“We pledge our honor that we have abided by the Stevens Honor System.”

CS 347: SOFTWARE DEVELOPMENT PROCESS

INTERNET OF THINGS:

HUGS THE RAIL

The Little Engineers That Could

Jack Schneiderhan, Cindy Zhang, Hao Dian Li, Peter Rauscher. 02.16.2020

TABLE OF CONTENTS

1: INTRODUCTION

Locomotives used by Hugs the Rails should aim to get every package onboard as quickly and efficiently as possible so that the people and companies can get to their parcels in a timely manner. They should also be ensured that all systems are updated for their safety alongside the new protocols.

1.1 PROBLEM STATEMENT

Currently, the train system involved with the Hugs the Rail system is completely reliant on the internet and network connection it has. If a scenario occurred where the internet / network connection was lost, however, the trains would be lost with no idea on where to go or what to do incase of an emergency. This is precisely the problem at hand. The Hugs the Rail system will implement sensors into these trains, which function locally, not relying on a network connection with the outside world, so that the trains will function in a scenario where there is no possible way to connect to the internet.

1.2 PURPOSE

Internet of Things: Hugs The Rail is a project that will produce a product to railroad users and supporters to enhance the ride. These changes will include being able to make decisions locally in absence of cellular and wifi connectivity back the offices, being able to capture data from the locomotives and its surrounding environment, issuing an analytic engine on board to process as much information to safely guide passengers as well as the conductor on their destination, make decision that is passed on the locomotive control system, providing room for the operator to enter commands to receive status, and the ability to download the latest rules from the cloud into the engine. The Little Engineers That Could, Team Eight is a talented team filled with curiosity, structured by organized members, and constructed with knowledge to perform these tasks.

2: OVERVIEW

An important fact to note in regards to the Internet of Things / Hug the Rail project is that the trains involved depend on a wireless / cellular network to receive live data about the environment around them and their surrounding traffic. While we don't want the train to lose connection to the internet, there are sometimes unexpected problems that occur resulting in a loss of connectivity from the train to the rest of the world. When this happens, the trains need to have a fallback in order to operate safely in a local setting. To understand how the trains operate locally, it's important to understand and have knowledge of potential hazards the train could face.

2.1 PROCESS MODEL

Umbrella OR WaterFall OR Unified

2.2 TIMELINE

Event	Deadline
Template	02/16/2020
Choose Process Model	02/16/2020
Planning	02/25/2020
Modeling	02/28/2020
Construction / Testing / Prototypes (?)	03/29/2020
Deployment	03/30/2020
Software Increment	--

2.3 ROLES

We have a talented team of individuals, each with a background and drive to succeed in the computer science field. With our combined experiences and effort, our project will get off the ground and onto the rails with no problem.

Name	Role	Responsibilities
Cindy	Member / Developer	Complete Task, Communicate, Address Problems If Needed, Organize
Jack	Leader / Developer	Complete Task, Communicate, Initiate Motivation, Making sure we meet our deadlines.
Hao	Member / Developer	Complete Task, Communicate, Address Problems If Needed
Peter	Member / Developer	Complete Task, Communicate, Address Problems If Needed

2.4 PROBLEMS

Trains can face a number of unexpected problems when being used on railroads:

- Some of these problems have to do with the weather. Various weather conditions can cause trains to slow down or completely stop altogether. These weather conditions in particular include snow, rain, or ice on the track. While one would think an operator could see these conditions themselves, there is a slight possibility that it could be something like black ice, or the conductor could just be distracted. That's why it's important we have sensors to check for these conditions.
- Other hazardous conditions could be an object or animal on the rail, or even hanging above the rail, having a potential to fall onto the train / train track.
- High wind is also a very dangerous hazard, as it could cause the train's speed to alter from its normal trajectory.
- The last important hazard to realize is other trains themselves on the track. These are a *very* dangerous hazard, as if another train is not sensed on the rails, a horrible accident could occur, causing the train to derail and hurt other people involved in the ride.

Because of these hazards, it's important to have sensors that check for these problems.

2.5 EXTERNAL SENSORS

Outside of the locomotive are some unpredictable objects. That's why IoT can help us "see" these problems that the train could potentially run into. A locomotive has a cab also known as the driver's compartment. This part of the train is the control room. Establishing sensors and connecting them to the control room can help the engineer (train drive) to know what is going on. Having proximity sensors can detect motions between each train as well as if the train could approach some unknown object. Proximity sensors usually emit electromagnetic fields or beams of radiation such as infrared. Another sensor are level sensors, as the name in the title, level sensors are used to detect the level of substances including liquids, powders and granular material. This can help with detecting the surroundings and help what could possibly happen but easily avoidable. Gyroscope sensors measure the angular rate or velocity. This can stabilize the locomotive to ensure a safe ride to go fast but not too fast.

2.6 INTERNAL SENSORS

Inside of the locomotive is just as dangerous when it comes to unpredictability. Just as a computer has a BIOS when booting to ensure that the system is booting up properly to the preset settings, sensors of the locomotive can be implemented to do the same things. IoT can give the locomotive a temperature sensor to make sure nothing in the locomotive is overheating or if something is frozen they should be able to detect it. Locomotives are mainly diesel-electric, where the diesel engine is connected to the main generator which converts the engine's mechanical power to electric. It is important to make sure that you are not running low on fuel or anything other important valuables to ensure the train runs smoothly.

2.7 CONNECTING TO NETWORKS

After overviewing the variety of helpful sensors, how will they connect to the internet? One way is to have sensors connect to the Low-Power Wide Networks (LPWAN). They can help mitigate the damage and discord if the trains suddenly go offline. LPWAN are optimal

because they allow long-ranged communications at a low bit rate. The technology is low-power and low-cost, and since most LPWAN devices are powered by batteries, power outages cannot affect or sever the connection. LoRa (Long Range) is the proprietary of LPWAN. This technology can provide cellular-style communications with the devices that are connected to the network. In our case, data delivery is crucial, so sensors in the LoRa network can traverse below the Radio-Frequency noise floor (noise floor is the measure of the signal created from the sum of all the noise sources and unwanted signals) and transmit large data packets over long distances. We can depend on motion sensors, position sensors, weight sensors, and lidar sensors to relay information back to the control room. These sensors, if able to connect to less-demanding networks, can ensure the train's safety to the next station and allow the system to remain on schedule.

2.8 ONBOARD DECISION MAKING

All of the situational and environmental data collected by these sensors need some way to be processed into a more meaningful and digestible format for the operators. In the event of complete network or power loss, conductors should be able to rely solely on onboard computing and power to process this information. For example, a display in the operator's room - connected to its own power grid, delivering both data and energy to and from the sensors using an existing standard like Power over Ethernet (PoE) - could improve redundancy and provide graphical analytics and last-known train schedules. Most importantly, the system could suggest recommended courses of action. As an example, using the external temperature and humidity sensors to predict the slickness of the tracks, and onboard accelerometers to track the train's velocity, the system could recommend a safe stopping distance to operators. All of these decisions would need to be handled with computers and software running locally in the operator's rooms, but would serve as a list line of defense in the case of multiple system failure.

3: REQUIREMENTS

The end goals of IoT: Hugs the Rail is to deliver a high quality software system. To achieve this, we need to implement requirements.

3.1 FUNCTIONAL REQUIREMENTS

Functional Requirements are the primary way that a customer communicates with the development team. It keeps the project team in the right direction. These requirements specify a behavior or function or what the system should do. In IoT: Hugs the Rail, the functional requirements include:

3.1.1 TRANSACTIONS

Inevitably in a dynamic business environment, requirements will be subject to change with stakeholder expectations and development team capabilities. it is important to define a standard set of procedures for amending the requirements document moving forward and notifying the appropriate parties of changes:

R1. When one or many stakeholders decide a requirement needs updating, the line of communication for making such requests should come in the form of a standalone email address and should be actively monitored.

R2. When the development team has relevant information for the stakeholders, or a request has been fulfilled, they should notify all affected parties directly via their preferred email address.

R3. Only trusted members of the development team should be able to update this document, but all members should be able to read it. The team can use GitLab's group feature for this functionality.

R4. Any and all changes to this document should be tracked automatically using Git source control.

3.1.2 ADMINISTRATIVE FUNCTIONS

When necessary, conductors and engineers should have more complete control than the regular interface provides. Here are some things an administrator of the system would need:

R5. A dense interface with access to all system functions

R6. Terminal to onboard computer

R7. Control of the train's onboard power grid

R8. Physical access to IoT sensors

R9. Uptime logs, error reports, and crash analytics

3.1.3 EXTERNAL INTERFACES

One of the ways to manage a locomotive to the best of its ability is to create an interface that is:

R10. Simple and Compact User Interface.

R11. Easy to the Eye.

R12. Regularly updated with Software Patches.

The interface should be able to:

R13. Expand on the details in case the conductor becomes unavailable and an inexperienced person is needed to take charge.

R14. Prioritize the most important objectives

R15. Alert the conductor when the system is not in homeostatic condition.

3.1.4 HISTORICAL DATA

Historical Data helps the company see what issues come up and what solutions have worked previously. It also helps with any patterns on the routes. Historical Data should be main by:

R16. Station storage will record all stop details. (Problems, Time, and Changes)

R17. Each locomotive will store data on a local drive and then transfers data to the next station.

R18. Data should be uploaded onto the internet to inform customers of the reality of any situation.

3.1.6 STANDING OBJECTS ACTIONS

Detect standing objects on the path with distance and suggest action to the operator to brake or increase/decrease the speed.

R41. Radars or sensors should be able to detect 270 degrees in a fixed 3 meter radius in order to scan or to track crowds. Markers should be placed on the track to alert the technology if there is a disruption in the restricted area. If this is the case, the locomotive should be alerted at least 500 meters away to slow down or stop.

R42. The locomotive should have vibration sensors aboard to detect its displacement. If it passes a certain threshold, precautions should be taken to decelerate the train. This is extremely important on bridges.

3.1.7 MOVING OBJECTS ACTIONS

Detect moving objects and distance ahead or behind and their speed and suggest to the operator branking or changing speed.

R43. If any objects are moving towards the train with no signs of braking or slowing down from the object, then the locomotive interface should alert the operator to move away and increase speed away from the object.

R44. If any objects are moving towards the train with signs of braking or slowing down, then the locomotive interface should alert the operator to slow down.

R45. If a locomotive detects a moving object but the proximity was becoming far too close that could lead to an interception or crash, the locomotive should alert the operator to slow down and possibly a potential brake.

R46. If there is a locomotive behind coming closer to the locomotive, it should alert the operator to speed up to a reasonable pace.

R47. The locomotive should have sensors implemented in the front and in the back of the train to detect proximity of other objects.

3.1.8 GATE ACTIONS

Detect gate crossing open/closed, distance and suggest speed change or brake to stop the operator.

R48. If a detected gate crossing is closed within 100 feet, begin to slow the train down to a stop so the train will not crash into the barrier.

R49. If a gate in front of the train is detected to open, begin to speed up the train so it can start to travel at its normal speed again.

R50. While the train is stopped in front of a closed gate crossing, keep the train stopped. Do not start moving until the gate is detected to have opened once more.

R51. The train should only be moving at maximum speed at a distance of 150 feet from the gate (regardless of whether or not the gate is in front or behind, for the purpose of safety).

R52. If a gate is detected within 1 mile, the train will display a gate is approaching and alert with a horn.

R53. If the train is at the gate, the train will display an alert and honk horn.

3.1.9 WHEEL ACTIONS

Detect wheel slippage using GPS speed data and comparing it with wheel RPM and suggest to operator change speed or brake.

R52. If GPS-based speed data is greater than the calculated speed using the wheels RPM, alert the operator that there is wheel slippage and report the difference in projected and actual speed as a percentage.

R53. When wheel slippage is detected, automatically slow the train until the GPS speed data is equivalent to the speed calculated from the wheel's RPM.

R54. If wheel slippage is detected and the weather data suggests high moisture content in the air, alert the operator that the tracks are wet and suggest slowing.

R55. If still connected to the network, report to the control room at the next station that wheels are slipping and the train may be delayed or off schedule.

R56. The locomotive should have a sensor for every three wheels to check whether there is a dangerous amount of slippage.

3.2 NON-FUNCTIONAL REQUIREMENTS

3.2.1 HARDWARE USABILITY

R24. The sensor hardware should be durable and easy to implement because frequent maintenance is not ideal.

R25. It should be able to withstand temperatures lower than 0 degrees Celsius in the winter and temperatures above 40 degrees in the summer.

R26. Of course, the sensors should be covered or sheltered so that rodents and other animals cannot tamper with the technology.

R27. Since it is exposed to various weather conditions, the lens should be hydrophobic and protected with a layer of anti-dust coating.

R28. The external structure of the sensors also need to tolerate 60 mph winds just in case there are windstorms.

3.2.2 PERFORMANCE

Hugs the Rails Performance Requirement should include:

R29. IoT HTR should process an event within 3 ms of its occurrence.

R30. IoT HTR sensors should be able to detect an event within 1 ms.

R31. IoT HTR should be able to store 2 TB worth of data.

R32. IoT HTR should have a throughput of 100 gbps.

3.2.3 RELIABILITY

While not technically required to be active 24/7 as the train is ideally going to be connected to the internet for a majority of the time, we need our sensors to be both reliable and accountable.

R33. IoT HTR should have sensors with an uptime of 99.9%.

R34. IoT HTR should have sensors that can reliably reboot if an error occurs.

R35. IoT HTR should have sensors that have a 90% accuracy when sensing the environment, etc.

3.2.4 SECURITY

Physical security may not be a large problem in this project (If anything, we'd need to ensure that no one physically breaks into the compound with the train in it to prevent anyone tampering / breaking the sensors themselves), but cybersecurity can be a big worry in today's world.

R36. IoT HTR needs a security system tight enough to prevent common cyber attacks.

R37. IoT HTR needs a physical security system as well, to prevent break-ins to tamper with the train itself.

R38. IoT HTR should utilize an outside business / system to make a security system on the train.

3.2.5 SCALABILITY

Scalability is a question of whether or not the system in question can handle this growth. In the scope of this project, the "growth" would be defined as getting more train cars, so the question is if the amount of sensors we acquire can handle the amount of trains we acquire.

R39. We should only purchase new trains, new sensors, etc. when we have

enough materials to purchase all at once.

R40. To prevent overbuying, only utilize new materials when absolutely necessary.

3.2.6 LEGAL & REGULATORY REQUIREMENTS

R19. The network nodes should not be able to carry out data analysis.

R20. All the sensors need to do is transmit information through the network to the central server.

