Team 1

Hug the Rails IoT Project

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

Table of Contents

Section 1: Introduction

1.1 Problem Statement

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

1.2 Stakeholders and Users

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

1.3 Importance and Values

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

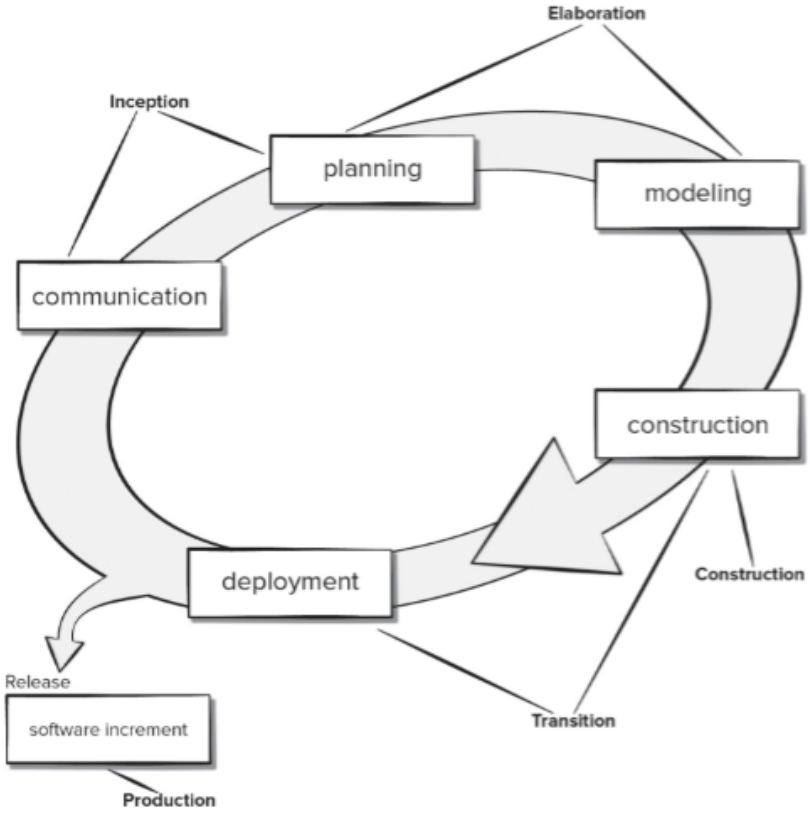
1.4 Expected Delivery

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

1.5 Approach

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

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



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

Section 2: Overview

2.1 Define IoT

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

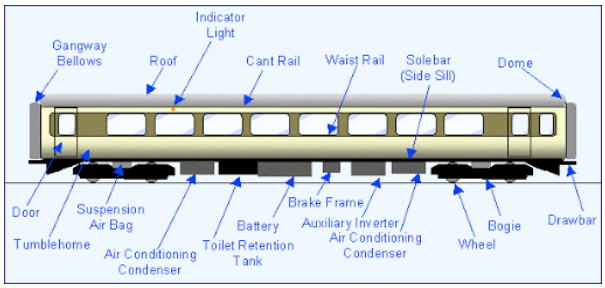
2.2 Define the Problem

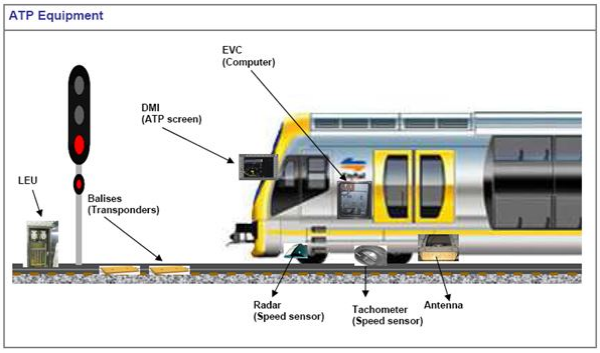
The operation of a train is often dependent on wifi network availability to receive data about environmental, travel, and traffic conditions. Based on those data, the train operator makes decisions about how the train operates. The purpose of this project is to use IoT to make those decisions locally,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

