGE);
                        } else {
                            JOptionPane.showMessageDialog(null ,"<html><div
color=orange>"+"Open Gate! Slow down.", "Warning" ,
JOptionPane.ERROR_MESSAGE);
                        }
                    }

                    // wheel circum = 1
                    int rpm_to_speed = (int) (wheelRPM * 1 * Math.PI * 60 /
63360);
                    if(Math.abs(rpm_to_speed - gpsSpeed) >= 5) {
                        if (Math.abs(rpm_to_speed - gpsSpeed) >= 10) {
                            JOptionPane.showMessageDialog(null ,"<html><div
color=red>"+"Major Slippage! Break immediately.", "Warning" ,
JOptionPane.ERROR_MESSAGE);
                        } else {
                            JOptionPane.showMessageDialog(null ,"<html><div
color=orange>"+"Minor Slippage! Slow down.", "Warning" ,
JOptionPane.ERROR_MESSAGE);
                        }
                    }

                    if(objDist <= 2) {
                        if(objDist <= 1) {
                            if(objSpeed != 0) {
                                JOptionPane.showMessageDialog(null
,"<html><div color=red>"+"Moving Object! Break immediately.", "Warning" ,
JOptionPane.ERROR_MESSAGE);
                            } else {
                                JOptionPane.showMessageDialog(null
,"<html><div color=red>"+"Stationary Object! Break immediately.", "Warning"
, JOptionPane.ERROR_MESSAGE);
                            }
                        } else {
                            if(objSpeed != 0) {
                                JOptionPane.showMessageDialog(null
,"<html><div color=orange>"+"Moving Object! Slow down.", "Warning" ,
JOptionPane.ERROR_MESSAGE);
                            } else {
                                JOptionPane.showMessageDialog(null
,"<html><div color=orange>"+"Stationary Object! Slow down.", "Warning" ,
JOptionPane.ERROR_MESSAGE);
                            }
                        }
                    }

                    // else {
                    //     JOptionPane.showMessageDialog(this ,"Normal
operations");
                    // }
                    if ((gateStat != 1) && (Math.abs(rpm_to_speed -
gpsSpeed) < 5) && (objDist > 2) ) {
                        if (gateDist == 0) {
                            JOptionPane.showMessageDialog(null ,"<html><div
color=red>"+"Blow horn for 5 seconds.", "Warning" ,
JOptionPane.ERROR_MESSAGE);
```

```
                        } else if (gateDist == 1) {
                            JOptionPane.showMessageDialog(null ,"<html><div
color=red>"+"Blow horn for 15 seconds.", "Warning" ,
JOptionPane.ERROR_MESSAGE);
                        } else {
                            JOptionPane.showMessageDialog(this ,"Normal
operations");
                        }
                    }
                    return;

                    // JOptionPane.showMessageDialog(frame, pane);
                }
            } if (userText.equalsIgnoreCase("user") &&
pwdText.equalsIgnoreCase("pass")) {
                JOptionPane.showMessageDialog(this, "Login Successful");

                // if correct user login, display operations in terminal

                Simulations sim = new Simulations();
                while(true){
                    displayInfo(sim.runSimulation());
                    try {
                        Thread.sleep(1000);
                    } catch (InterruptedException e1) {
                        // TODO Auto-generated catch block
                        e1.printStackTrace();
                    }
                    // change ok to next/update
                }
            } else {
                JOptionPane.showMessageDialog(this, "Invalid Username or
Password");
            }
        }
        if (e.getSource() == showPassword) {
            if (showPassword.isSelected()) {
                passwordField.setEchoChar((char) 0);
            } else {
                passwordField.setEchoChar('*');
            }
        }
    }

    public static void displayInfo(double[] info) {
        JOptionPane.showMessageDialog(null, "Wheel Slippage: "+
slippageDetection(info[0], info[1]) + '\n' +
                                            "Gate Status: "+
gateDetection(info[2], info[3]) + '\n' +
                                            "Obstruction: "+
objDetection(info[4], info[5]));
    }
    public static void writeToLog(String text) {
        Time time = new Time(System.currentTimeMillis());
        try {
            FileWriter fw = new FileWriter("log.txt", true);
            // will append to end of already existing file in repository
            fw.write(time.toString() + ": " + text);
            fw.close();
        } catch (IOException e) {
            System.out.println("An error occurred while logging.");
            e.printStackTrace();
        }
    }

    // the functions named Xdetection preform the logic to test if the rng
data would generate warnings/recommendations

    private static String slippageDetection(double rpm, double speed) {
        // in this case circum is 1m
        // vehicle speed = wheel rpm * wheel diameter * pi * 60 / 63360
         String x = new String();
         double rpm_to_speed = rpm * 1 * Math.PI * 60 / 63360;
         if (Math.abs(rpm_to_speed - speed) >= 5) {
            if (Math.abs(rpm_to_speed - speed) >= 10) {
                x = "ALERT!  Major slippage. Brake immediately.";
                writeToLog(x);
            } else {
                x = "ALERT!  Minor slippage. Slow down.";
                writeToLog(x);
```

```
       }
    } else {
        x = "Normal operation";
    }
    return x;
}

private static String gateDetection(double gateStat, double gateDist) {
   String x = new String();
   if ((gateStat == 1) && (Math.abs(gateDist) <= 3)) {
       if(Math.abs(gateDist) <= 2) {
           x = "ALERT!   Open Gate. Brake immediately.";
           writeToLog(x);
       } else {
           x = "ALERT!   Open Gate. Slow down.";
           writeToLog(x);
       }
   } else {
       x = "Normal operation";
   }
   return x;
}

private static String objDetection(double objDist, double objSpeed) {
   String x = new String();
   if(Math.abs(objDist) <= 2) {
       if(Math.abs(objDist) <= 1) {
           if(objSpeed != 0) {
               x = "ALERT!   Moving Object. Brake immediately.";
               writeToLog(x);
           } else {
               x = "ALERT!   Stationary Object. Brake immediately.";
               writeToLog(x);
           }
       } else {
           if(objSpeed != 0) {
               x = "ALERT!   Moving Object. Slow down.";
               writeToLog(x);
           } else {
               x = "ALERT!   Stationary Object. Slow down.";
```

```java
                    writeToLog(x);
                }
            }
        } else {
            x = "Normal operation";
        }
        return x;
    }

    public static void main(String[] a) {
        Display frame = new Display();
        frame.setTitle("Login Form");
        frame.setVisible(true);
        frame.setBounds(10, 10, 370, 600);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setResizable(false);
    }
}
```

```java
import java.lang.Math;
import java.io.FileWriter;
import java.io.IOException;
import java.sql.Time;

/**
 * Main.java is similar to Display.java except that it runs
 * on simulated data instead of manual user input data. The
 * warning/recommendation generation are reported to the
 * terminal. To run the simulation, simply run Main.java
 */
public class Main {
    /**
     * Formats the text outputted to the terminal
     * Simply provides the following states:
     *      Normal operations
     *      Yellow warning
     *      Red warning
     * for each of the three functionalities
     */
    public static void displayInfo(double[] info) {
        /**
        1. login
        2. object
            - speed determins stationary/moving
            - distance 2mile radius
            - [object: speed, distance]
        3. gate
            - distance 2mile radius
            - status
            - [gate: status, distance]
        4. slippage
            - ADD calculate rpm using tachometer
            - [slip: difference threshold reacheddisplay warning]
        */