R21. Data collection must be kept private and secure by all means.

R22. System should only store data temporarily; this minimizes the damage done if there are ever security breaches.

R23. Trains should be tagged with Radio-Frequency Identification (RFID) technologies recognizable only to the sensors and private domain.

4: REQUIREMENT MODELING

In section four, we want to depict requirements in a way that is relatively easy to understand and more important, straightforward to review for correctness, completeness and consistency. This way, they can readily evaluate all stakeholders, resulting in useful feedback at the earliest time possible. Our section for requirement analysis will cover five principles:

1. The information domain of a problem must be represented and understood.
2. The functions that the software performs must be defined.
3. The behavior of the software must be represented.
4. The models that depict information, function and behavior must be partitioned in a manner that uncovers detailed in layered or hierarchical.
5. The analysis task should move from essential information toward implementation detail.

4.1 USE CASES | SCENARIO-BASED MODELS

4.1.1 LOG IN

Primary Actor: Conductor

Secondary Actor(s): IoT Engine, Display Interface.

Goal In Context: To secure access to the IoT Engine.

Preconditions:

1. IoT Engine is on.
2. Display Interface shows fields for UserID and Password.
3. System must be fully configured.

Trigger: Conductor decides to log in to operate the train.

Scenario:

1. Conductor enters UserID and Password.
2. IoT Engine verifies the UserID and Password.
3. If verification succeeds:
   | IoT Engine displays the IoT HTR Software Interface.
4. Otherwise:
   | Conductors can contact support at HQ for password assistance.
   | Conductors can reset their password with their HTR email address.
   | System will send an alert to HQ if an unknown user is attempting to enter.

4.1.2 LOG OUT

Primary Actor: Conductor

Secondary Actor(s): IoT Engine, Display Interface.

Goal In Context: To secure access to the IoT Engine.

Preconditions:

1. IoT Engine is on.

2. Display Interface shows logout button.
3. System must be fully configured.

Trigger: Conductor decides to log in to operate the train.

Scenario:

1. Conductor clicks to logout.
2. IoT Engine verifies the UserID and Password.
3. If verification succeeds:
   | IoT Engine displays the IoT HTR Software Interface.
4. Otherwise:
   | Conductors can contact support at HQ for password assistance.
   | Conductors can reset their password with their HTR email address.
   | System will send an alert to HQ if an unknown user is attempting to enter.

4.1.3 WHEEL SLIPPAGE

Primary Actor: IoT Wheel Sensors

Secondary Actor(s): Conductor, IoT Engine, IoT Display Interface, GPS Speed Sensors.

Goal In Context: Will warn conductor of potential hazards from the wheels.

Preconditions:

1. IoT Engine is on.
2. IoT HTR Software Interface is logged in.
3. IoT HTR Software initializes the sensors.

Trigger: Logging into the IoT HTR Software.

Scenario:

1. IoT Wheel Sensors sends wheel RPM to IoT Engine.
2. IoT Engine determines the average of the wheel sensor RPM data.
3. IoT Engine calculates the speed based on average RPM Data & Wheel Radius.
4. GPS Speed Sensor sends train speed to the IoT Engine.
5. IoT Engine compares GPS Speed with Wheel Speed.
6. If difference is greater than or equal to 0.1:
   | IoT Engine displays the slippage warning on the Display Interface.
   | IoT Engine display recommendations to “SLOW DOWN” or “STOP”
   | Conductor will act according to the requirements.
7. Otherwise:
   | IoT Engine displays all “All systems are a go.”

4.1.4 OBJECTS ON TRACK

Primary Actor: LIDAR Sensors (Light Detection and Ranging)

Secondary Actor(s): Conductor, IoT Engine, IoT Display Interface, GPS Speed Sensors

Goal In Context: Will warn conductors of potential hazards on rail, behind and infront.

Preconditions:

1. IoT Engine is on.
2. IoT HTR Software is logged in.
3. IoT HTR Software initialized all sensors.

Trigger: Logging into the IoT HTR Software.

Scenario:

1. IoT Proximity Sensors send location information to IoT Engine.
2. IoT Engine calculates the range of the object and train.
   | IoT Engine uses the average location of the train in the previous 10 seconds.
3. IoT Proximity Sensor will keep sending location information if the object's location is changing.
4. GPS Speed Sensor sends train speed to the IoT Engine.
5. IoT Engine will calculate the speed of the object if step 3 happens.
6. IoT Engine compares GPS Speed with Proximity (and Object Speed) .
7. IoT Engine will display a simple surrounding interface to information the conductor.
8. If Object is in front of them and Object Speed is non-existent:
   | IoT Engine will display a crash warning on the Display Interface.
   | IoT Engine will display recommendations to either "STOP" or "SLOW DOWN"
   | Conductor will act according to the requirements.
9. If Object is behind them and GPS speed is less than Object Speed:
   | IoT Engine will display a crash warning on the Display Interface.
   | IoT Engine will display recommendations to either "SPEED UP"
   | Conductor will act according to the requirements.
10. If Object is in front of the train and GPS speed is greater than Object Speed.
    | IoT Engine will display a crash warning on the Display Interface.
    | IoT Engine will display recommendations to either "STOP" or "SLOW DOWN"
    | Conductor will act according to the requirements.
11. Otherwise:
    | IoT Engine will display all "All systems are a go."

4.1.5 GATES

Primary Actor: Gates Sensors

Secondary Actor(s): Conductor, IoT Engine, IoT Display Interface, GPS Speed Sensors

Goal In Context: The train slows down or speeds up depending on the state of the gate ahead.

Preconditions:

1. IoT Engine is on.
2. HTR software is logged on.
3. Gates on the tracks are fully functional.

Trigger: Sensors detect a gate ahead, either open or closed.

Scenario:

1. The train is travelling at normal speeds and detects a gate ahead on the tracks.
2. IoT Engine calculates the distance between the gate and the train.
   | IoT Engine will play horn sound when the train is within 1 mile of the gate.
   | IoT Engine uses the average location of the train in the previous 10 seconds.
3. GPS Speed Sensor sends the train speed to the IoT engine.
4. IoT Engine compares the GPS Speed with the distance calculated to the gate.
5. IoT Engine will calculate whether the gate is opened or closed.
6. IoT Engine will display an interface with information to the conductor.
7. If the gate ahead is open:
   | IoT Engine will play horn sound.
   | IoT Engine will display that it is all clear ahead to the Display Interface.
   | IoT Engine will display a recommendation to “SPEED UP” or “STAY AT PACE”
   | Conductor will act according to the requirements.
8. If the gate ahead is closed:
   | IoT Engine will display a crash warning on the Display Interface.
   | IoT Engine will play horn sound.
   | IoT Engine will display recommendations to either
   “SLOW DOWN” or “STOP”
   | Conductor will act according to the requirements, depending on if the train is already stopped.

4.1.6 ENGINE

Primary Actor: Temperature Sensors

Secondary Actor(s): Conductor, IoT engine, IoT Display Interface, LCS

Goal In Context: The train’s coolant systems go into overdrive if the engine starts overheating.

Preconditions:

1. IoT Engine is on.
2. HTR software is logged in.
3. The thermostat and coolant flow work properly and get regular maintenance.

Trigger: Depending on the temperature of the train engine, the coolant systems activate or remain on standby.

Scenario

1. All system interfaces are online.
2. The temperature sensors are constantly monitoring the engine to detect temperature spikes.
3. If the temperature graph increases exponentially, the sensors will send a message to the IoT Engine to stay alert.
4. Once the temperature of the engine exceeds 150°C, the IoT Engine gets notified and sends a warning to the conductor and LCS that the train requires attention.

   | IoT Engine will display the warning: “ENGINE OVERHEATING. COOLANT SYSTEMS ARE IN EFFECT.”
5. The coolant systems activate, and will continue to work on cooling the engine until the temperature returns to the acceptable range of around 120°C.

   | Once everything is back to normal, the conductor will receive the message “All systems are a go.”
6. The coolant systems should be enough to allow the train to function normally. If there does exist a scenario where the engine temperature is still above the optimal range for the next hour, the conductor should notify the control room operators (or God) and follow the given instructions.

4.1.7 POWER

Primary Actor: UPS Battery

Secondary Actor(s): IoT Engine, Display Interface

Goal In Context: Report warnings to the Conductor about status of electrical power to the system.

Preconditions:

1. IoT Engine is on.
2. HTR software is logged in.
3. System has electrical power.

Trigger: Logging in to the IoT HTR software.

Scenario:

1. The system is logged in and in use.
2. The UPS battery backup has a percentage representing how full it is.
3. The percentage is read from the battery and sent to the IoT Engine.
4. If the battery is completely drained:

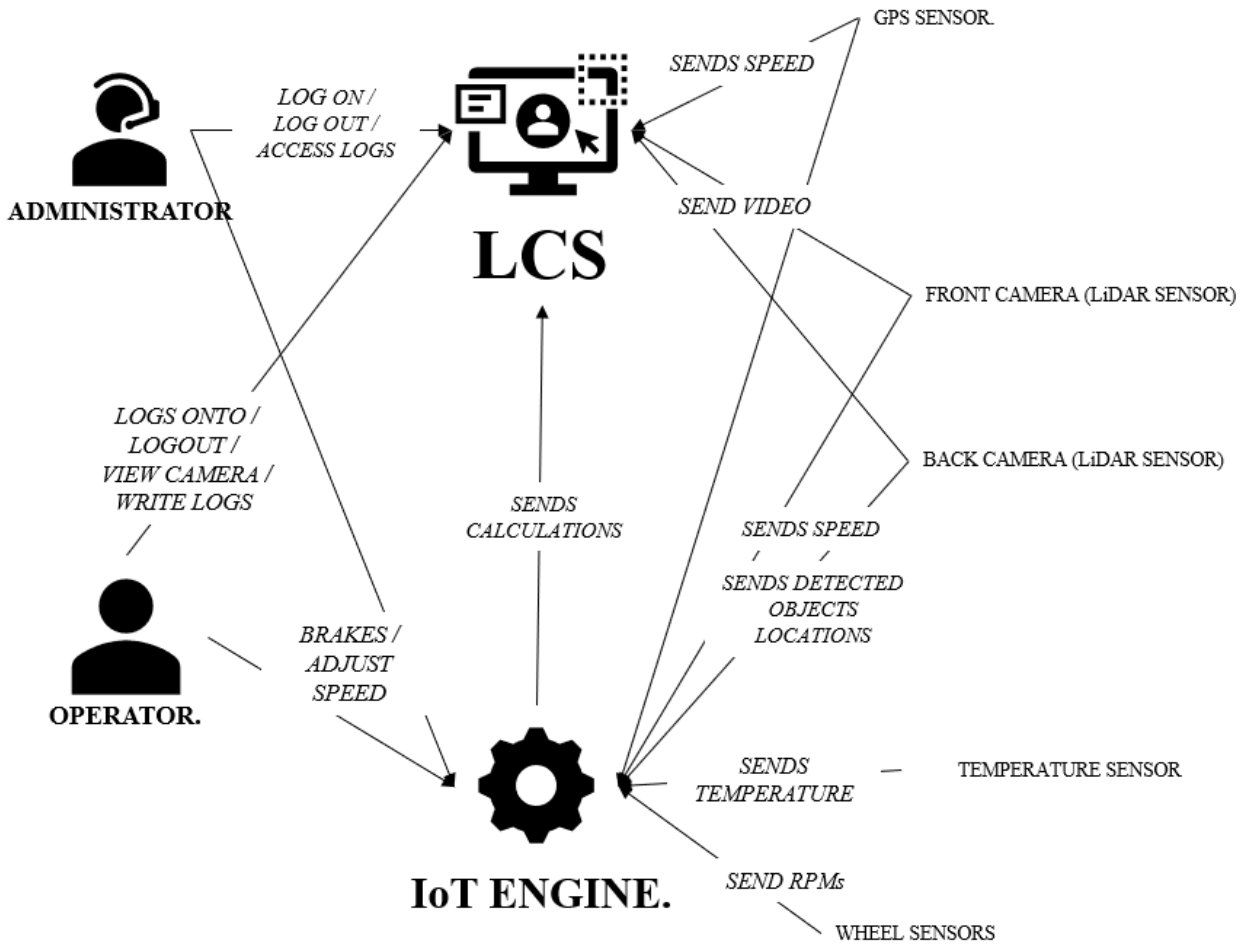
   | IoT Engine displays “WARNING: BATTERY DEPLETED” and suggests the train slows to allow passive charging.
5. If the battery percentage is below the safety threshold of 50%:

   | IoT Engine displays “WARNING: BATTERY LOW” alongside the percentage.
6. If the battery percentage is above the threshold:

   | IoT Engine displays “BATTERY SAFE” alongside the percentage.