2.3 Explain how IoT can Solve the Problem

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

2.4 Overview of Architecture and Components





Components:

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

Architecture:

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

Section 3: Requirements

3.1 Non-Functional Requirements

3.1.1 Reliability Requirements

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

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

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

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

3.1.2 Performance Requirements

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

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

3.1.3 Security Requirements

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

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

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

3.1.4 Operating System Requirements

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

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

3.2 Functional Requirements

3.2.1 Detect the Wind Speed

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

R-13: If the speed of the wind reached a potentially hazardous level (55+ mph), the operator shall receive a notification on the IoT interface with a recommendation to slow the train down to a specified safer speed based on the particular wind speed detected.

3.2.2 Detect the Amount of Precipitation

R-14: The precipitation sensor shall detect the amount of precipitation in inches that is in the area in which the train is operating.

R-15: If the amount of precipitation reaches a potentially hazardous level (0.3+ inches), the operator shall receive a notification on the IoT Interface with a recommendation to slow the train down to a specified safer speed based on the particular amount of rain.

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

R-16: The radar sensor shall detect whether there is a stationary object in the path of the train and how far away it is

R-17: The reading from the radar sensor shall be passed to the IoT Engine.

R-18: If the distance between the train and the object is less than 10,000 feet the operator shall be notified, via the interface, of the object's distance and if the object that was detected requires a speed change for the train or for the brakes to be applied.

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

R-19: The radar sensor shall detect any other objects on the rails, and calculate the distance and speed of said object.

R-20: The distance reading from the radar sensor shall be passed to the IoT Engine.

R-21: If the distance between the train and the object is less than 10,000 feet the operator shall be notified, via the interface, of the object's distance and if the object that was detected requires a speed change for the train or for the brakes to be applied.

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

R-22: The camera sensor shall be used to detect if the gate is open or closed, whether the red lights are flashing, the distance the train is away from the gate, and the speed.

R-23: The information from the sensor shall be passed to the IoT Engine which will analyze the data.

R-24: If the gate is not down, the interface shall notify the operator that the gate is open.

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

R-24: The sensor to detect wheel slippage shall be used to detect the amount of slippage occurring in the wheels, the wheels RPM, and based on this data the conductor will be informed of any speed changes that should be made or if the conductor should brake.

R-25: The sensor shall pass the RPM and wheel circumference to the IoT Engine which shall convert and analyze the data and use the GPS location and train speed to determine wheel slippage.

R-26: If the difference between the projected speed and calculated wheel speed is between 20-50 RPM, the IoT Engine shall inform the operator of wheel slippage using the IoT interface and display a caution message in orange.

R-27: If the difference between the projected speed and calculated wheel speed is greater than 50, the IoT Engine shall inform the operator of wheel slippage using the IoT interface and display an extreme caution message and the speed that should not be exceeded in red.

3.2.7 Sensors will store their readings in a local log

R-28: Every reading taken by the sensors shall be stored in a local log.

R-29: The size of the local log shall be 5TB.

R-30: The local log shall be uploaded to the cloud when wifi connection is available, then cleared from the local database.

R-31: The log shall only be accessible with valid login information.

Section 4: Requirements Modeling

4.1 Use Cases

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

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

Exceptions:

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

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

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

Exceptions:

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

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

1. Rain gauge sensor sends data to IoT Engine.
2. IoT displays “normal condition” if rain level is less than 0.3 inches.
3. IoT displays “hazardous condition” if rain level is greater than or equal to 0.3 inches and the train is moving.
    a. IoT Engine displays a warning message on the IoT interface with recommended speed changes:
        i. Reduce speed to half of current speed if 0.3 inches ≤ rain level < 0.6 inches
        ii. Reduce speed to ⅓ of current speed if rain level ≥ 0.6 inches
    b. Train Operator acknowledges the message on the interface by pressing “okay”.
    c. Log File records operator response with timestamp.