        System.out.println();
        System.out.println(java.util.Calendar.getInstance().getTime());
        System.out.println("Wheel Slippage: "+ slippageDetection(info[0],
info[1])); // rpm and gps speed
        System.out.println("Gate Status: "+ gateDetection(info[2],
info[3])); // gateStat and gateDist
        System.out.println("Obstruction: "+ objDetection(info[4], info[5]));
//objDist and objSpeed
        System.out.println();
    }

    public static void writeToLog(String text) {
        Time time = new Time(System.currentTimeMillis());
        try {
            FileWriter fw = new FileWriter("log.txt", true);
            // will append to end of already existing file in repository
            fw.write(time.toString() + ": " + text);
            fw.close();
        } catch (IOException e) {
            System.out.println("An error occurred while logging.");
            e.printStackTrace();
        }
    }
    // the functions named Xdetection preform the logic to test if the rng
data would generate warnings/recommendations

    private static String slippageDetection(double rpm, double speed) {
        // in this case circum is 1m
        // vehicle speed = wheel rpm * wheel diameter * pi * 60 / 63360
         String x = new String();
         char ch=27;
         double rpm_to_speed = rpm * 1 * Math.PI * 60 / 63360;
         if (Math.abs(rpm_to_speed - speed) >= 5) {
            if (Math.abs(rpm_to_speed - speed) >= 10) {
                x = ch+"[31mMajor slippage. Brake immediately."+ch+"[0m";
                writeToLog("Major slippage. Brake immediately.");
            } else {
                x = ch+"[33mMinor slippage. Slow down."+ch+"[0m";
                writeToLog("Minor slippage. Slow down.");
            }
         } else {
             x = "Normal operation";
         }
         return x;
     }

     private static String gateDetection(double gateStat, double gateDist) {
        String x = new String();
        char ch=27;
        if ((gateStat == 1) && (Math.abs(gateDist) <= 2)) {
            if(Math.abs(gateDist) == 1) {
                x = ch+"[31mOpen Gate. Brake immediately and Blow horn for
15 seconds."+ch+"[0m";
                writeToLog("Open Gate. Brake immediately and Blow horn for
15 seconds.");
            } else if(Math.abs(gateDist) == 0) {
                x = ch+"[31mOpen Gate. Brake immediately and Blow horn for 5
seconds."+ch+"[0m";
                writeToLog("Open Gate. Brake immediately and Blow horn for 5
seconds.");
            } else if(Math.abs(gateDist) < 1) {
                x = ch+"[31mOpen Gate. Brake immediately."+ch+"[0m";
                writeToLog("Open Gate. Brake immediately.");
            } else {
                x = ch+"[33mOpen Gate. Slow down."+ch+"[0m";
                writeToLog("Open Gate. Slow down.");
            }
        } else {
            x = "Normal operation";
        }
        return x;
     }

     private static String objDetection(double objDist, double objSpeed) {
        String x = new String();
        char ch=27;
        if(Math.abs(objDist) <= 2) {
            if(Math.abs(objDist) <= 1) {
                if(objSpeed != 0) {
                    x = ch+"[31mMoving Object. Brake immediately."+ch+"[0m";
                    writeToLog("Moving Object. Brake immediately.");
                } else {
                    x = ch+"[31mStationary Object. Brake
immediately."+ch+"[0m";
                    writeToLog("Stationary Object. Brake immediately.");
                }
            } else {
                if(objSpeed != 0) {
                    x = ch+"[33mMoving Object. Slow down."+ch+"[0m";
                    writeToLog("Moving Object. Slow down.");
                } else {
                    x = ch+"[33mStationary Object. Slow down."+ch+"[0m";
                    writeToLog("Stationary Object. Slow down.");
                }
            }
        } else {
            x = "Normal operation";
        }
        return x;
     }

    // run the simulation
    public static void main(String[] args) throws InterruptedException {
        Simulations sim = new Simulations();
        while(true){
            displayInfo(sim.runSimulation());
            Thread.sleep(1000);
        }
    }
}
```

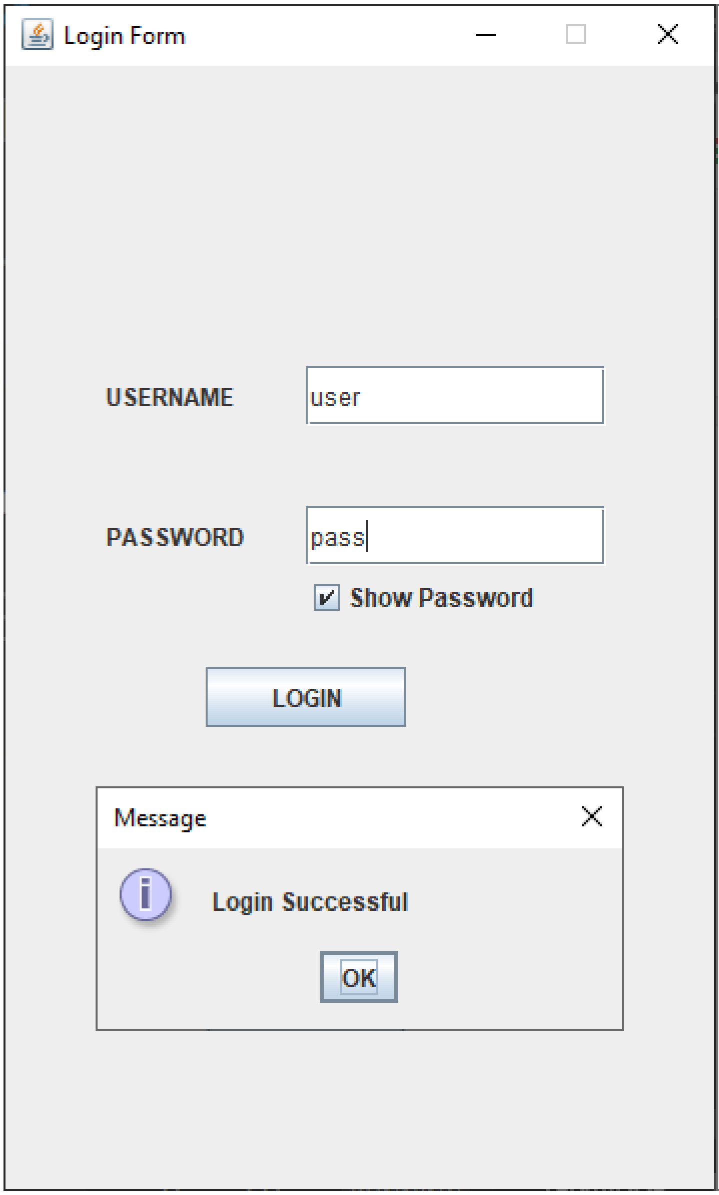
```
import java.util.Random;
/**
 * Simulation.java is the method that assigns rng values
 * to each input every 1 second
 */

public class Simulations {
    private static final int MAX_rpm = 20000; // converts to 60mph
    private static final double SAFETY = 3;
    private static final double INTERVAL = 1.0;

    private Random rng = new Random();

    private double rpm;
    private double speed;
    private int gateStat;
    private double gateDist;
    private double objDist;
    private double objSpeed;
    private boolean station;