7. The conductor should act according to the requirements and the expected length of the trip to the next station.

4.2 USE CASE DIAGRAM



4.3 CLASS-BASED MODELING

4.3.1 KEY CHARACTERISTICS

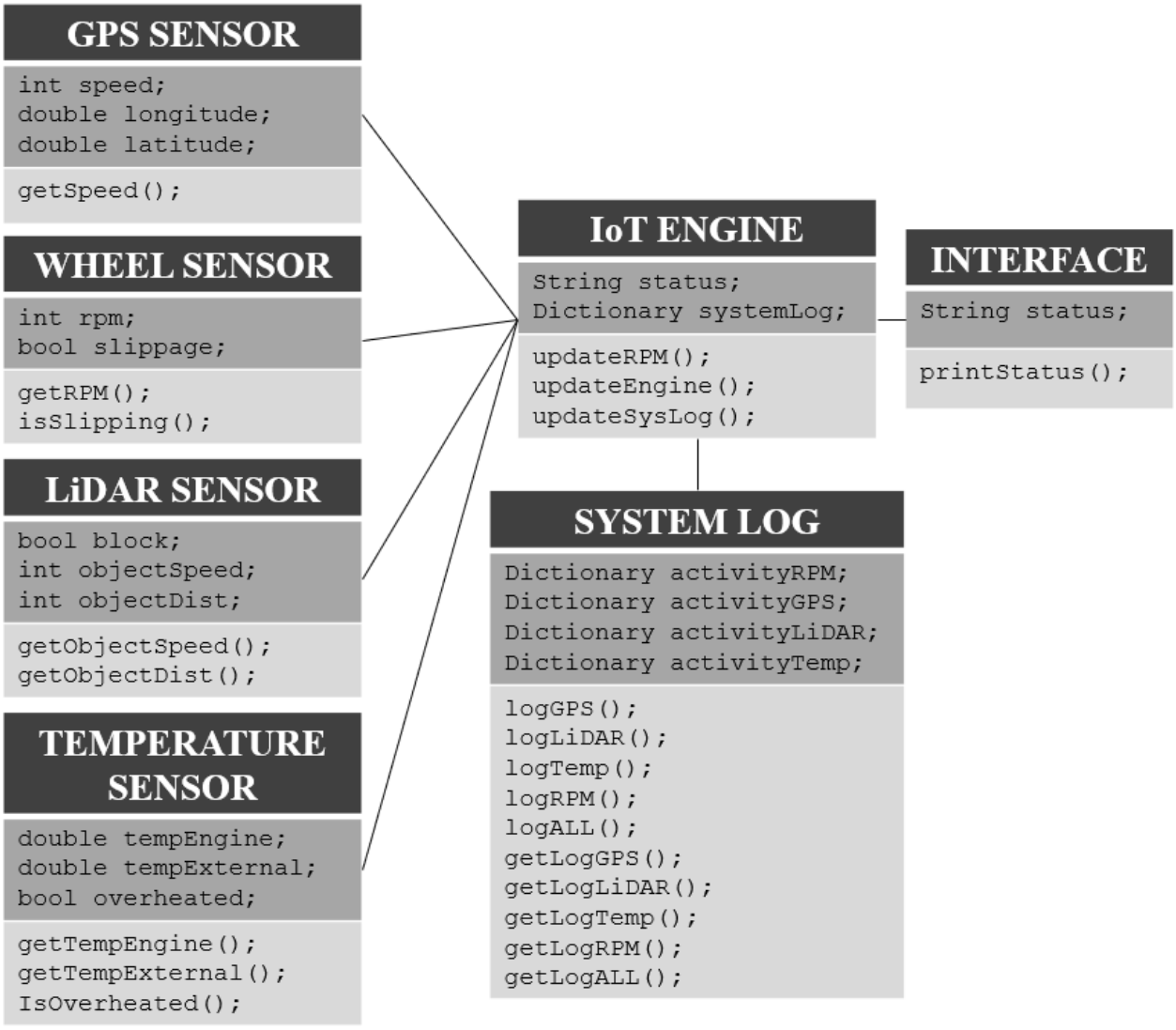
1. Retained Information.
2. Needed Services.
3. Multiple Attributes.
4. Common Attributes.
5. Common Operations.
6. Essential Requirements.

4.3.2 POTENTIAL CLASSES

Potential Class	General Classification

Operator / Conductor	Role, External Entity, Internal Entity
GPS Sensor	External Entity
Wheel Sensor	External Entity
Front Camera / LiDAR Sensor	External Entity
Back Camera / LiDAR Sensor	External Entity
Temperature Sensor	External Entity
IoT Engine	Internal Entity
Layout Control System (LCS)	Internal Entity
User ID / Password	Internal Entity

4.3.3 SYSTEM CLASS DIAGRAM

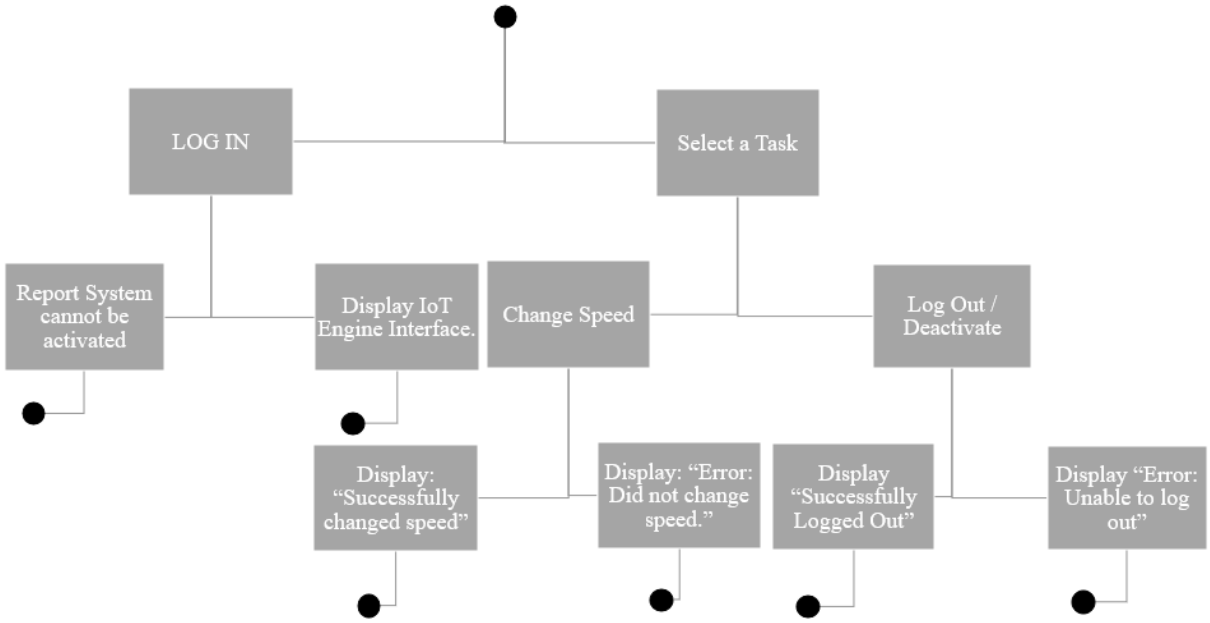


4.4 CRC MODELING / CARDS

GPS SENSOR	TEMPERATURE SENSOR	LiDAR SENSOR
System Status: ON Display Message: N/A Display Status: N/A	System Status: ON Display Message: <Temperature> Display Status: Current Temp	System Status: ON Display Message: <Video> Display Status: Feed From Cameras
TODO: • If train is on, detect speed.	TODO: • If train is on, detect engine temp. • If train is on, detect external temp.	TODO: • If object in the way, prompt warning message.

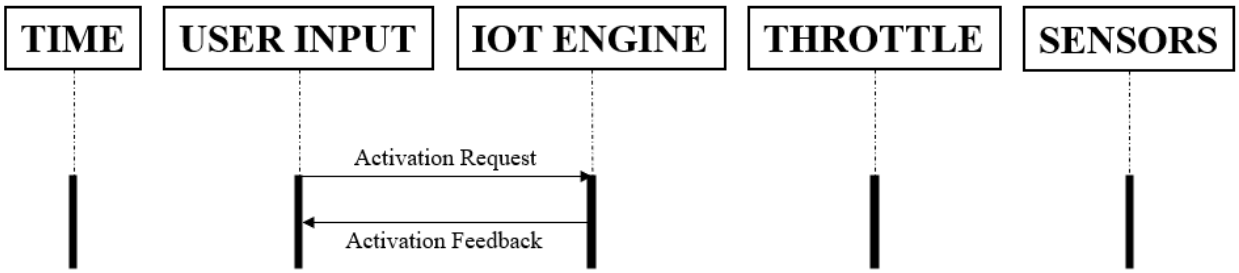
WHEEL SENSOR	SYSTEM LOGS	INTERFACE
System Status: ON Display Message: N/A Display Status: N/A	System Status: ON Display Message: N/A Display Status: N/A	System Status: ON Display Message: <interface> Display Status: Interface
TODO: • If train is on, detect RPM.	TODO: • If train is on, log all sensors. • If logged in, user may input comments.	TODO: • If logged in, display interface.

4.5 ACTIVITY DIAGRAMS

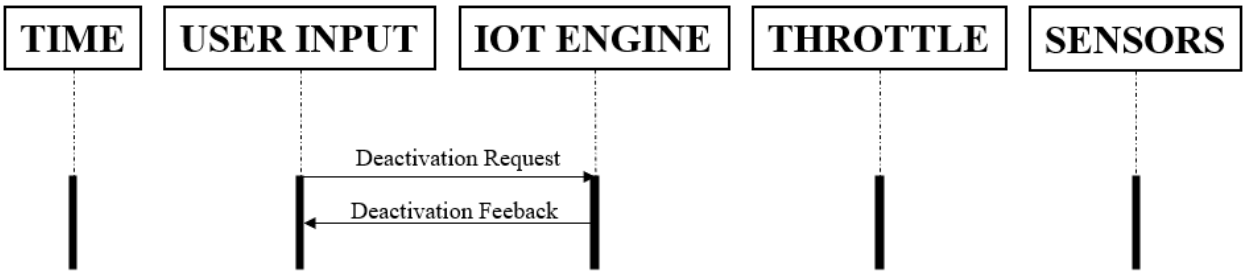


4.6 SEQUENCE DIAGRAMS

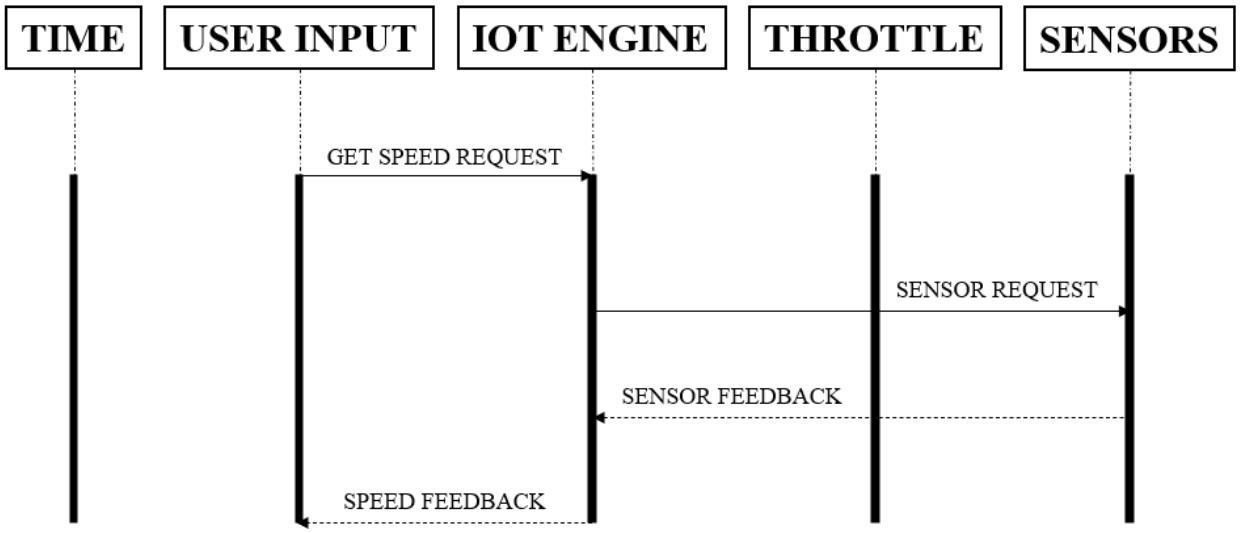
4.6.1 LOG IN



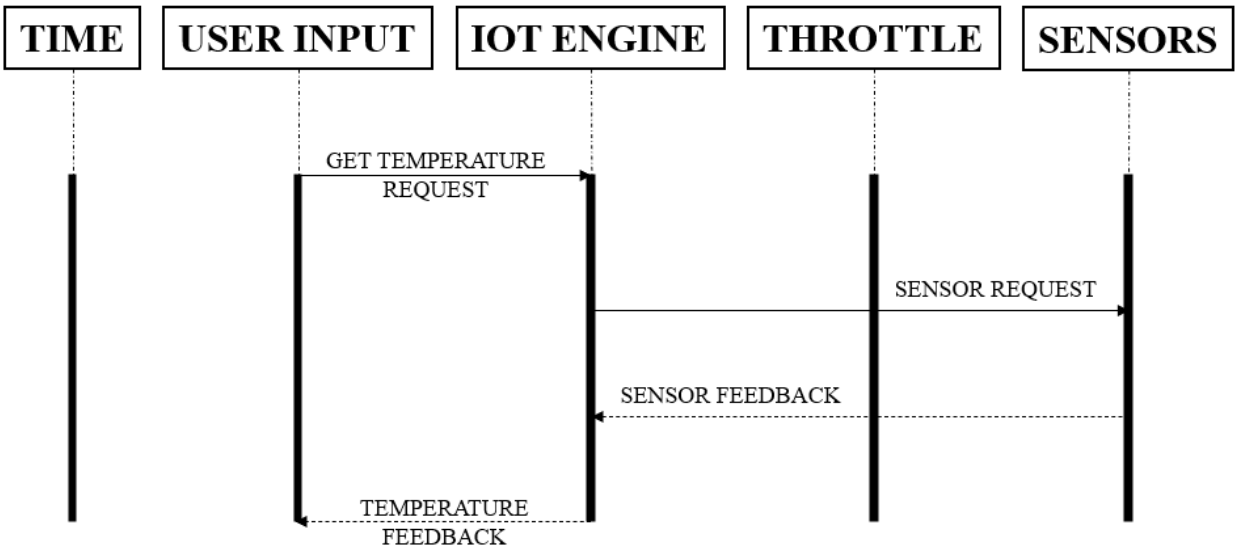
4.6.2 LOG OUT



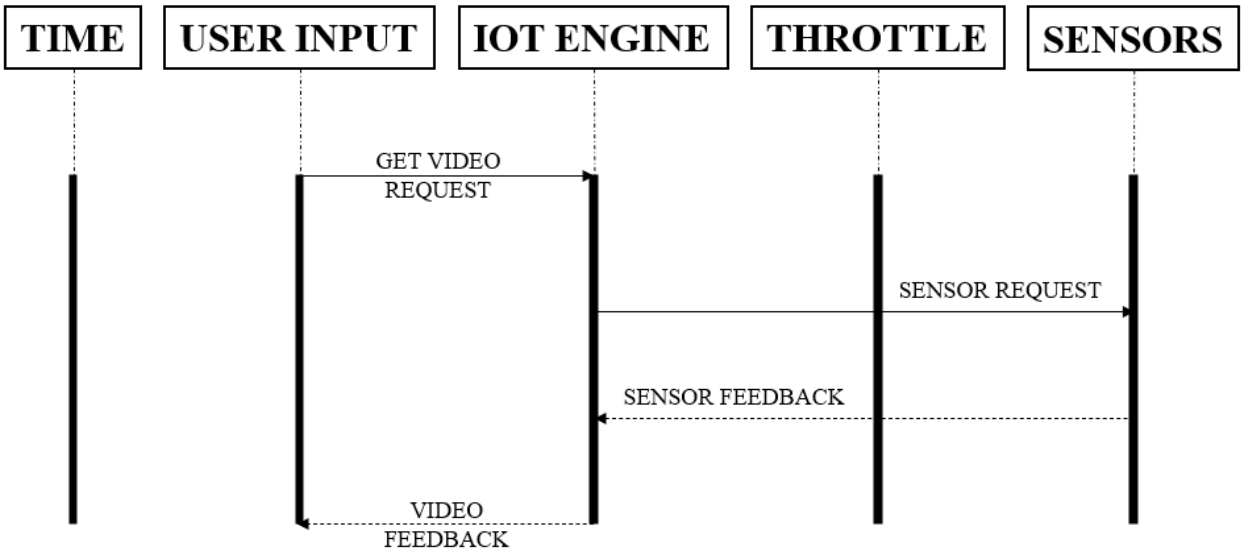
4.6.3 GETSPEED()