Exceptions:

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

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

2. Anemometer sends data to IoT Engine.
3. IoT displays “normal condition” if wind level is less than 55mph.
4. IoT displays “hazardous condition” if wind level is greater than or equal to 55mph and the train is moving.
    a. IoT Engine displays a warning message on the IoT interface with recommended speed changes:
        i. Reduce speed to half of current speed if 55mph ≤ wind speed < 65mph
        ii. Reduce speed to ⅓ of current speed if wind speed ≥ 65mph
    b. Train Operator acknowledges the message on the interface by pressing “okay”.
    c. Log File records operator response with timestamp.

Exceptions:

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

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

1. Wheel Slippage Sensor sends data to IoT Engine.
2. IoT displays “normal condition” if wheel slippage is less than 20 RPM.
3. IoT displays “hazardous condition” if wind level is greater than or equal to 20 RPM and the train is moving.
    a. IoT Engine displays a warning message on the IoT interface with recommended speed changes:
        i. Reduce speed to half of current speed if 20RPM ≤ wheel slippage < 50RPM
        ii. Reduce speed to ⅓ of current speed if wheel slippage ≥ 50RPM
    b. Train Operator acknowledges the message on the interface by pressing “okay”.
    c. Log File records operator response with timestamp.

Exceptions:

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

Use Case: Moving Object Detection
Use Case No. : 4.1.6
Primary Actor: Radar Sensor
Secondary Actor: Train Operator, IoT Engine, IoT Display, Log File
Goal: Notify train operator on the IoT interface in the case of moving object ahead
Preconditions: IoT Display is activated
Trigger: Radar sensor is running
Scenario:

1. Radar Sensor sends data to IoT Engine.
2. IoT displays “normal condition” if no moving object detected ahead
3. IoT displays “hazardous condition” if moving object is detected ahead
    a. IoT Engine displays a warning message on the IoT interface with recommended action to stop the train
    b. Train Operator acknowledges the message on the interface by pressing “okay”.
    c. Log File records operator response with timestamp.

Exceptions:

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

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

1. Radar Sensor sends data to IoT Engine.
2. IoT displays “normal condition” if no stationary object detected ahead
3. IoT displays “hazardous condition” if stationary object is detected ahead
    a. IoT Engine displays a warning message on the IoT interface with recommended action to stop the train
    b. Train Operator acknowledges the message on the interface by pressing “okay”.
    c. Log File records operator response with timestamp.

Exceptions:

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

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

1. Camera Sensor sends data to IoT Engine.
2. IoT displays “normal condition” if closed gate detected ahead
3. IoT displays “hazardous condition” if open gate detected ahead
    d. IoT Engine displays a warning message on the IoT Display with recommended action to stop the train
    e. Train Operator acknowledges the message on the IoT Display by pressing “okay”.
    f. Log File records operator response with timestamp.

Exceptions:

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

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

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

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

1. Technician enters username/password into IoT Engine
    a. Technician's login is valid and access is granted to the logs
2. Technician will access the Log File
    a. Technician can upload Log File to Cloud Log if there is wifi connection

Exceptions:

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

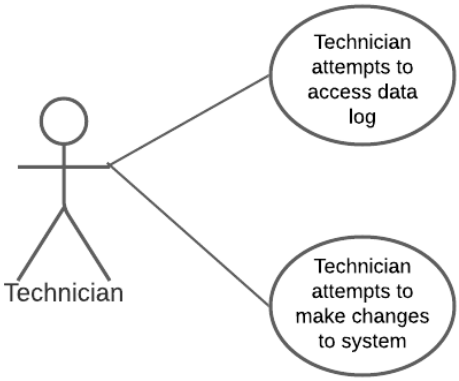
Use Case: Technician make changes to the system
Use Case No. : 4.1.10
Primary Actor: Technician
Secondary Actor: IoT Engine, IoT Display
Goal: Technician makes successful changes to the IoT Engine
Preconditions: Train is on, technician has valid clearance, and there is a wifi connection
Trigger: Technician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