    // assigns the initial values of the inputs
    public Simulations() {
        rpm = 0.0;
        speed = 0;
        gateStat = 1;
        gateDist = (rng.nextInt(51-0) + 0);
        objDist = (rng.nextInt(51-0)+0);
        objSpeed = 0;
    }
    public double[] runSimulation() {
        if(rpm <= 0) { // start engine up and running : speeding up
            speed = 5*INTERVAL;
            rpm+=speed;
            station =false;
        } else if (station || objDist < SAFETY || rpm > MAX_rpm) { // at
station or obj nearby: slowing down
            rpm += -3.0*INTERVAL;
            if(rpm<0){
                rpm = 0;
            }
            objDist +=(((rng.nextInt(2)+1)/10.0)*INTERVAL);
        }
        // else if(rpm > MIN_rpm){
        //     speed = 1.5;
        //     rpm+= speed;
        //     objSpeed += (((rng.nextInt(2))/10.0)*INTERVAL);
        // }
        else {
            speed = (rpm<MAX_rpm) ? (1.5*INTERVAL): 0;
            rpm += speed;
            objSpeed += ((((rng.nextInt(5))-2)/10.0)*INTERVAL);
            if(rng.nextInt(10)==9 && objSpeed>=SAFETY){
                station = true;
            }
        }
        // the following four inputs are independent of speed, rpm, station
        gateStat =  rng.nextInt(2);
        gateDist =  rng.nextInt(101-0)+0;
        objDist = (rng.nextInt(51-0)+0);
        objSpeed = (rng.nextInt(61-0)+0);

        // format and return results
        double[] result = {rpm, speed, gateStat, gateDist, objDist,
objSpeed};
        return result;
    }
}
```

```
/**
 * Sensor.java is the sensor class that hold/sets/gets data that
 * relates to the respective sensor
 */
public class Sensors {
    // individual sensor instances
    Lidar           lidar           = new Lidar();
    GPS             gps             = new GPS();
    Tachometer      tachometer      = new Tachometer();
    Radar           radar           = new Radar();

    public class Lidar {
        private int distance;
        private int speed;

        public Lidar() {
            this.distance = 0;
            this.speed = 0;
        }
        public void LidarDist(int distance){
            this.distance = distance;
            this.speed = 0;
        }
        public void LidarSpeed(int speed){
            this.distance = 0;
            this.speed = speed;
        }
        public Lidar(int distance, int speed){
            this.distance = distance;
            this.speed = speed;
        }
        public void setDistance(int distance){
            this.distance = distance;
        }
        public void setSpeed(int speed){
            this.speed = speed;
        }
        public int getDistance(){
            return distance;
        }
        public int getSpeed(){
            return speed;
        }
    }

    public class GPS {
        private int speed;

        public GPS(){
            this.speed = 0;
        }
        public GPS(int speed){
            this.speed = speed;
        }
        public void setSpeed(int speed) {
            this.speed = speed;
        }
        public int getSpeed() {
            return speed;
        }
    }

    public class Tachometer{
        private int rpm;

        public Tachometer(){
            this.rpm = 0;
        }
        public Tachometer(int rpm) {
            this.rpm = rpm;
        }
        public void setTachometer(int rpm) {
            this.rpm = rpm;
        }
        public int getTachometer(){
            return rpm;
        }
    }
    public class Radar {
        private int distance;
        private int gateStat;

        public Radar() {
            this.distance = 0;
            this.gateStat = 0;
        }
        public void RadarDist(int distance){
            this.distance = distance;
            this.gateStat = 0;
        }
        public void RadarGate(int gateStat){
            this.distance = 0;
            this.gateStat = gateStat;
        }
        public Radar(int distance, int gateStat){
            this.distance = distance;
            this.gateStat = gateStat;
        }
        public void setDistance(int distance){
            this.distance = distance;
        }
        public void setGateStat(int gateStat){
            this.gateStat = gateStat;
        }
        public int getDistance(){
            return distance;
        }
        public int getGateStat(){
            return gateStat;
        }
    }


////////////////////////////////////////////////////////////////////////////
///////////////////////////////////////////////

    /**
     * @return returns dist from lidar
     */
    public int getLidarDist() {
        return lidar.getDistance();
    }

    /**
     * @return returns speed from lidar
     */
    public int getLidarSpeed() {
        return lidar.getSpeed();
    }

    /**
     * @return returns speed from GPS
     */
    public int getGPSSpeed() {
        return gps.getSpeed();
    }

    /**
     * @return returns wheel speed from tachometer
     */
    public double getRPM() {
        return tachometer.getTachometer();
    }

    /**
     * @return returns dist from radar
     */
    public int getRadarDist() {
        return radar.getDistance();
    }