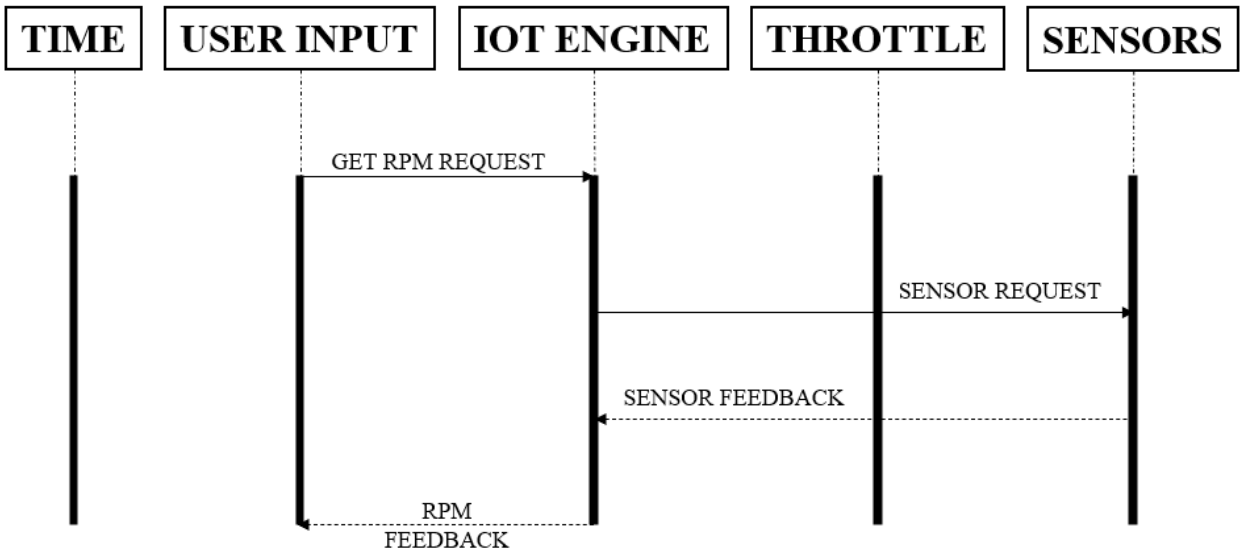
4.6.4 GETTEMPERATURE()



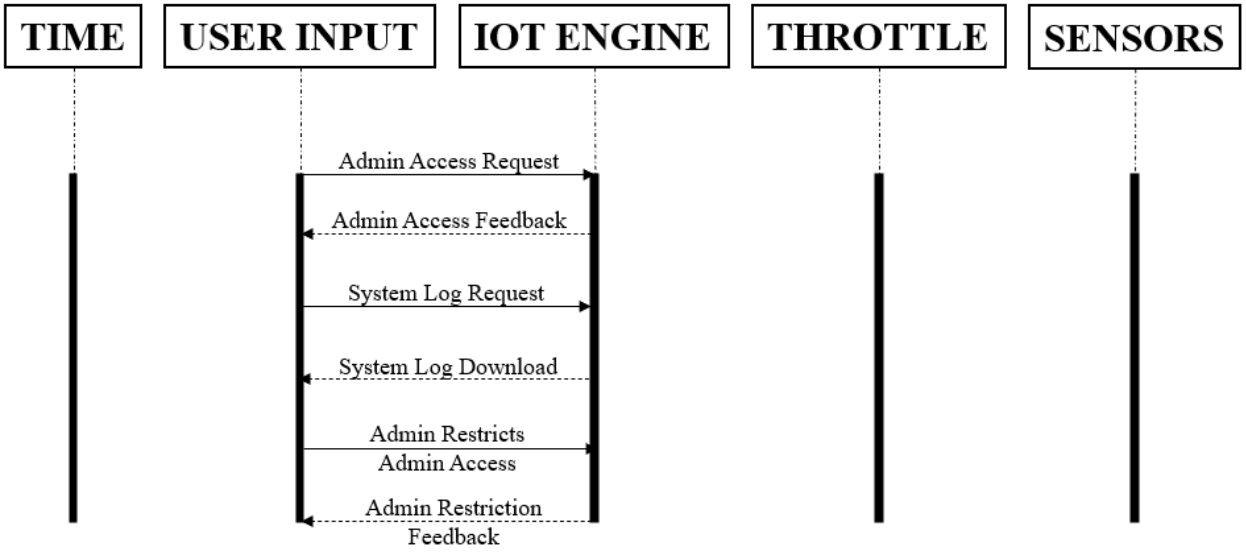
4.6.5 GETVIDEO()



4.6.6 GETRPM()

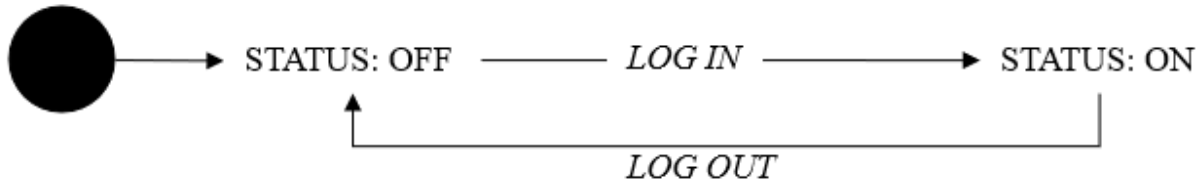


4.6.7 ADMIN ACCESS

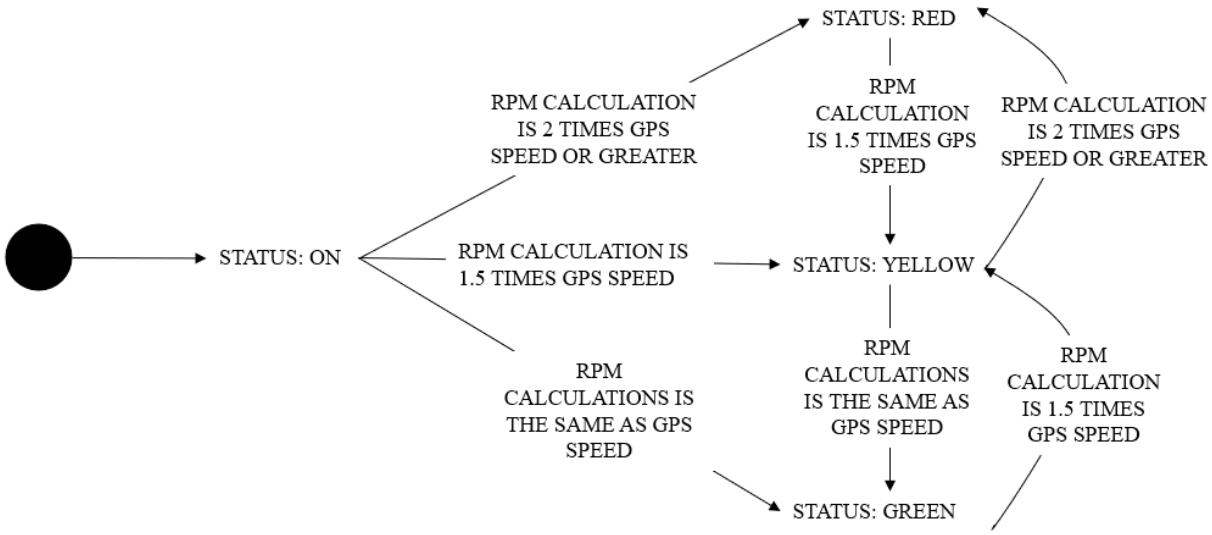


4.7 STATE DIAGRAM

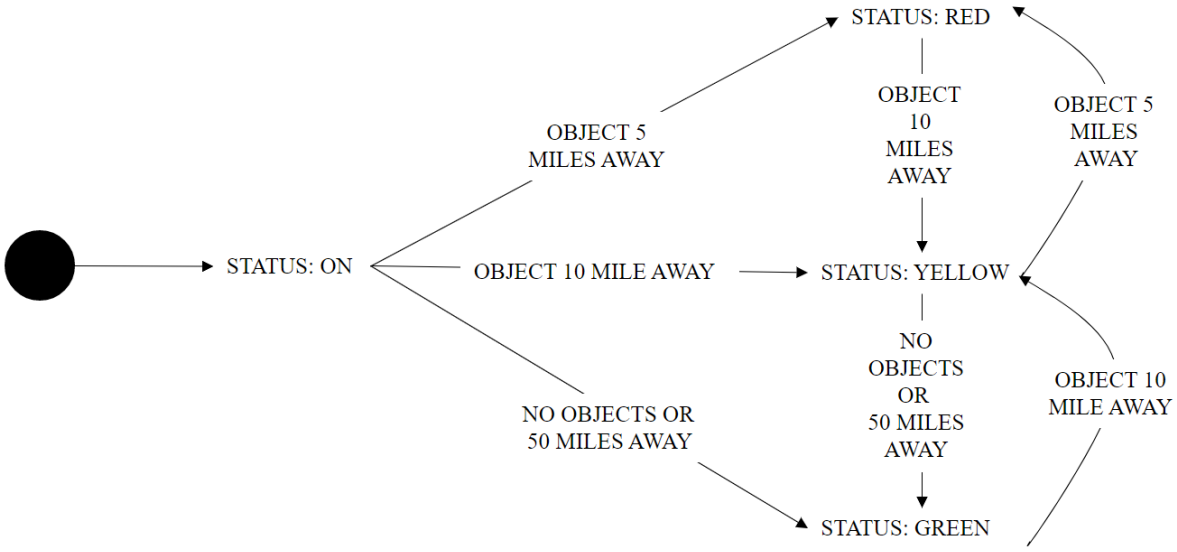
4.7.1 LOG IN / LOG OUT



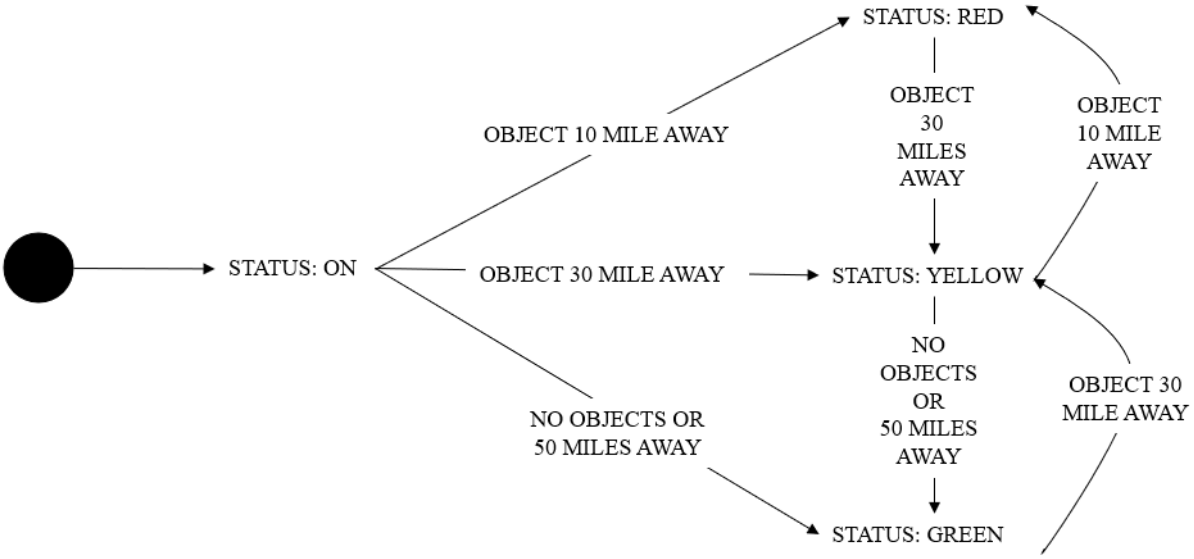
4.7.2 WHEEL SLIPPAGE



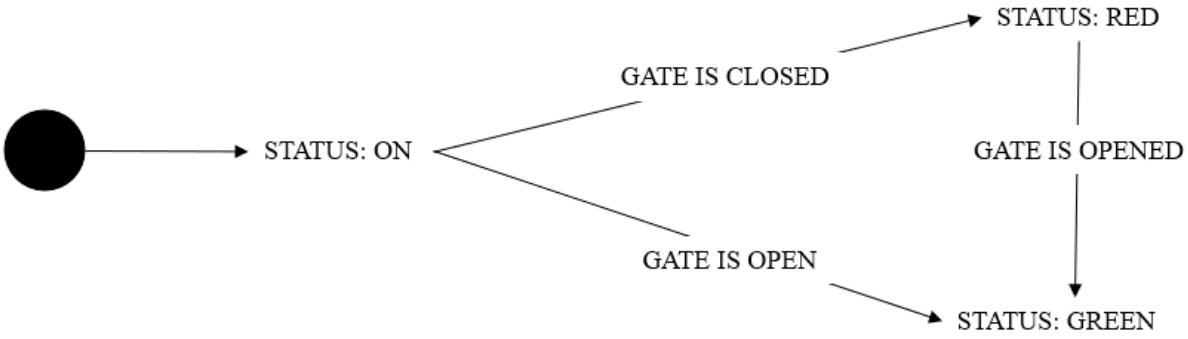
4.7.3 STANDING OBJECTS



4.7.4 MOVING OBJECTS



4.7.5 GATE DETECTION



5: SOFTWARE ARCHITECTURE / DESIGN

5.1 SOFTWARE ARCHITECTURE MODELS

5.1.1 DATA CENTERED ARCHITECTURE

Feasibility: This would not be very feasible because the train is meant to be built independently of outside data centers.

Pros:

- Provides data integrity, backup, and restore features.
- Provides scalability and reusability of agents as they do not have direct communication with each other.
- Reduces overhead of transient data between software components.