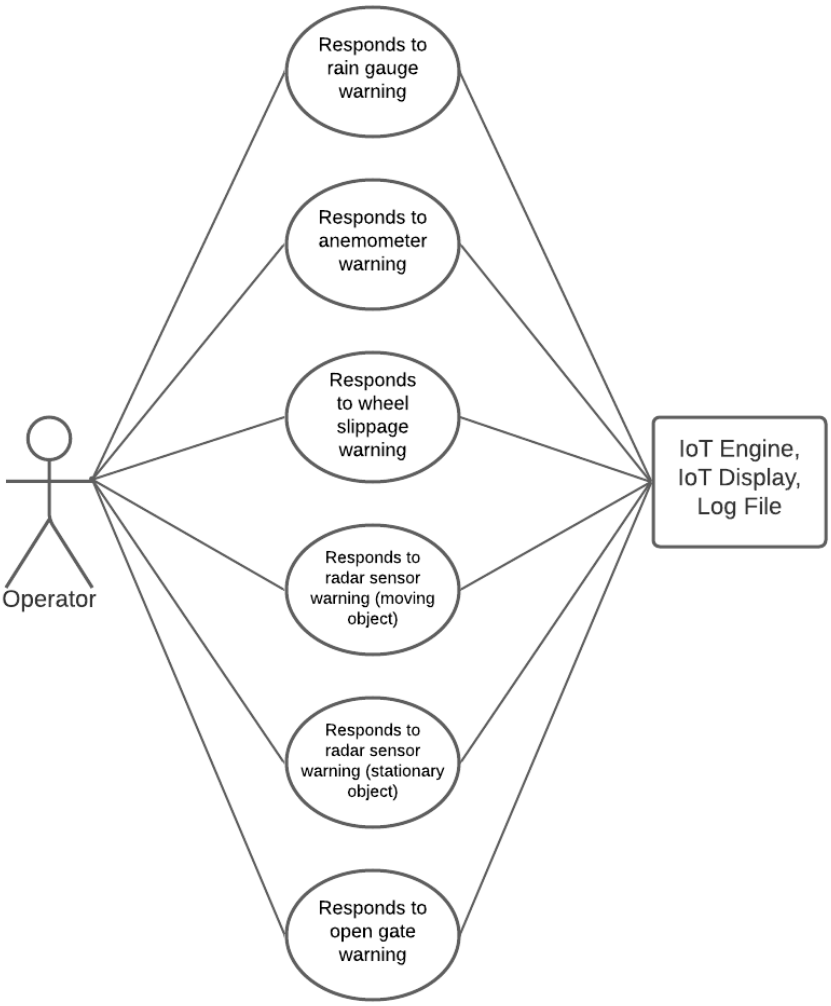
1. Technician logs into IoT Engine either onboard or remotely and selects "update system"
    a. IoT will display "failed to connect" message if there is no wifi connection on board
2. Technicians will upload changes to the system.

Exceptions:

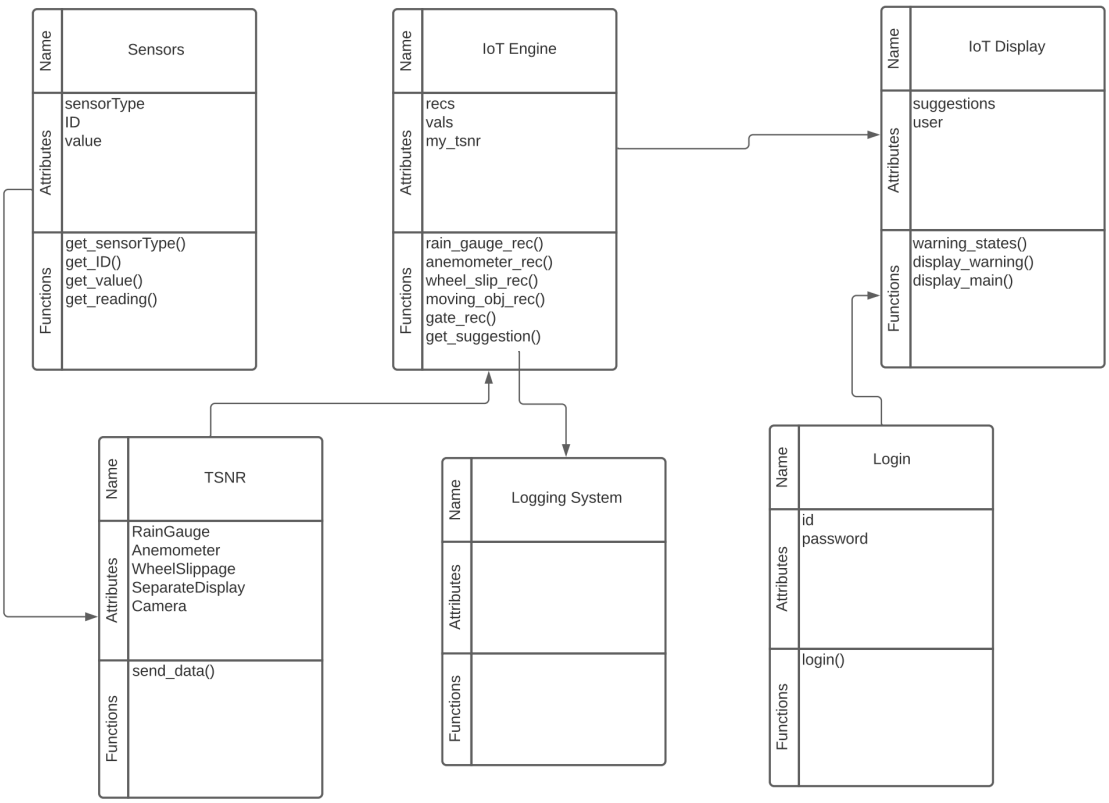
1. If wifi connection is lost part way through upload, all changes are reverted
2. If technician does not have valid clearance or passes an invalid username/password combo they are not allowed access to system
3. If there is no wifi connection, updates are not possible

4.2 Use Case Diagram





4.3 Class-Based Modeling



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

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

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

The IoT Display will allow the conductor to see all the data from the sensors. The display will have separate sections for each of the sensors, which allows the conductor to access data

from every hazardous condition that can be detected with the sensors. Each section will also turn green, orange, or red; depending on the level to which the conditions are hazardous.

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

4.4 CRC Modeling/Cards

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

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

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

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

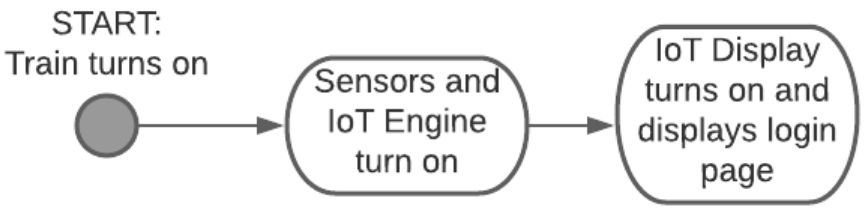
Login
Description: Authenticates operator or technician login information.
Responsibilities: ● Authenticate user ID and password ● Display correct display for operator

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

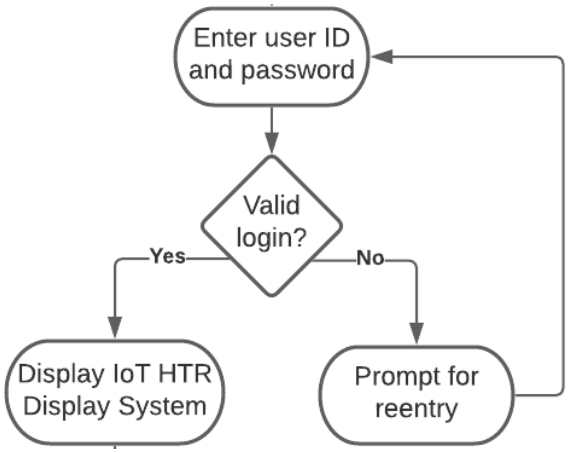
• Display all data collected from the sensors	or technician	