    /**
     * @return returns gate status from radar
     */
    public int getRadarGateStat() {
        return radar.getGateStat();
    }
}
```

7. Test Cases

7.1 Scenario-Based Testing

Use case 1: Initialize IoT HTR

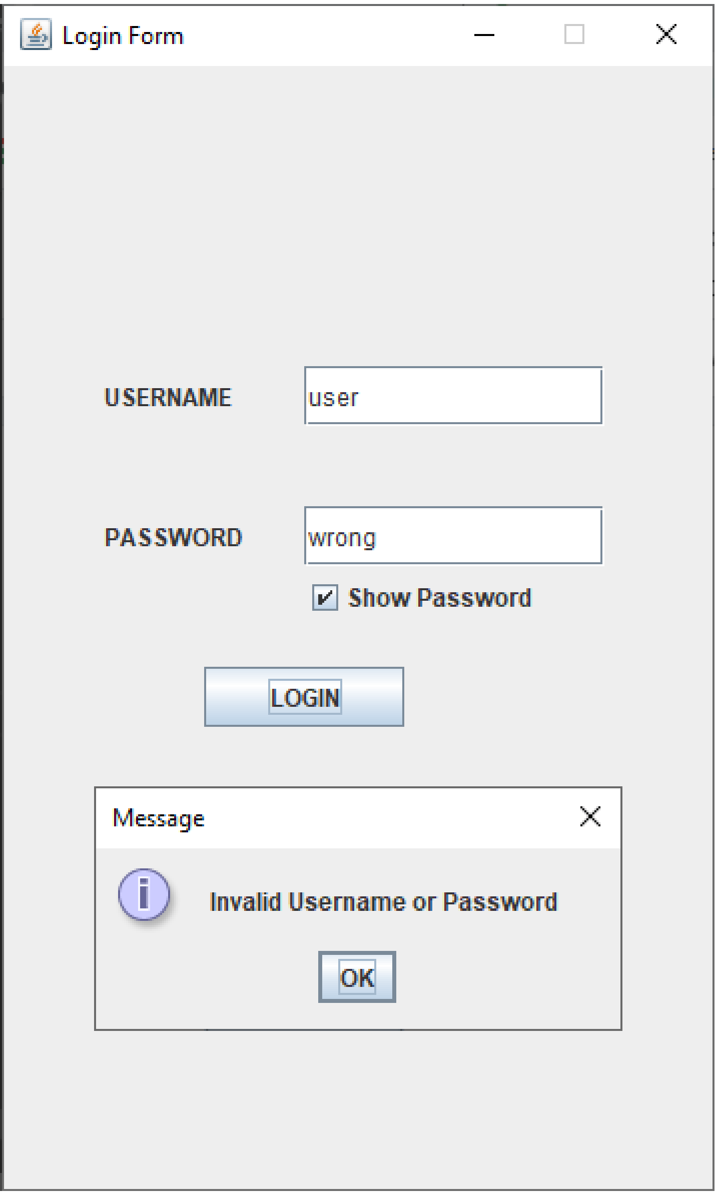


USERNAME user

PASSWORD pass

☑ Show Password

Login Successful



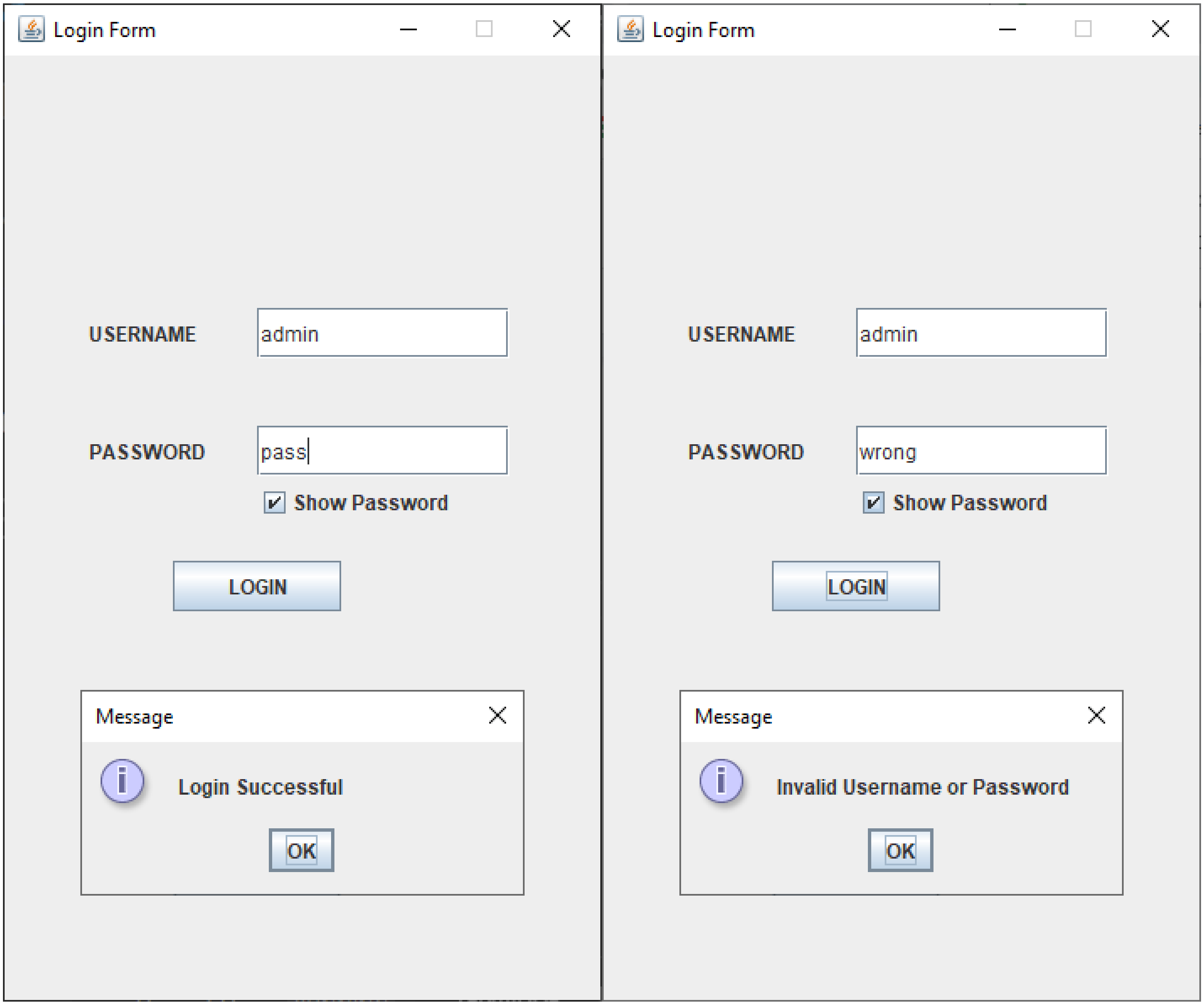
USERNAME user

PASSWORD wrong

☑ Show Password

Invalid Username or Password

Use case 2: Initialize IoT HTR with admin privileges



Login Form

USERNAME admin

PASSWORD pass

☑ Show Password

Message

Login Successful

Login Form

USERNAME admin

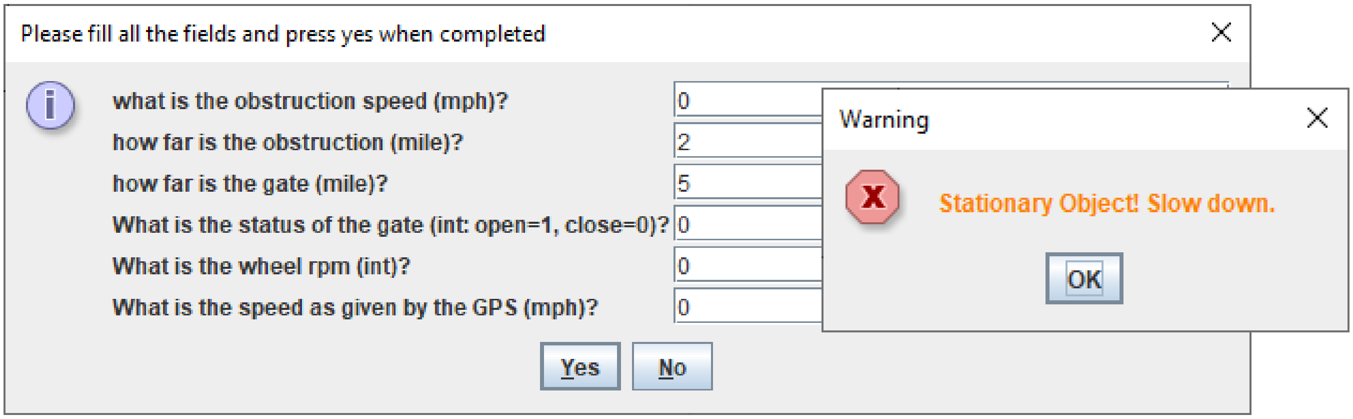
PASSWORD wrong

☑ Show Password

Message

Invalid Username or Password

Use case 3: Obstruction detection

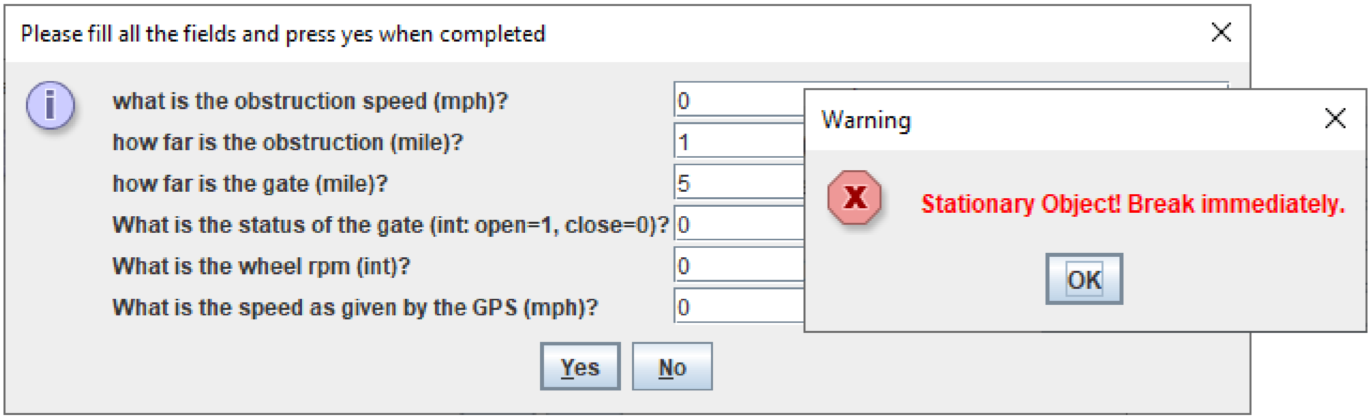


Please fill all the fields and press yes when completed

what is the obstruction speed (mph)?	0
how far is the obstruction (mile)?	2
how far is the gate (mile)?	5
What is the status of the gate (int: open=1, close=0)?	0
What is the wheel rpm (int)?	0
What is the speed as given by the GPS (mph)?	0

Warning

Stationary Object! Slow down.

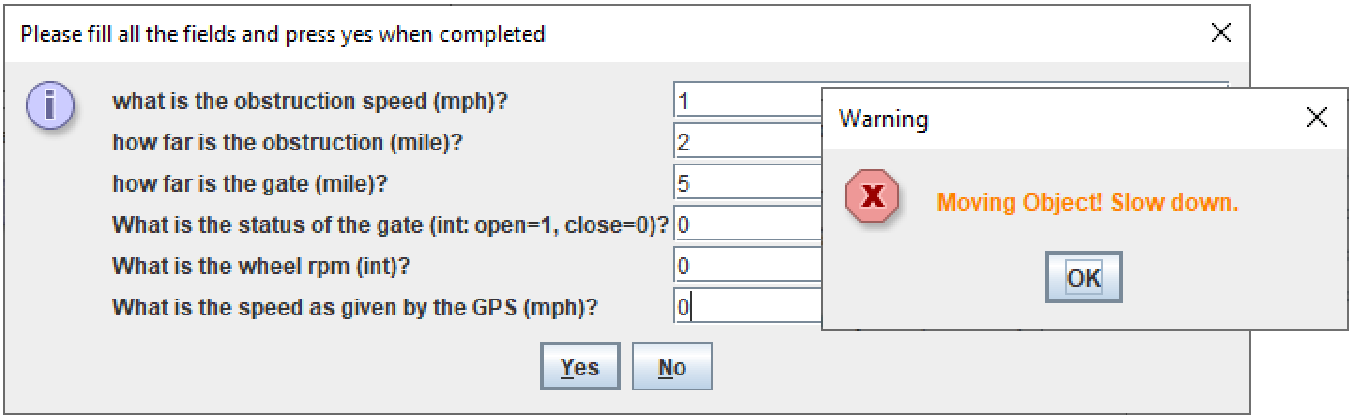


Please fill all the fields and press yes when completed

what is the obstruction speed (mph)?	0
how far is the obstruction (mile)?	1
how far is the gate (mile)?	5
What is the status of the gate (int: open=1, close=0)?	0
What is the wheel rpm (int)?	0
What is the speed as given by the GPS (mph)?	0

Warning

Stationary Object! Break immediately.

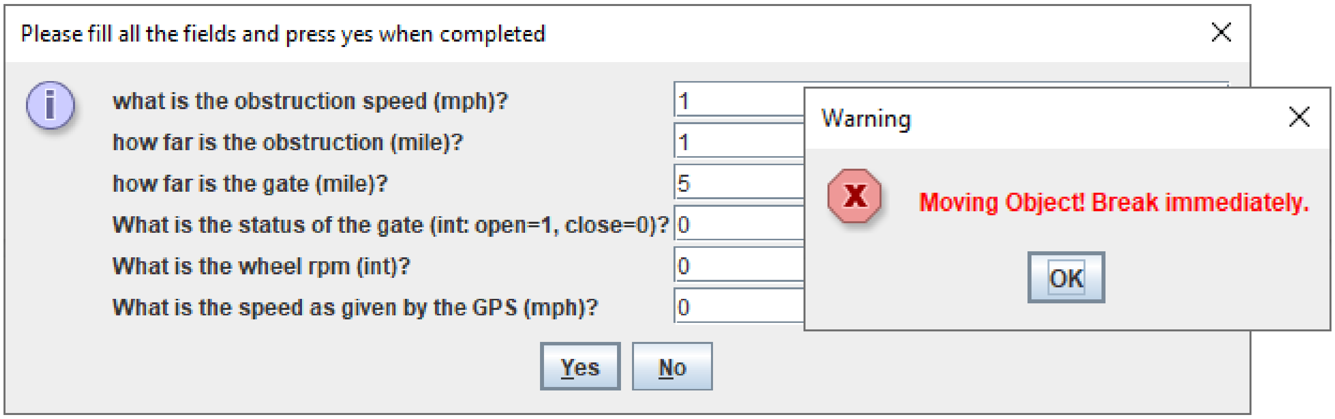


Please fill all the fields and press yes when completed

what is the obstruction speed (mph)?	1
how far is the obstruction (mile)?	2
how far is the gate (mile)?	5
What is the status of the gate (int: open=1, close=0)?	0
What is the wheel rpm (int)?	0
What is the speed as given by the GPS (mph)?	0

Warning

Moving Object! Slow down.



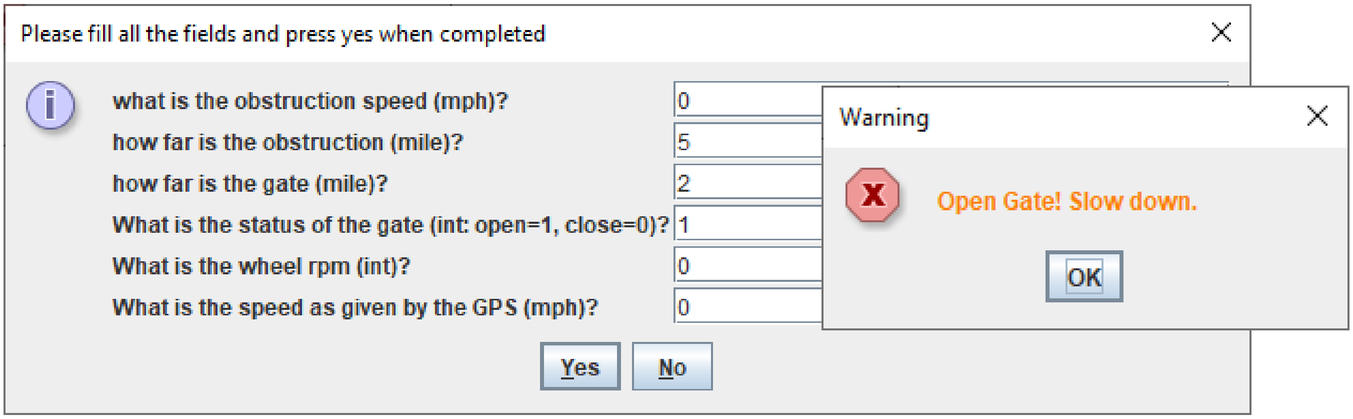
Please fill all the fields and press yes when completed

what is the obstruction speed (mph)?	1
how far is the obstruction (mile)?	1
how far is the gate (mile)?	5
What is the status of the gate (int: open=1, close=0)?	0
What is the wheel rpm (int)?	0
What is the speed as given by the GPS (mph)?	0

Warning

Moving Object! Break immediately.

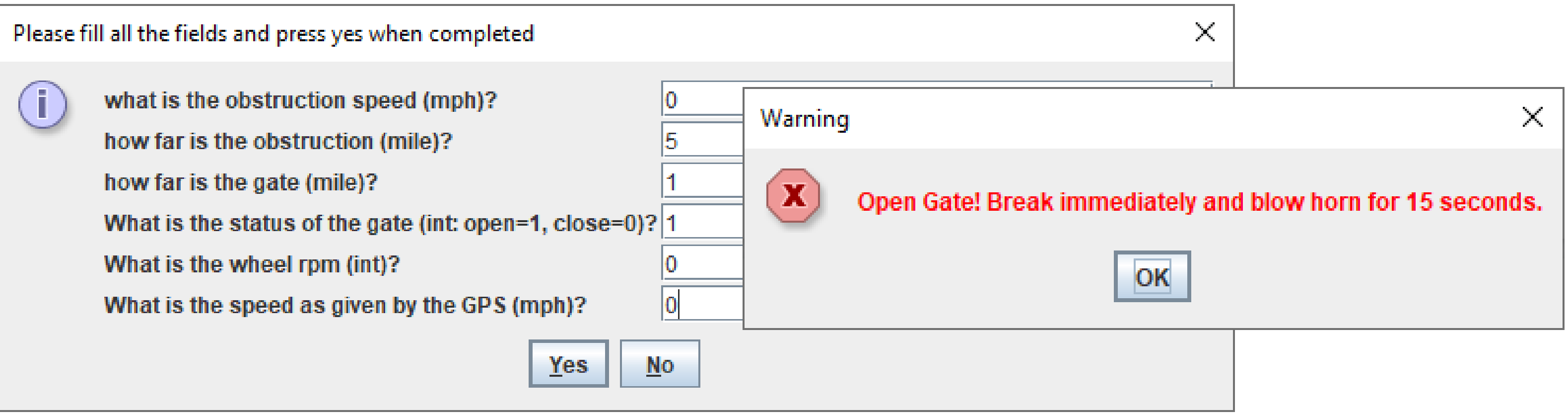
Use case 4: Gate Crossing Detection


Please fill all the fields and press yes when completed

what is the obstruction speed (mph)? 0
how far is the obstruction (mile)? 5
how far is the gate (mile)? 2
What is the status of the gate (int: open=1, close=0)? 1
What is the wheel rpm (int)? 0