Cons:

- It is more vulnerable to failure and data replication or duplication is possible.
- High dependency between data structure of data store and its agents.
- Changes in data structure highly affect the clients.
- Evolution of data is difficult and expensive.
- Cost of moving data on network for distributed data.

5.1.2 DATA FLOW ARCHITECTURE

Feasibility: Relatively applicable because the data from our train's sensors is transformed through a flow of computations and data manipulation.

Pros:

- Provides simpler division on subsystems.
- Each subsystem can be an independent program workin on input data and producing output data.

Cons:

- Provides high latency and low throughput
- Does not provide concurrency and interactive interface. External control is required for implementation.

5.1.3 CALL AND RETURN ARCHITECTURE

Feasibility: Since the train has a number of programs to manage each sensor, this is a feasible model.

Pros:

- Easily Modified
- Grow system functionality by adding more modules.
- Simple to analyze control flow.

Cons:

- Parallel processing might be difficult.

- May be difficult to distribute across machines. (traditional)
- Exceptions to normal operations are awkward to handle.

5.1.4 OBJECT ORIENTED ARCHITECTURE

Feasibility: The train will take in an input from the sensors and produce a readable output for the conductor, so object oriented architecture is a very appropriate model.

Pros:

- Objects and methods are very readable and understandable.
- Class hierarchies (showing a combination of structure and the interconnections) help visualize the real structure of the design.
- Utilizes an imperative style, in which code reads like a straightforward set of instructions as a computer would read it.
- Our team members are somewhat familiar with the architecture.
- Allows us to find documentation and information easily.

Cons:

- Commonly depends upon shareable state. This means that a variable can be accessed from more than one function, for example when a function(s) changes some variables without explicitly saying so.
- Can end up creating a massive amount of unnecessary code. This can cause the Uses more CPU than others. It is an inefficient choice when there are limitations.

5.1.5 LAYERED ARCHITECTURE

Feasibility: This is a feasible model because there are multiple layers in the train: for example, the sensors, the mechanics to take in input and return output, and the conductor's interface.

Pros:

- Can help with solving problems by separating the issue into multiple smaller problems.
- Provides a sense of modularity, along with interfaces being very clear.
- Easier to test and debug.

Cons:

- Development time is much slower due to the necessity to work on multiple layers.
- Performance time slows down as more layers are added to the project.

5.1.6 MODEL VIEW CONTROLLER ARCHITECTURE

Feasibility: With the conductor acting as a controller, this architecture works well because the conductor can receive input and send output to the train's system.

Pros:

- Provides a faster development cycle.
- Easier to update programming, along with debugging.
- Better environment for multiple developers to work together.

Cons:

- Must be strict with method ruling, otherwise the hierarchy between controller and model is ruined.
- Must be formatted and organized by the appropriate view object.

5.2 ARCHITECTURE STYLE

We have chosen to implement Object Oriented Architecture.

5.3 SOFTWARE COMPONENTS

Software components are software entities that are executed somewhere in a distributed environment like the internet. They offer a specific set of services through a well-defined interface. We will be using the following software components:

- User Interface
- Network / IoT
- Operating System (Hardware)
- Operating System (Software)
- System / Command Shell

The main component of our HTR software is the engine. In order for the engine to be controlled and changed, the HTR will receive data from the sensors on the outside of the train. To complete this, the sensors will relay their data from their respective parts of the train (front, side, back) and send them to the conductor's UI, which is used to control the engine itself.

The components stated above are how the sensors link together. By using the connectors in this architecture, the sensors can send their data to the conductor, who can use connectors once again to send instructions to the engine.

Examples of sensors using connectors to send data: utilizing the speed sensor to send speed to the conductor, who can alter speed depending on what the result is; temperature sensor returning how hot the engine currently is; etc.

5.4 SOFTWARE MANAGEMENT

MAIN FUNCTION

isSlipping();
isOverheated();
 	
updateRPM();
updateEngine();
updateSysLog();
 	
logGPS();
logLiDAR();
logTemp();
logRPM();
logALL();
 	
getSpeed();
getRPM();
getObjectSpeed();
getObjectDist();
getTempEngine();
getTempExternal();
 	
getLogGPS();
getLogLiDAR();
getLogTemp();
getLogRPM();
getLogALL();
 	
printStatus();

FUNCTION PURPOSE

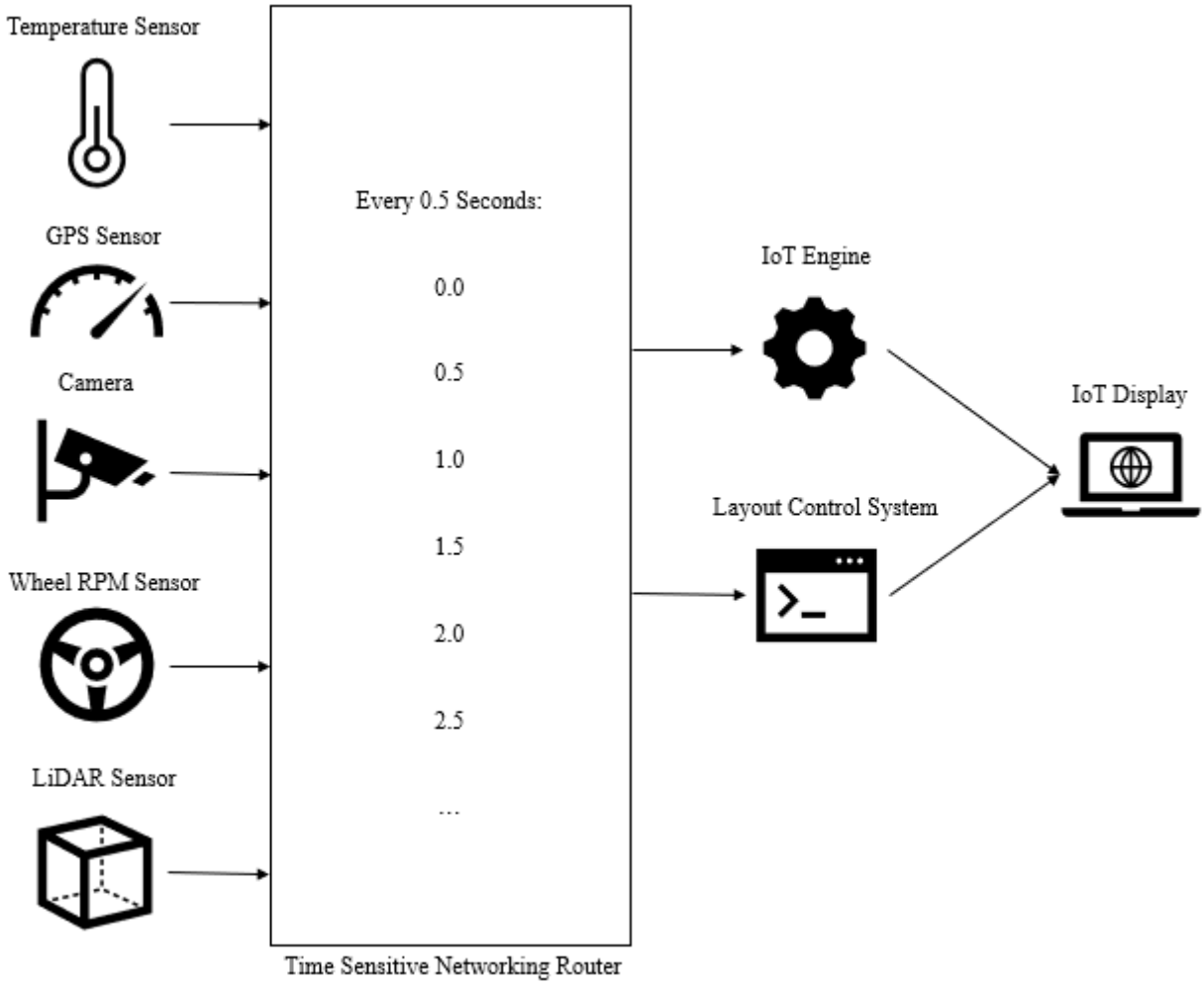
```
isSlipping() - Are the wheels slipping? Returns a Boolean.
isOverheated() - Is the Temperature of the Engine overheating? Returns a Boolean.

updateRPM() - Every 5 seconds it will call getRPM(). Returns a Integer.
updateEngine() - Every 5 seconds it will call getTempEngine(). Returns a Double.
updateSysLog() - Every 5 seconds it will logALL(). Does not return a value.

logGPS() - Store getSpeed() value into the logs. Does not return a value.
logLiDAR() - Store getObjectDist() value into the logs. Does not return a value.
logTemp() - Store getTempEngine() and getTempExternal() values into the logs.
            Does not return a value.
logRPM() - Store getRPM() value into the logs. Does not return a value.
logALL() - Calls all log functions. Does not return a value.

getSpeed() - Returns speed from GPS sensor.
getRPM() - Returns RPM of wheels(s) from Wheel Sensor.
getObjectSpeed() - Return speed of object by calculation.
getObjectDist() - Returns object distance from LiDAR sensor.
getTempEngine() - Returns engine temperature from temperature sensor.
getTempExternal() - Returns outside temperature from temperature sensor.

getLogGPS() - Returns the log value after call getSpeed().
getLogLiDAR() - Returns the log value after call getObjectSpeed() and getObjectDist().
getLogTemp() - Returns the log value after call getTempEngine() and
               getTempExternal().
getLogRPM() - Returns the log value after call getRPM().
getLogALL() - Returns a string of all getLog functions;

printStatus() - No return value. Prints Status onto interface.
```

The main function shall call all functions listed in the figure above: isSlipping(), isOverheated(), updateRPM(); updateEngine(), updateSysLog(), logGPS(), logGPS(), logLiDAR(), logTemp(), logRPM(), logALL(), getSpeed(), getRPM(), getObjectSpeed(), getObjectDist(), getTempEngine(), getTempExternal(), getLogGPS(), getLogLiDAR(), getLogTemp(), getLogRPM(), getLogALL(), and printStatus(). These functions are essentially what is going in the Layout Control System.

The first two functions isSlipping() and isOverheated() are booleans that represent if true when the wheels are sleeping or when the engine is over the normal temperature and false when not. Whether the LCS is connected or not connected to the cloud, it will update the user interface every 5 seconds with the RPM and the engine temperature. This is able to occur because of the update functions. The last update function allows us to store the logs continuously on a temporary cloud.

The rest of the function except the last one returns a value. The first six get functions that take in raw data from the sensors and convert them into computable values. The rest of the get functions returns a readable value for the user. While the print status is a function that prints the RPM, Engine Temperature and Camera feed onto the User Interface.

5.5 DATA ARCHITECTURE



The data architecture for this project will also be represented in object oriented fashion. This is because our overall architecture style is object oriented, so having the two in the same technical architecture is convenient for the overall progress of the project. It is most likely that the project will be coded in Java as well, as it is an object oriented programming language that most of us have experience with.

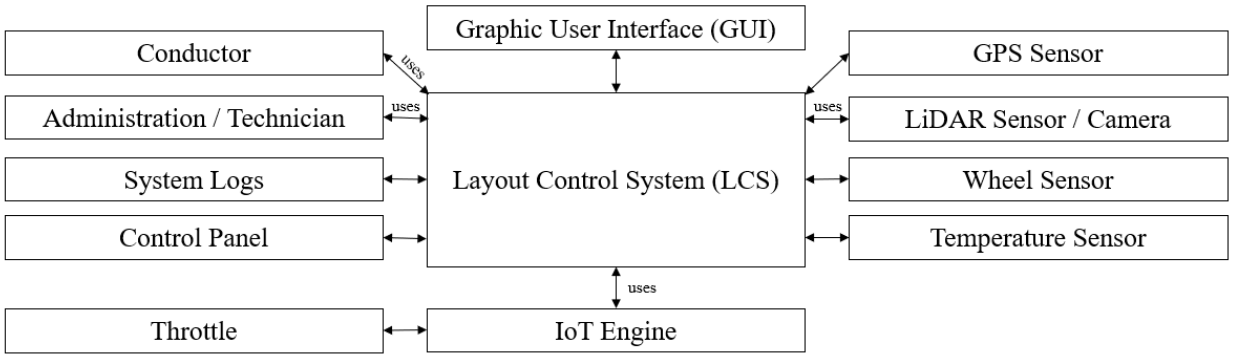
Communication between the components of the project will be accomplished by passing through parameters from one function to another. This will allow each class to communicate with each other, allowing a coherent flow of information. The HTR software requires a lot of communication between the sensors and the engine itself, so these functions and methods will prove very useful. In addition to the necessity of passing information from one place to another, it needs to happen quickly to allow for critical decisions to be made. This architecture lends itself well to this requirement, as it passes information from one place to another using utilities such as parameters, leading to an easier time accessing data. This will minimize the risk of a crash or other catastrophe.

To implement this structure, our team defined functional components that each handle its own job and can be called and return its own data. For example, our structure has components that can calculate and return the current speed, temperature of the engine, and RPM of the train's wheels. These are just a few of the many functions we have that are used to return information to the conductor's user interface. These functions are always available for use as functional components in the HTR software, as they will repeatedly be used to poll and return the data so the conductor can accurately make decisions based on the information at hand.

5.6 ARCHITECTURAL DESIGNS

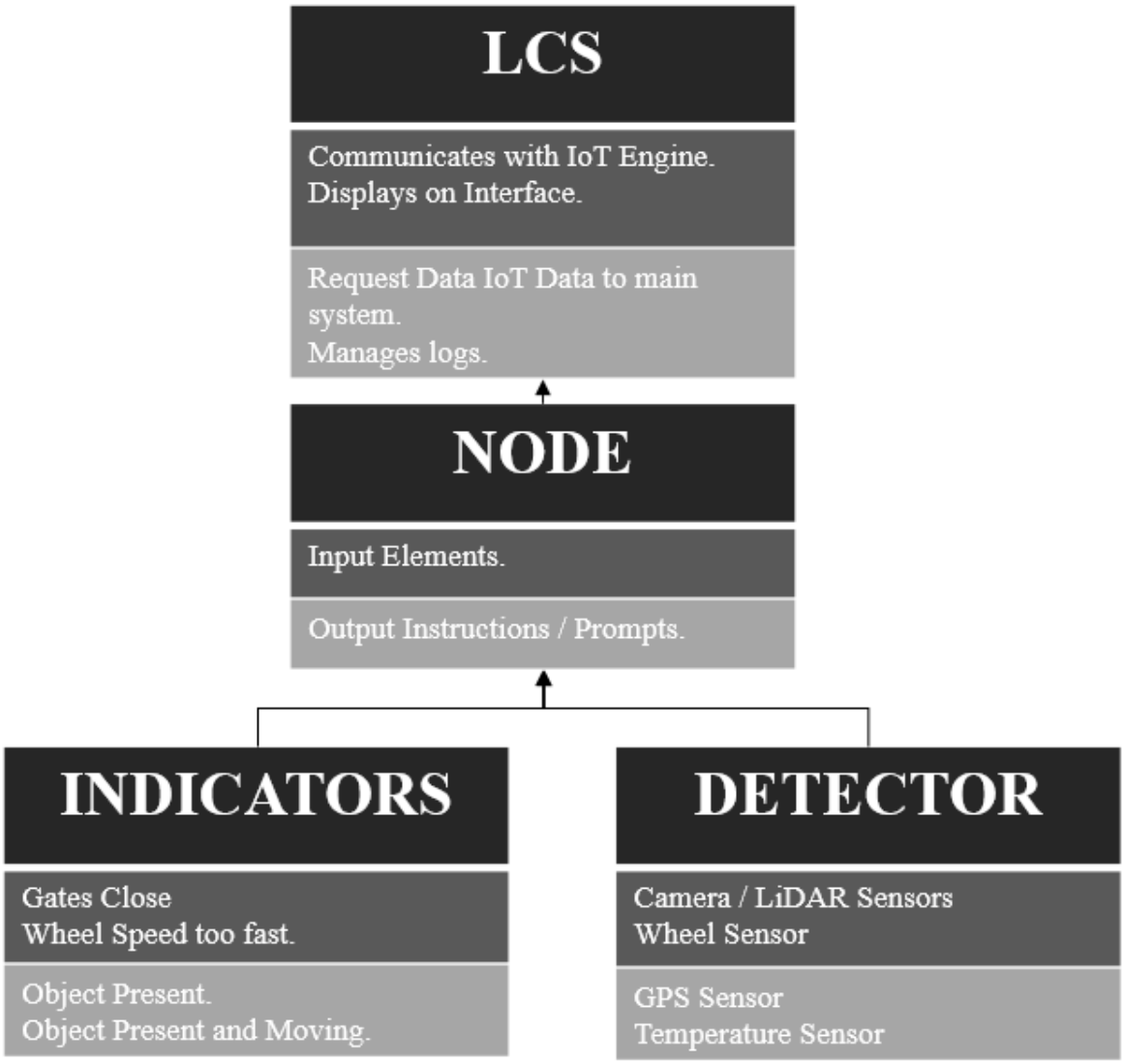
5.6.1 ARCHITECTURAL CONTEXT DIAGRAM

The figure below represents the architectural context diagram. This diagram models the manner in which software interacts with the entities external to its boundaries.



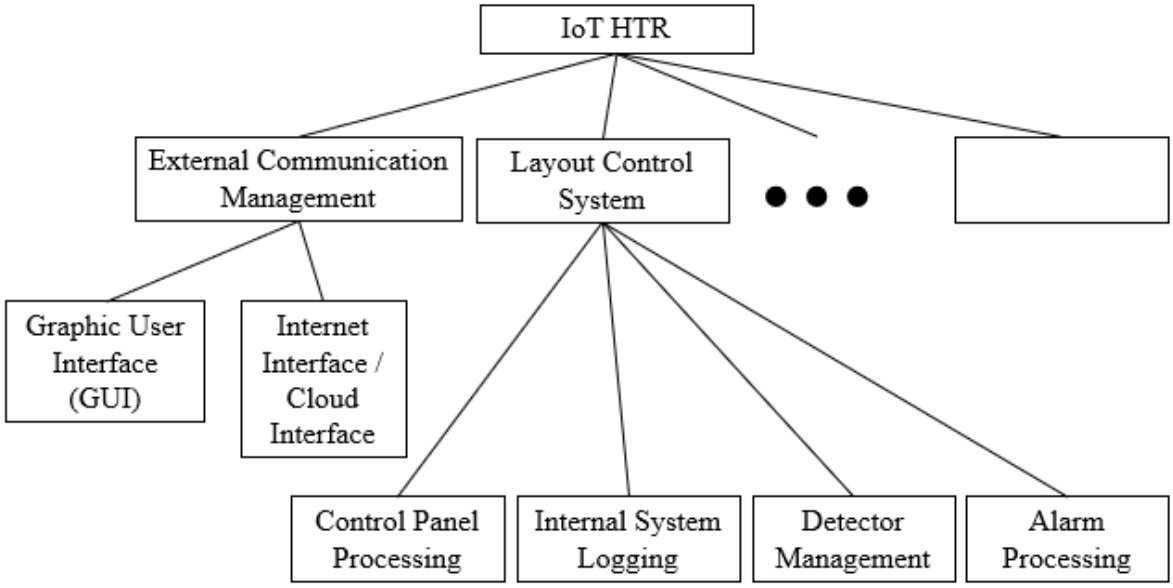
5.6.2 ARCHETYPE FUNCTION UML DIAGRAM

The figure below represents the archetype diagram. The archetype is an abstraction that represents an element of the Layout Control System behavior.



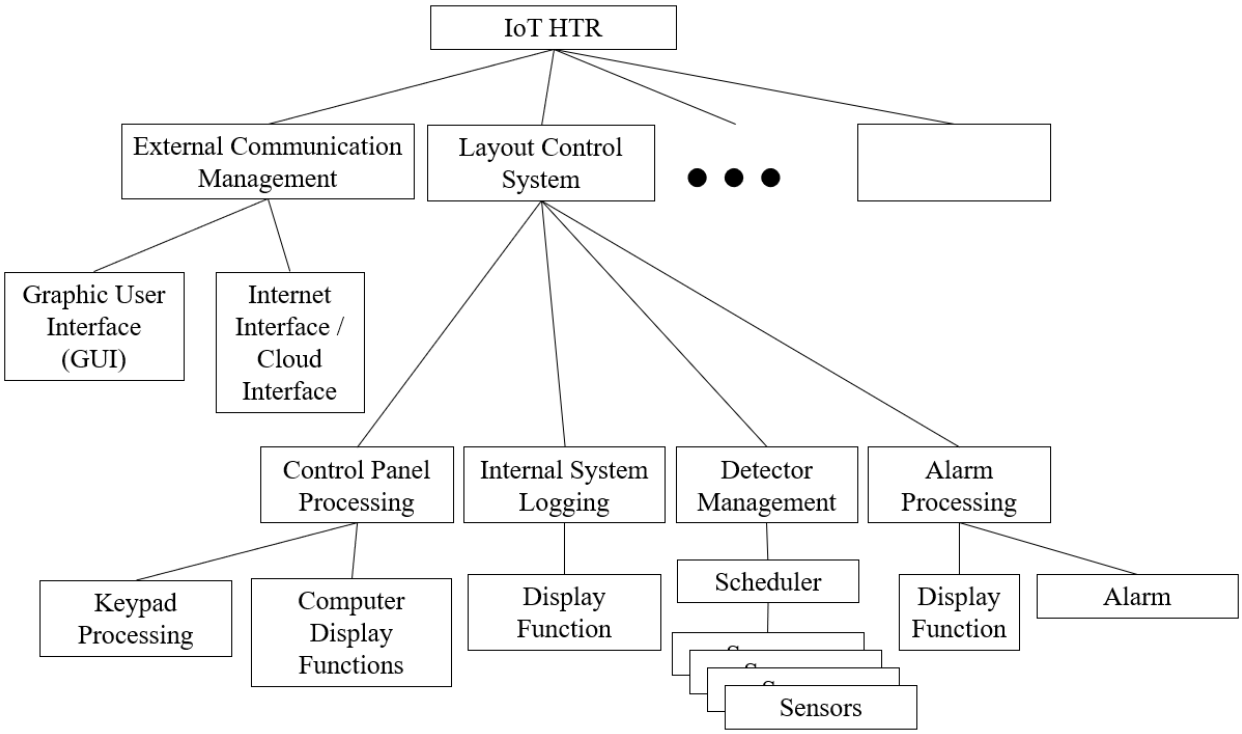
5.6.3 TOP LEVEL COMPONENTS

In the figure below shows the top-level component in which the archetype is further defined for help with implementation. This level gives the view an overview of what is going on.



5.6.4 REFINED COMPONENTS

In the figure below shows the refined components architecture. This is the top-level component with more abstraction. Being a little more abstract allows the developer to easily implement what is needed.