What is the speed as given by the GPS (mph)? 0

Warning

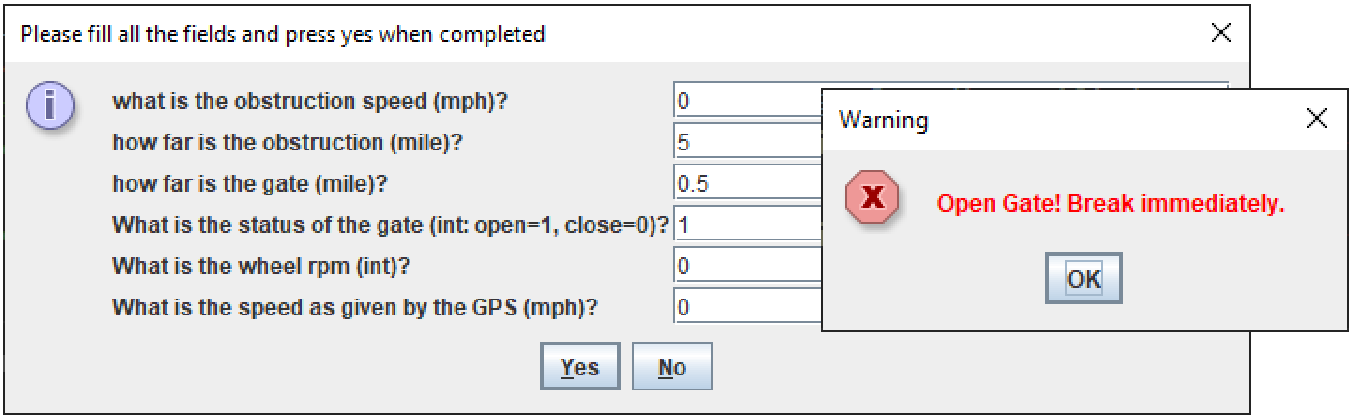
Open Gate! Slow down.


Please fill all the fields and press yes when completed

what is the obstruction speed (mph)? 0
how far is the obstruction (mile)? 5
how far is the gate (mile)? 1
What is the status of the gate (int: open=1, close=0)? 1
What is the wheel rpm (int)? 0
What is the speed as given by the GPS (mph)? 0

Warning

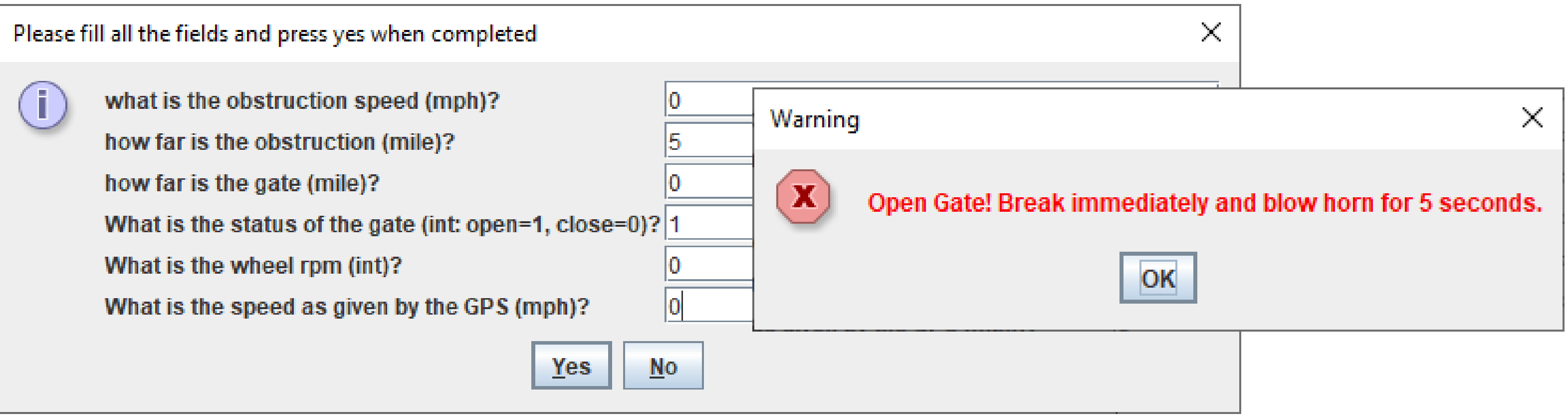
Open Gate! Break immediately and blow horn for 15 seconds.


Please fill all the fields and press yes when completed

what is the obstruction speed (mph)? 0
how far is the obstruction (mile)? 5
how far is the gate (mile)? 0.5
What is the status of the gate (int: open=1, close=0)? 1
What is the wheel rpm (int)? 0
What is the speed as given by the GPS (mph)? 0

Warning

Open Gate! Break immediately.

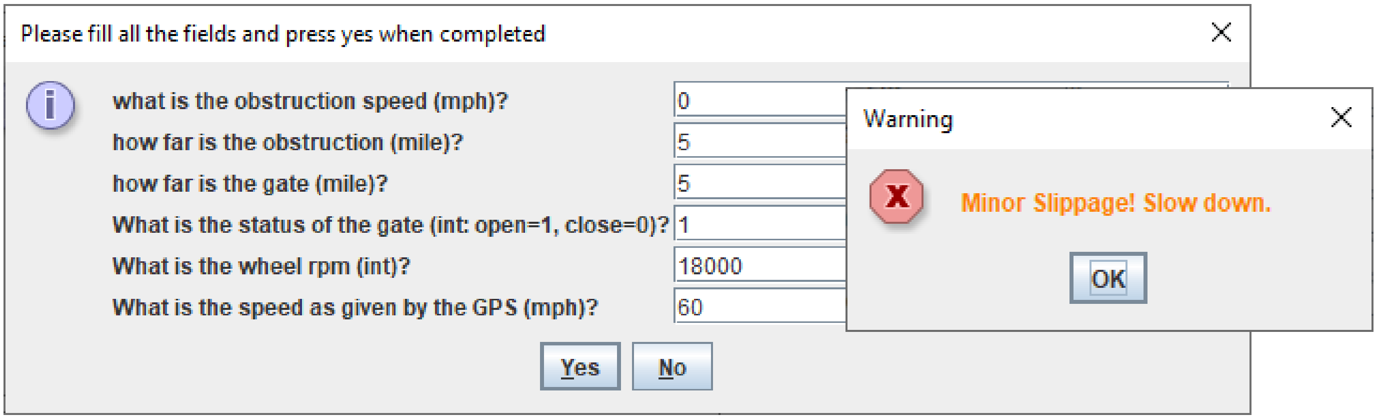

Please fill all the fields and press yes when completed

what is the obstruction speed (mph)? 0
how far is the obstruction (mile)? 5
how far is the gate (mile)? 0
What is the status of the gate (int: open=1, close=0)? 1
What is the wheel rpm (int)? 0
What is the speed as given by the GPS (mph)? 0

Warning

Open Gate! Break immediately and blow horn for 5 seconds.

Use case 5: Wheel Slippage Detection

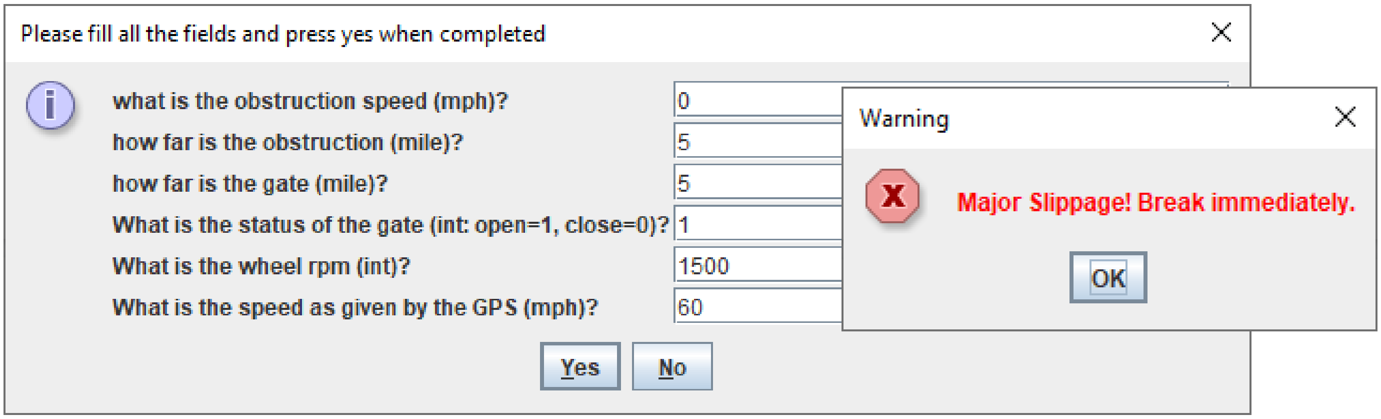


Please fill all the fields and press yes when completed

what is the obstruction speed (mph)?	0
how far is the obstruction (mile)?	5
how far is the gate (mile)?	5
What is the status of the gate (int: open=1, close=0)?	1
What is the wheel rpm (int)?	18000
What is the speed as given by the GPS (mph)?	60

Warning

Minor Slippage! Slow down.



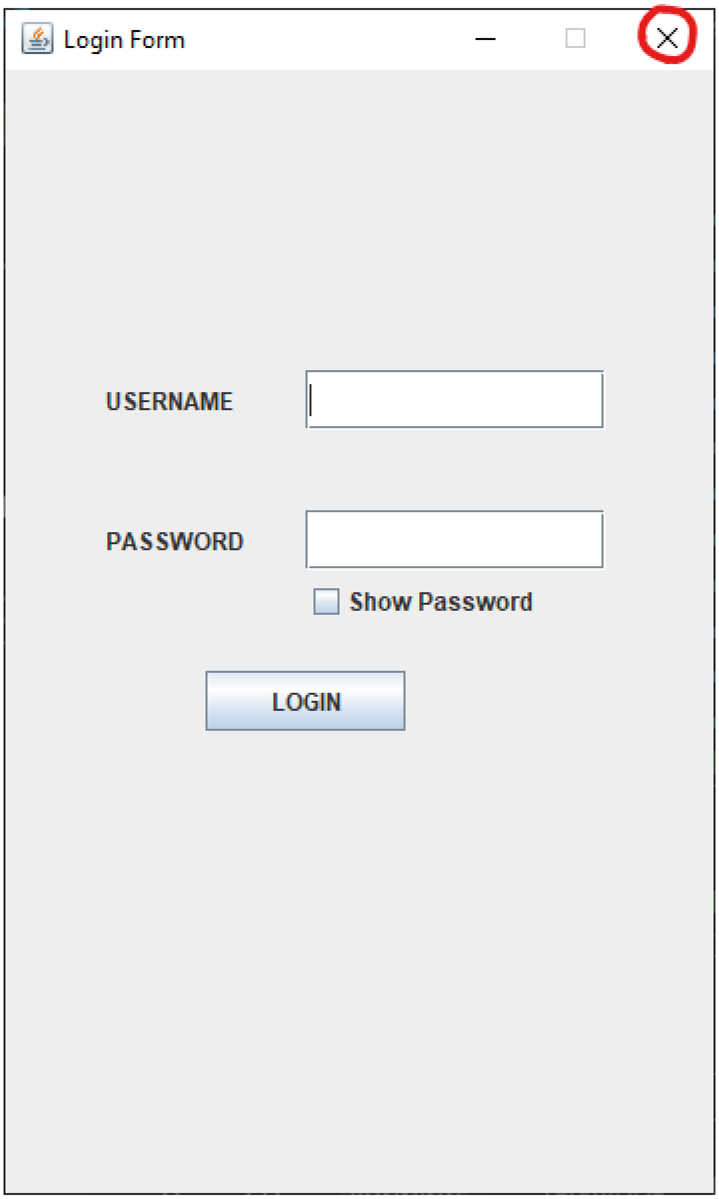
Please fill all the fields and press yes when completed

what is the obstruction speed (mph)?	0
how far is the obstruction (mile)?	5
how far is the gate (mile)?	5
What is the status of the gate (int: open=1, close=0)?	1
What is the wheel rpm (int)?	1500
What is the speed as given by the GPS (mph)?	60

Warning

Major Slippage! Break immediately.

Use case 6: Uninitialize IoT HTR



7.2 Validation Testing

7.2.1 Non-Functional Requirements

7.2.1.1 Security

R-1: [Success] IoT HTR shall only be accessed via user ID and password.
R-2: [Success] Operators shall only see the operations of the train on the IoT HTR Display .
R-3: [Success] Only Admins shall have access to raw data and logs.
R-4: [Success] Only Admins shall have access to system configurations.
R-5: [Not testable] IoT HTR network shall be secured by a LoRaWan Protocol.
R-6: [Not testable] IoT HTR shall encode all user-supplied output.

7.2.1.2 Performance