6: CODING

6.1 PROJECT CODE

```
<!DOCTYPE html>
<html>
  <title>IoT Details</title>
  <head><script src="main.js"></script></head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="style.css">
  <link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Montserrat">
  <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.mi
n.css">

  <body class="-black">

  <nav class="-sidebar -bar-block -small -hide-small -center">
    <a href="#" class="-bar-item -button -padding-large -hover-black">
      <i class="fa fa-home -xxlarge"></i>
      <p>HOME</p>
    </a>
    <a href="#about" class="-bar-item -button -padding-large -hover-black">
      <i class="fa fa-user -xxlarge"></i>
      <p>ABOUT</p>
    </a>
  </nav>

  <!-- Navbar on small screens -->
  <div class="-top -hide-large -hide-medium" id="myNavbar">
    <div class="-bar -black -opacity -hover-opacity-off -center -small">
      <a href="#home" class="-bar-item -button" style="width:50%
!important">HOME</a>
      <a href="#about" class="-bar-item -button" style="width:50%
!important">ABOUT</a>
    </div>
  </div>


  <div class="-padding-large" id="main">
    <!--HOME-->
    <div class="-padding-large -center" id="home">
      <i class="fa fa-train -xxlarge -trainicon"></i>
      <header class="-container -padding-32 -center -black" id="home">
        <h1 style="font-size:150px;margin-top:0px">Welcome to Hugs the Rail.</h1>
      </header>
      <form name = "login">
        <div class = "-center" style = "position:relative; padding-bottom:15px;">
            Username: <input type = 'text' name = "userid">
        </div>
        <div class = "-center" style = "position:relative; padding-bottom:15px;">
            Password: <input type = 'text' name = "password">
```

```
        </div>
        <div class = "-center" style = "position:relative;padding-bottom:15px;">
          <button type = "reset" onclick = "check(this.form)" class = "nicebutton">
Login </button>
        </div>
      </form>
    </div>

  <!-- About Section -->
  <div class="-content -justify -text-grey -padding-64" id="about">
    <h2 class="-text-light-grey">ABOUT</h2>
    <hr style="width:200px" class="-opacity">
    <p>
        Internet of Things: Hugs The Rail is a project that will produce a product
to railroad users and
        supporters to enhance the ride. These changes will include being able to
make decisions locally
        in absence of cellular and wifi connectivity back the offices, being able
to capture data
        from the locomotives and its surrounding environment, issuing an analytic
engine on board to
        process as much information to safely guide passengers as well as the
conductor on their destination,
        make decision that is passed on the locomotive control system, providing
room for the operator
        to enter commands to receive status, and the ability to download the latest
rules from the cloud
        into the engine. The Little Engineers That Could, Team Eight is a talented
team filled with curiosity,
        structured by organized members, and constructed with knowledge to perform
these tasks.
    </p>
    <p>
        Currently, the train system involved with the Hugs the Rail system is
completely reliant on the internet
        and network connection it has. If a scenario occurred where the internet /
network connection was lost,
        however, the trains would be lost with no idea on where to go or what to do
incase of an emergency.
        This is precisely the problem at hand. The Hugs the Rail system will
implement sensors into these trains,
        which function locally, not relying on a network connection with the
outside world, so that the trains will
        function in a scenario where there is no possible way to connect to the
internet.
    </p>
    <p>
        Locomotives used by Hugs the Rails should aim to get every package onboard
as quickly and efficiently
        as possible so that the people and companies can get to their parcels in a
timely manner. They
        should also be ensured that all systems are updated for their safety
alongside the new protocols.
    </p>
```

```
  </div>
</div>
</html>

<!DOCTYPE html>
<html>
  <title>IoT Details</title>
  <head><script src="main.js"></script></head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="style.css">
  <link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Montserrat">
  <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.mi
n.css">

  <body class="-black">

  <nav class="-sidebar -bar-block -small -hide-small -center">
    <a href="login.html" class="-bar-item -button -padding-large -hover-black">
      <i class="fa fa-sign-out -xxlarge"></i>
      <p>LOG OUT</p>
    </a>
    <a href="#main" class="-bar-item -button -padding-large -hover-black">
      <i class="fa fa-gears -xxlarge"></i>
      <p>MAIN DISPLAY</p>
    </a>
    <a href="#cam" class="-bar-item -button -padding-large -hover-black">
      <i class="fa fa-video-camera -xxlarge"></i>
      <p>CAMERA</p>
    </a>
    <a href="#iot" class="-bar-item -button -padding-large -hover-black">
      <i class="fa fa-envelope -xxlarge"></i>
      <p>SYSTEM LOGS</p>
    </a>
  </nav>

  <!-- Navbar on small screens -->
  <div class="-top -hide-large -hide-medium" id="myNavbar">
    <div class="-bar -black -opacity -hover-opacity-off -center -small">
      <a href="#home" class="-bar-item -button" style="width:25%
!important">LOGOUT</a>
      <a href="#main" class="-bar-item -button" style="width:25%
!important">MAIN</a>
      <a href="#cam" class="-bar-item -button" style="width:25%
!important">CAMERA</a>
      <a href="#iot" class="-bar-item -button" style="width:25% !important">SYSTEM
LOGS</a>
    </div>
  </div>
 <div class="-padding-large">

    <i class="fa fa-train -xxlarge -trainicon"></i>
    <!-- MAIN SECTION -->
```

```
    <div class="-content -justify -text-grey -padding-64" id="main">
      <h2 class="-text-light-grey">MAIN DISPLAY</h2>
      <hr style="width:200px;" class="-opacity">
    </div>
    <div class="-center -content -justify -text-grey -padding-64"
style="margin-top:0px;margin-bottom:40px;">
      <p class="-text-green" style="font-size:300px;margin:0px;" id="Speed">0</p>
      <p style="font-size:50px;">MPH</p>
      <div class="-third -section"><span class="-medium" id="EngTemp">0</span>
°F<br>Engine</div>
      <div class="-third -section"><span class="-medium" id="OutTemp">0</span>
°F<br>Outside</div>
      <div class="-third -section"><span class="-medium"
id="RPM">0</span><br>RPM</div>
      <div class="-third -section">X: <span class="-medium"
id="Lat">0</span><br>Latitude</div>
      <div class="-third -section">Y: <span class="-medium"
id="Long">0</span><br>Longitude</div>
      <div class="-third -section"><span class="-medium" id="RPMs">0</span>
MPH<br>RPM Speed</div>
    </div>
    <br>

    <div style = "margin: 3rem" class = "-center">
      <span id = "speedWarning" style="background-color:
green;padding-left:100px;padding-right:100px;padding-top:10px;padding-bottom:10px;"
>OK</span>
    </div>
    <div style = "margin: 3rem" class = "-center">
      <span id = "rpmWarning" style="background-color:
green;padding-left:100px;padding-right:100px;padding-top:10px;padding-bottom:10px;"
>OK</span>
    </div>
    <div style = "margin: 3rem" class = "-center">
      <span id = "tempWarning" style="background-color:
green;padding-left:100px;padding-right:100px;padding-top:10px;padding-bottom:10px;"
>OK</span>
    </div>
    <div class = "-center" style = "padding-bottom:15px;">
      <button type = "submit" onclick = "getRandInput()" class = "nicebutton">
TESTING </button>
    </div>
   <!-- CAMERA SECTION-->
    <div class="-content -justify -text-grey -padding-64" id="cam">
      <h2 class="-text-light-grey">CAMERA</h2>
      <hr style="width:200px;" class="-opacity">
      <div class="-center"><span id="container"><video autoplay="true"
id="videoElement"></video></span></div>
      <script>
            var video = document.querySelector("#videoElement");

            if (navigator.mediaDevices.getUserMedia) {
                navigator.mediaDevices.getUserMedia({ video: true })
            .then(function (stream) {
```

```
            video.srcObject = stream;
          })
          .catch(function (err0r) {
            console.log("Something went wrong!");
          });
      }
    </script>
  </div>


  <!-- STYSTEM LOGS SECTION -->
  <div class="-content -justify -text-grey -padding-64" id="iot">
    <h2 class="-text-light-grey" style="margin-bottom: 0px;">SYSTEM LOGS</h2>
    <hr style="width:200px;" class="-opacity">
    <table id="Logs" style="width:100%">
      <tr>
        <th>Date</th>
        <th>Time</th>
        <th>Speed</th>
        <th>RPM</th>
        <th>Eng. Temp.</th>
        <th>Out. Temp.</th>
        <th>Conductor</th>
      </tr>
      <tr>
        <td>18:06:00</td>
        <td>4-26-2021</td>
        <td>100</td>
        <td>100</td>
        <td>80</td>
        <td>100</td>
        <td>Example</td>
      </tr>
    </table>
    <br>
    <div class = "-center" style = "padding-bottom:15px;">
      <button type = "submit" onclick = "downloadData()" class = "nicebutton">
DOWNLOAD </button>
    </div>
  </div>


</div>

<footer class="-content -padding-64 -text-grey -center">
  Copyright © 2021 LittleEngineersThatCould. All rights rights reserved.
</footer>

</body>
</html>

// global variables:
var speed = 0;
var rpm = 0;
```

```
var longitude = 0;
var latitude = 0;
var etemp = 0;
var otemp = 0;
var clicked = false;

function check(form) {
    if(form.userid.value == "admin" && form.password.value == "password") {
        window.location.href ='main.html'
    }
    else {
        alert("Invalid Username or Password.")
    }
}

function getRandInput(){
    speed = Math.floor(Math.random() * 100);
    document.getElementById("Speed").innerHTML = speed;
    rpm = Math.floor(Math.random() * 1000);
    document.getElementById("RPM").innerHTML = rpm;
    longitude = Math.floor(Math.random() * 180);
    document.getElementById("Long").innerHTML = longitude;
    latitude = Math.floor(Math.random() * 90);
    document.getElementById("Lat").innerHTML = latitude;
    etemp = Math.floor(Math.random() * 190);
    document.getElementById("EngTemp").innerHTML = etemp;
    otemp = Math.floor(Math.random() * 100);
    document.getElementById("OutTemp").innerHTML = otemp;
    actualSpeed = Math.floor((2.89 * rpm / 1609) * 60);
    document.getElementById("RPMs").innerHTML = actualSpeed;

    throwWarnings();
    putInTable();
    gateCheck();
}

function putInTable(){
    let table = document.getElementById('Logs');
    let row = table.insertRow();

    var today = new Date();

    let dateCell = row.insertCell(0);
    dateCell.innerHTML =
(today.getMonth()+1)+'-'+today.getDate()+'-'+today.getFullYear();

    let timeCell = row.insertCell(1);
    timeCell.innerHTML =
today.getHours()+':'+today.getMinutes()+':'+today.getSeconds();

    let speedCell = row.insertCell(2);
    speedCell.innerHTML = speed;

    let rpmCell = row.insertCell(3);
```

```
        rpmCell.innerHTML = rpm;

        let engTCell = row.insertCell(4);
        engTCell.innerHTML = etemp;

        let outTCell = row.insertCell(5);
        outTCell.innerHTML = otemp;

        let condCell = row.insertCell(6);
        condCell.innerHTML = "Admin";
    }
function throwWarnings(){
    if(speed < 70){
        document.getElementById("speedWarning").innerHTML = "Speed Status: OK";
        document.getElementById("speedWarning").style.backgroundColor = "green";
        document.getElementById("speedWarning").style.color = "white";
        document.getElementById("speedWarning").style.padding = "10px 100px 10px
100px";
    }
    if(speed > 70){
        document.getElementById("speedWarning").innerHTML = "Speed Status: SLOW DOWN!
Speed is approaching maximum speed.";
        document.getElementById("speedWarning").style.backgroundColor = "yellow";
        document.getElementById("speedWarning").style.color = "black";
        document.getElementById("speedWarning").style.padding = "10px 100px 10px
100px";
    }
    if(speed > 90){
        document.getElementById("speedWarning").innerHTML = "Speed Status: BRAKE!
Speed is maximum! Slow Down!";
        document.getElementById("speedWarning").style.backgroundColor = "red";
        document.getElementById("speedWarning").style.color = "black";
        document.getElementById("speedWarning").style.padding = "10px 100px 10px
100px";
    }
if(etemp < 150){
        document.getElementById("tempWarning").innerHTML = "Temperature Status: OK";
        document.getElementById("tempWarning").style.backgroundColor = "green";
        document.getElementById("tempWarning").style.color = "white";
        document.getElementById("tempWarning").style.padding = "10px 100px 10px
100px";
    }
    if(etemp > 150){
        document.getElementById("tempWarning").innerHTML = "Temperature Status:
DANGER! Engine Temperature is approaching maximum.";
        document.getElementById("tempWarning").style.backgroundColor = "yellow";
        document.getElementById("tempWarning").style.color = "black";
        document.getElementById("tempWarning").style.padding = "10px 100px 10px
100px";
    }
    if(etemp > 180){
        document.getElementById("tempWarning").innerHTML = "Temperature Status:
WARNING! Engine is overheating!";
        document.getElementById("tempWarning").style.backgroundColor = "red";
```

```
        document.getElementById("tempWarning").style.color = "black";
        document.getElementById("tempWarning").style.padding = "10px 100px 10px
100px";
    }
 if(rpm < 750){
        document.getElementById("rpmWarning").innerHTML = "Slippage Status: OK";
        document.getElementById("rpmWarning").style.backgroundColor = "green";
        document.getElementById("rpmWarning").style.color = "white";
        document.getElementById("rpmWarning").style.padding = "10px 100px 10px 100px";
    }
    if(rpm > 750){
        document.getElementById("rpmWarning").innerHTML = "Slippage Status: DANGER!
RPM is approaching dangerous speeds.";
        document.getElementById("rpmWarning").style.backgroundColor = "yellow";
        document.getElementById("rpmWarning").style.color = "black";
        document.getElementById("rpmWarning").style.padding = "10px 100px 10px 100px";
    }
    if(rpm > 900){
        document.getElementById("rpmWarning").innerHTML = "Slippage Status: WARNING!
RPM dangerously high! Slippage likely!";
        document.getElementById("rpmWarning").style.backgroundColor = "red";
        document.getElementById("rpmWarning").style.color = "black";
        document.getElementById("rpmWarning").style.padding = "10px 100px 10px 100px";
    }
let isMoving = Math.floor(Math.random() * 2);
    let objectDist = Math.floor(Math.random() * 50);
    if(isMoving == 1) { // object IS moving
        if(objectDist <= 10) {
            document.getElementById("objectDetection").innerHTML = "Object Status:
BRAKE! You're about to hit a moving object!";
            document.getElementById("objectDetection").style.backgroundColor = "red";
            document.getElementById("objectDetection").style.color = "black";
        }
        else if(objectDist <= 30) {
            document.getElementById("objectDetection").innerHTML = "Object Status:
SLOW DOWN! There's a moving object close by!";
            document.getElementById("objectDetection").style.backgroundColor =
"yellow";
            document.getElementById("objectDetection").style.color = "black";
        }
        else{
            document.getElementById("objectDetection").innerHTML = "Object Status:
OK!";
            document.getElementById("objectDetection").style.backgroundColor =
"green";
            document.getElementById("objectDetection").style.color = "white";
        }
    }
  else { // object is NOT moving
        if(objectDist <= 5) {
            document.getElementById("objectDetection").innerHTML = "Object Status:
BRAKE! You're about to hit a standing object!";
            document.getElementById("objectDetection").style.backgroundColor = "red";
            document.getElementById("objectDetection").style.color = "black";
```

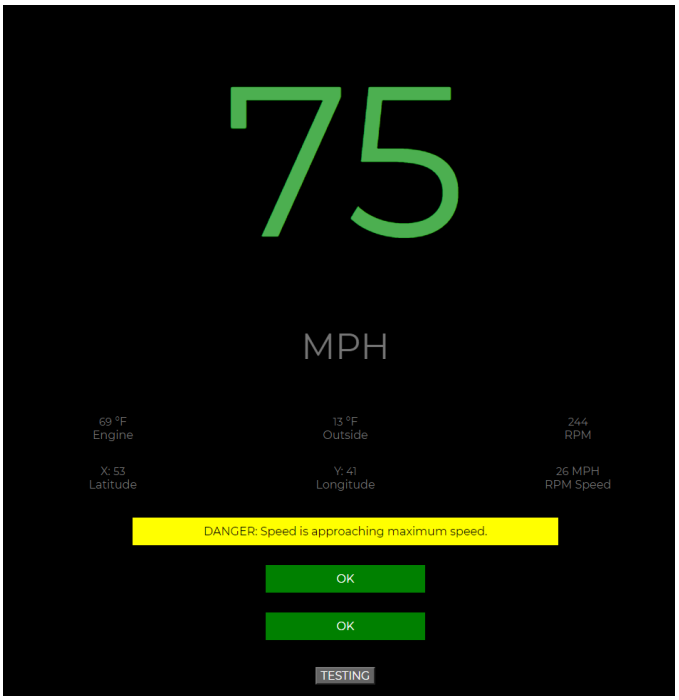
```
        }
        else if(objectDist <= 10) {
            document.getElementById("objectDetection").innerHTML = "Object Status:
SLOW DOWN! There's a standing object close by!";
            document.getElementById("objectDetection").style.backgroundColor =
"yellow";
            document.getElementById("objectDetection").style.color = "black";
        }
        else{
            document.getElementById("objectDetection").innerHTML = "Object Status:
OK!";
            document.getElementById("objectDetection").style.backgroundColor =
"green";
            document.getElementById("objectDetection").style.color = "white";
        }
    }
}
   function downloadData(){
       var table = document.getElementById("Logs");
       var rows =[];

       for(var i=0,row; row = table.rows[i];i++){
           column1 = row.cells[0].innerText;
           column2 = row.cells[1].innerText;
           column3 = row.cells[2].innerText;
           column4 = row.cells[3].innerText;
           column5 = row.cells[4].innerText;
           column6 = row.cells[5].innerText;
           column7 = row.cells[6].innerText;

           rows.push(
               [
                   column1,
                   column2,
                   column3,
                   column4,
                   column5,
                   column6,
                   column7
               ]
           );

           }
           csvContent = "data:text/csv;charset=utf-8,";
           rows.forEach(function(rowArray){
               row = rowArray.join(",");
               csvContent += row + "\r\n";
           });

           var encodedUri = encodeURI(csvContent);
           var link = document.createElement("a");
           link.setAttribute("href", encodedUri);
           link.setAttribute("download", "HTR_Logs.csv");
           document.body.appendChild(link);
```

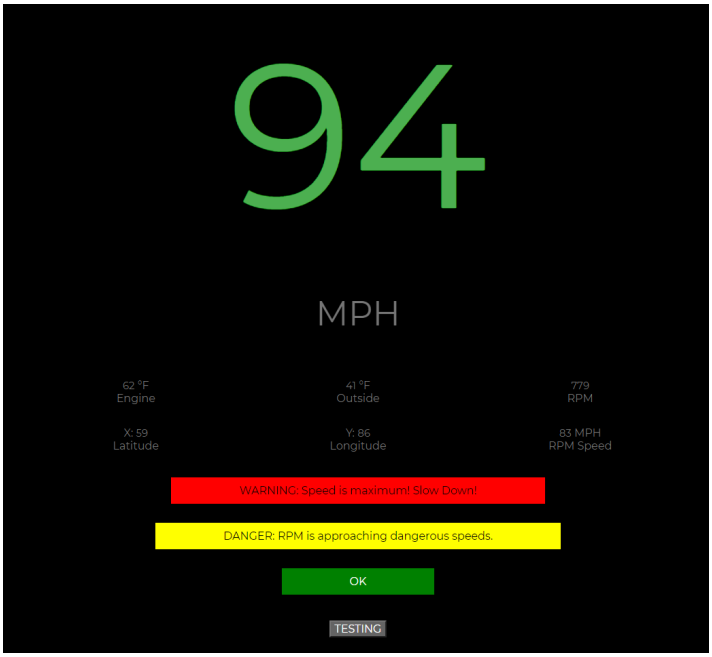
```
        link.click();
}

function gateCheck() {
    let distance = Math.floor(Math.random() * 10);
    const audio = new Audio("train_horn.mp3");
    if(distance == 1){
        audio.play();
    }

}
```

7: TESTING

7.1 TEST 1 - SPEED ERROR

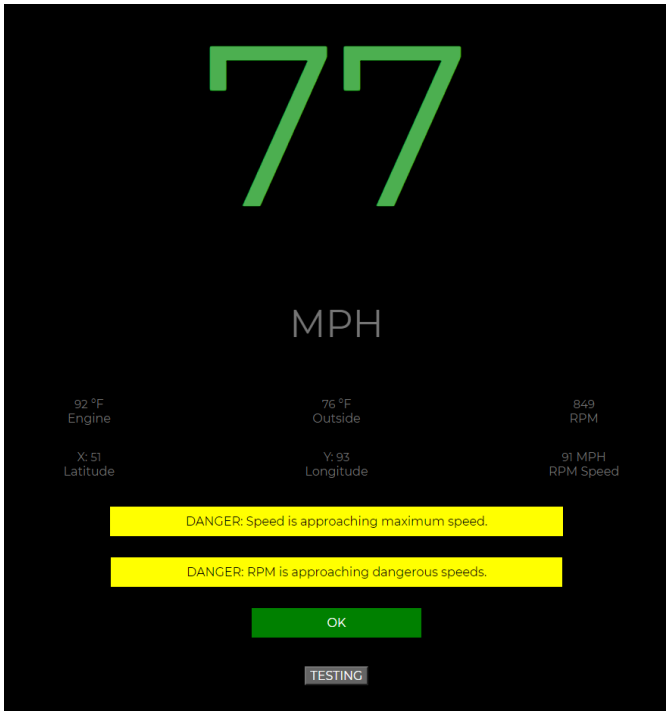
When the speed reaches a threshold of in between 70 mph and 90 mph, the following error will be thrown in yellow:



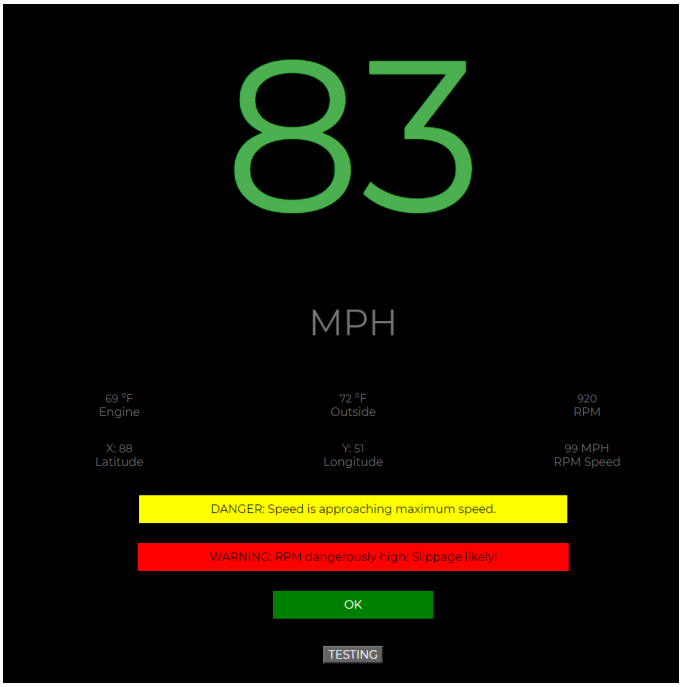


7.2 TEST 2 - RPM ERROR

When the RPM reaches a threshold of greater than 1.5 times the gps speed, the following error will be thrown in yellow:

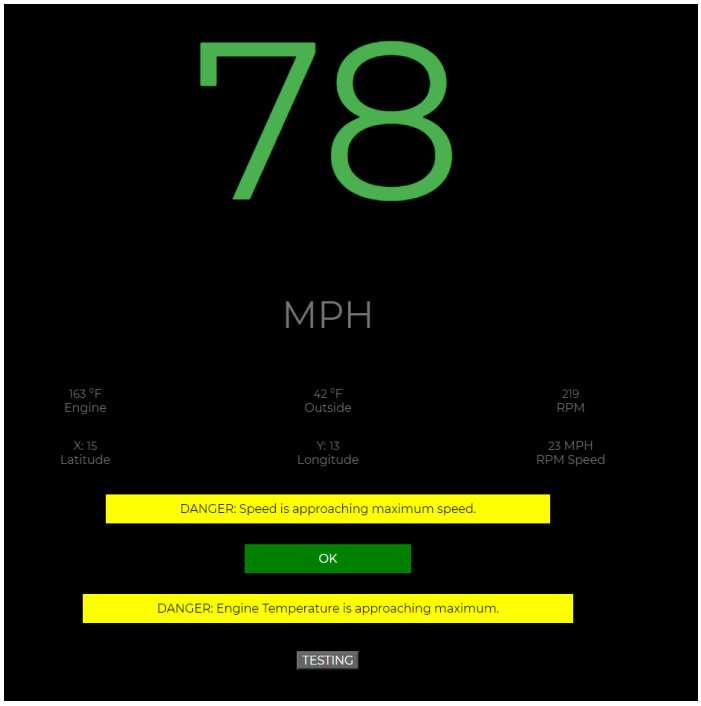


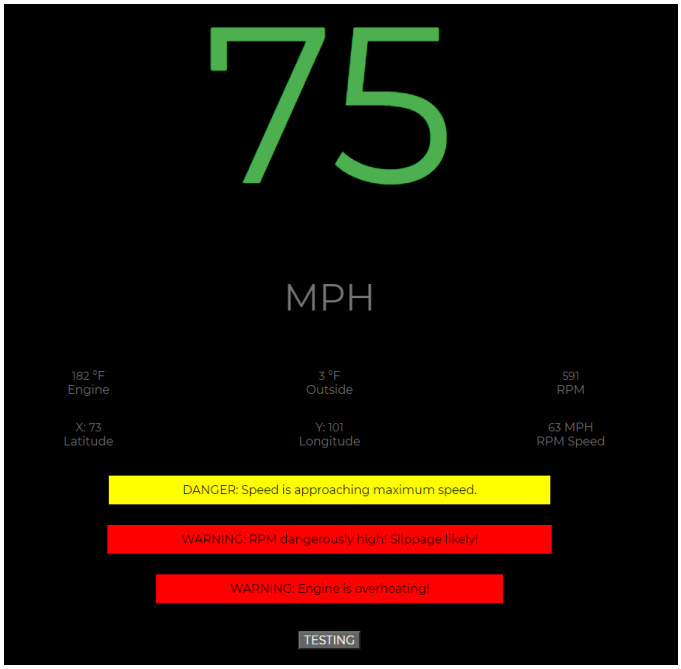
When the RPM reaches a threshold of above 2 times the gps speed, the following error will be thrown in red:



7.3 TEST 3 - ENGINE TEMPERATURE ERROR

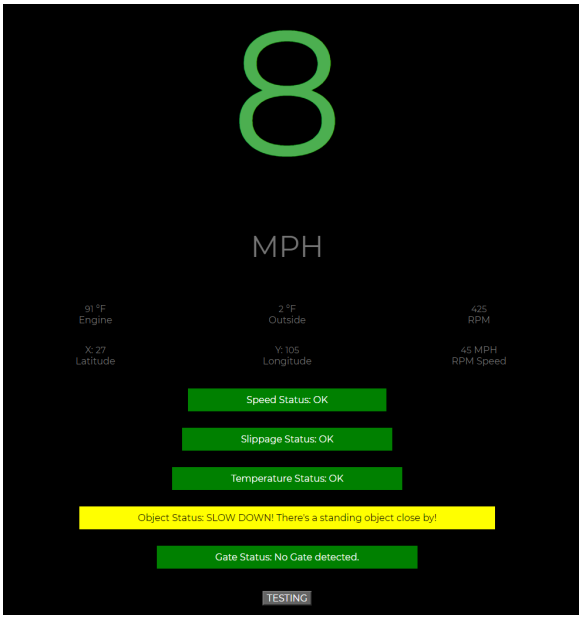
When the engine temperature reaches a temperature between 150 and 180 degrees fahrenheit, the following error is thrown in yellow:



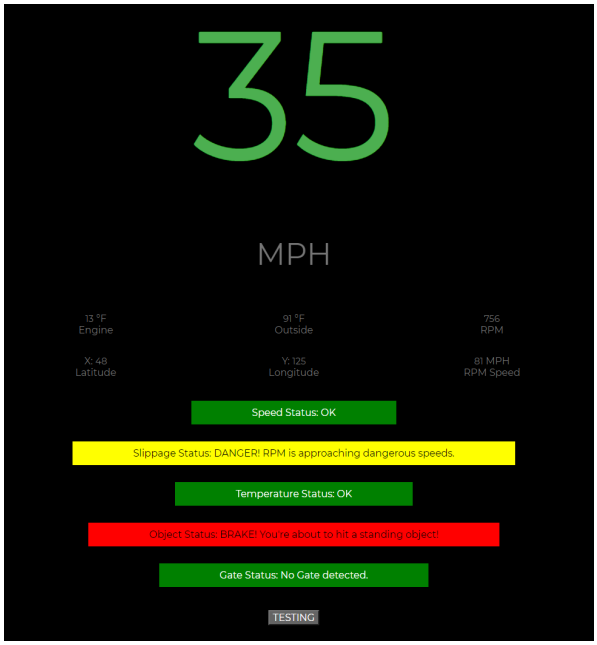


7.4 TEST 4 - OBJECT ERROR

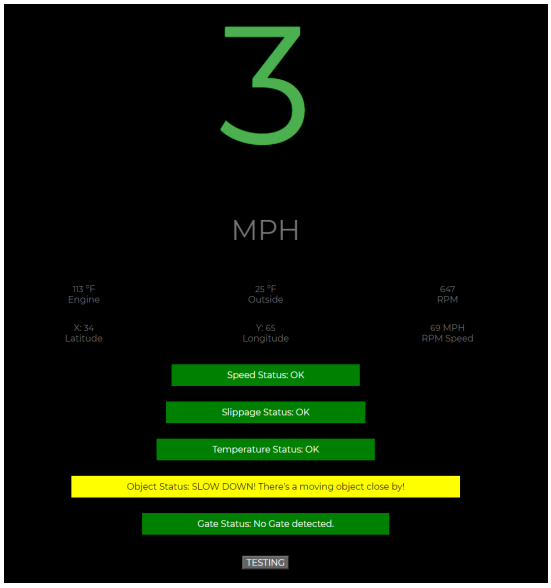
When a standing object is in between 10 and 30 miles away from the train, the following error will be thrown in yellow:



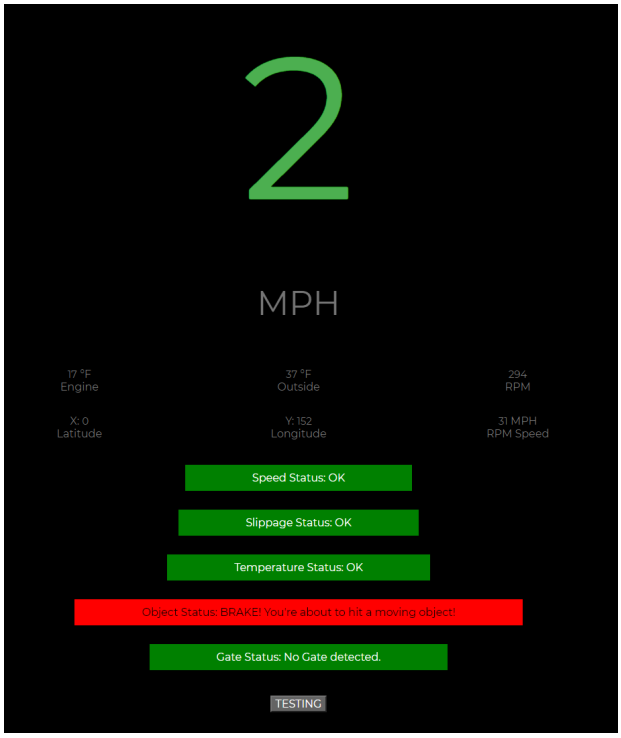
When a standing object is 10 miles within a train, the following error will be thrown in red:



When a moving object is within 5 and 10 miles away from the train, the following error will be thrown in yellow:

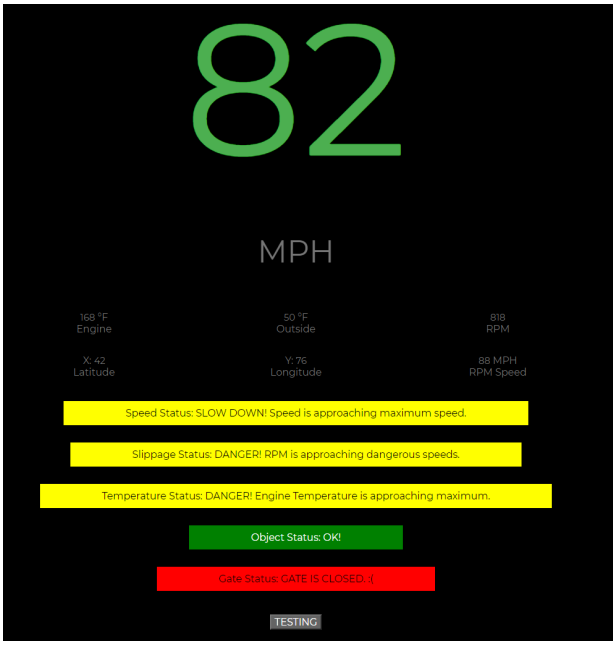


When a moving object is within 5 miles from the train, the following error will be thrown in red:

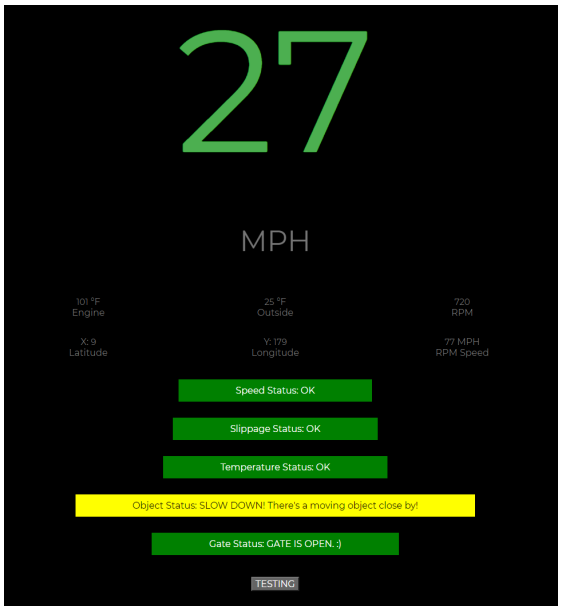


7.5 TEST 5 - GATE ERRORS

When a gate is closed within 1 mile from the train, the following warning will be thrown in red:



When a gate is open within 1 mile from the train, the following warning will be thrown in red:



8: RESOURCES

1. Main Source - *Software Engineering: A Practitioner's Approach (9th Edition)*
2. Object Oriented Programming Architecture (Section 5.1)
3. Data Center Architecture (Section 5.1)
4. Data Flow Architecture (Section 5.1)
5. Return and Call Architecture (Section 5.1)