R-7: [Not testable] IoT HTR sensors shall process an event within 0.5 seconds.
R-8: [Not testable] TSNR shall send sensor data to IoT HTR within 0.5 seconds.
R-9: [Success] IoT HTR shall process an event within 0.5 seconds.
R-10: [Success] IoT HTR shall process 1,000 events per second without delay.
R-11: [Success] IoT HTR shall display processed events within 1.0 second.
R-12. [Success] The Log will log events every 1 second.

7.2.1.3 Reliability

R-13: [Not testable] IoT HTR shall operate with no failures 99.99% of the time.
R-14: [Success] IoT HTR shall alert the Operator how fast obstructions are moving with 95% accuracy.
R-15: [Success] The precision of distance-based calculations shall be at least 0.000001.
R-16: [Not testable] IoT HTR defect rate shall be less than 1 failure per 1000 hours of standard operation.
R-17: [Success] IoT HTR shall activate within 100 ms after the power button is pressed.
R-18: [Success] IoT HTR shall deactivate within 100 ms after the power button is pressed.
R-19: [Not testable] IoT HTR shall meet or exceed 99.9% uptime.
R-20: [Not testable] The network of sensors shall be operational at least 99.99% of the time.

7.2.1.4 Other Non-Functional Requirements

R-21: [Success] IoT HTR shall be equipped with a display.
R-22: [Success] IoT HTR shall be equipped with a power button.
R-23: [Success] Operators and admins shall be able to start and stop the IoT HTR via a power button.
R-24: [Success]The Operator shall be prompted with a login screen after pressing on the power button.
R-25: [Success] The Display shall show recommendations from the IoT HTR.
R-26: [Success] The Display will display a continue normal operations message if no issues are presented by all the sensors.
R-27: [Success] The Log continuously logs data and sends it to a single log file to keep track of all events.
R-28: [Success] IoT HTR shall have a clock to attach timestamps to the log file entries.

7.2.2 Functional Requirements

7.2.2.1 Standing Objects

R-29: [Not testable] There shall be two LiDAR sensors, one in the front and one in the back of the train.
R-30: [Not testable] The LiDAR sensors shall continuously take pictures to map the surroundings.
R-31: [Not testable] The LiDAR sensors shall provide the distance and speed of obstructions to the TSNR every 0.05 seconds.
R-32: [Not testable] IoT HTR shall receive obstruction data from the TSNR.
R-33: [Success] IoT HTR shall display a warning message if an obstruction is detected via the Display.
R-34: [Success] IoT HTR shall display that the object is stationary via the Display.
R-35: [Success] IoT HTR shall suggest to the Operator to slow down if there is an object within a 2.0 mile radius of the train via the Display.
R-36: [Success] IoT HTR shall suggest to the Operator to brake if there is an object within a 1.0 mile radius of the train via the Display.
R-37: [Success] IoT HTR shall display normal operations when there is no object within a 2.0 mile radius of the train via the Display.

7.2.2.2 Moving Objects

R-38: [Not testable] There shall be two LiDAR sensors, one in the front and one in the back of the train.
R-39: [Not testable] The LiDAR sensors shall continuously take pictures and map the surroundings.
R-40: [Not testable] The LiDAR sensors shall provide the distance and speed of obstructions to the TSNR every 0.05 seconds
R-41: [Not testable] IoT HTR shall receive obstruction data from the TSNR.
R-42: [Success] IoT HTR shall display a warning message if an obstruction is detected via the Display.
R-43: [Success] IoT HTR shall suggest to the Operator to slow down if there is an object within a 2.0 mile radius of the train via the Display.
R-44: [Success] IoT HTR shall suggest to the Operator to brake if there is an object within a 1.0 mile radius of the train via the Display.
R-45: [Success] IoT HTR shall display normal operations when there is no object within a 2.0 mile radius of the train via the Display.

7.2.2.3 Gate Crossings

A closed gate status means cars should stop at the railroad crossing as a train is approaching.
An opened gate means that cars can drive over the railroad crossing as no train is approaching.

R-46: [Not testable] There shall be one radar sensor located in the front of the train.
R-47: [Not testable] The Radar sensors shall continuously emit radio waves and checks the change in frequency after omission.
R-48: [Not testable] The Radar sensors shall provide the distance and status of a gate crossing to the TSNR every 0.05 seconds.
R-49: [Not testable] IoT HTR shall receive gate crossing data from the TSNR.
R-50: [Success] IoT HTR shall display a warning message if the gate is open via the Display.
R-51: [Success] IoT HTR shall suggest to the Operator to slow down if the gate is open within a

2.0-mile radius via the Display.

R-52: [Success] IoT HTR shall suggest to the Operator to brake if the gate is open within a 1.0-mile radius via the Display.

R-53: [Success] IoT HTR shall suggest to the Operator to honk the horn for 15 seconds when it is 1 mile away from the gate via the Display.

R-54: [Success] IoT HTR shall suggest to the Operator to honk the horn for 5 seconds when it reaches the gate via the Display.

R-55: [Success] IoT HTR shall display normal operations when the gate is closed via the Display.

7.2.2.4 Wheel Slippage

R-56: [Not testable] IoT HTR is equipped with four tachometers on the four outermost train wheels.

R-57: [Not testable] IoT HTR is equipped with one GPS.

R-58: [Not testable] The Tachometers shall continuously measure the rotation speed of the train wheel in rpm.

R-59: [Not testable] The GPS shall continuously send and receive signals to calculate the speed of the train.

R-60: [Not testable] The Tachometers shall provide the wheels' rpm to the TSNR every 0.05 seconds.

R-61:[Not testable] The GPS shall provide the speed of the HTR train to the TSNR every 0.01 seconds.

R-62: [Not testable] IoT HTR shall receive wheel slippage data from the TSNR.

R-63: [Success] IoT HTR shall calculate the speed of the HTR train from the tachometer data.

R-64: [Success] IoT HTR shall calculate the difference in the speeds from the GPS and tachometer.

R-65: [Success] IoT HTR shall display a warning message if the train's wheels are slipping via the Display.

R-66: [Success] IoT HTR shall suggest to the Operator to slow down if the train is minorly slipping (5mph < Δspeed < 10mph) via the Display.

R-67: [Success] IoT HTR shall suggest to the Operator to brake if the train is slipping severely (Δspeed > 10mph) via the Display.

R-68: [Success] IoT HTR shall display normal operations when slipping stops via the Display.

7.2.3 Hardware & Operating System

R-69: [Not testable] IoT HTR hardware shall be able to support at least 1,000 sensors.

R-70: [Not testable] IoT HTR shall support 5TB of data every day.

R-71: [Not testable] IoT HTR shall be able to import and export information obtained from operating.

R-72: [Not testable] IoT HTR shall be updated yearly with patches in between.

7.2.3.1 Operating System

R-73: [Not testable] IoT HTR shall be equipped with Microsoft Windows 10 IoT Enterprise

R-74: [Not testable] IoT HTR shall be equipped with Microsoft Windows 10, 64-bit

7.2.3.2 Processor

R-75: [Not testable] IoT HTR shall be equipped with ARM processor

7.2.3.3 RAM

R-76: [Not testable] IoT HTR shall be equipped with 2GB of RAM

7.2.3.4 Graphics Card

R-77: [Not testable] IoT HTR shall be equipped with a GPU that supports DirectX 9

7.2.3.5 Storage

R-78: [Not testable] IoT HTR shall be equipped with 32GB of storage

7.2.3.6 Sensors

R-79: [Not testable] IoT HTR shall be equipped with a Tachometer - measures the rpm of a wheel
R-80: [Not testable] IoT HTR shall be equipped with LiDAR sensors - maps surrounding environment
R-81: [Not testable] IoT HTR shall be equipped with Radar sensors - detect is the status of gate crossing
R-82: [Not testable] IoT HTR shall be equipped with a GPS - give global positioning