4.5 Activity Diagram

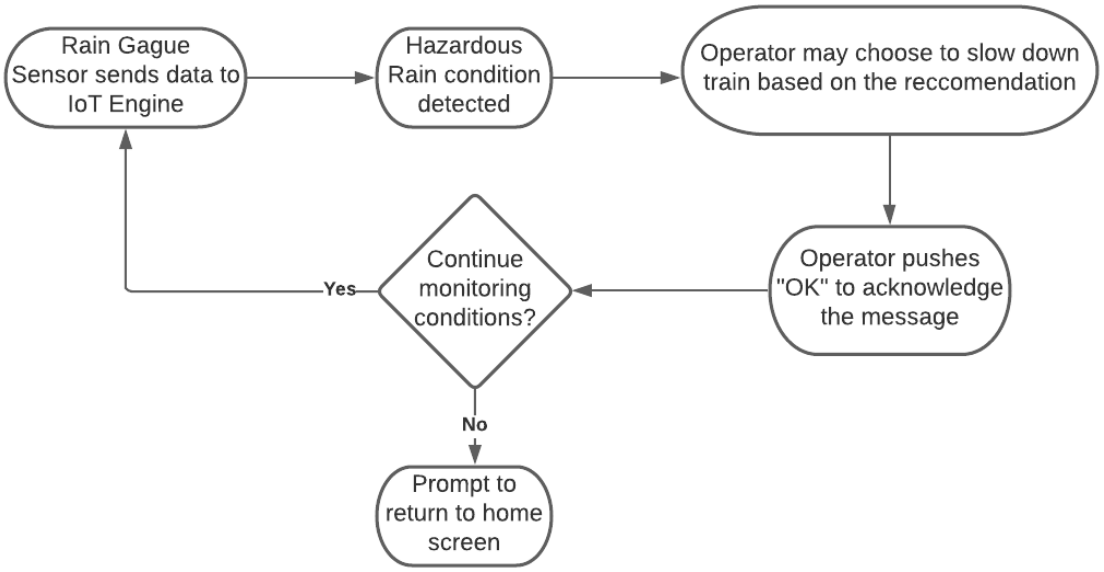
Use Case 4.1.1



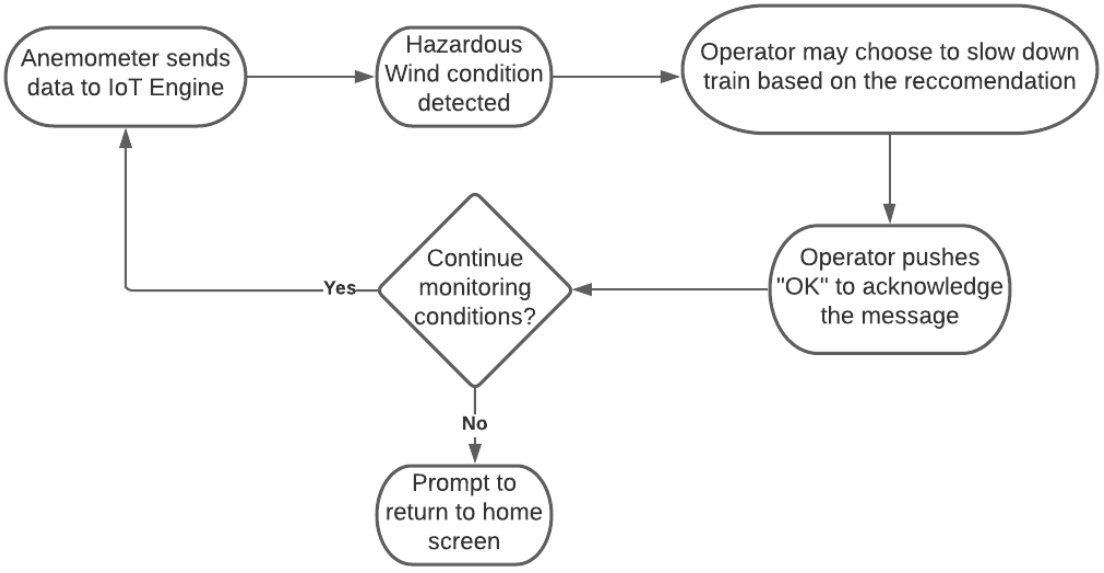
Use Case 4.1.2



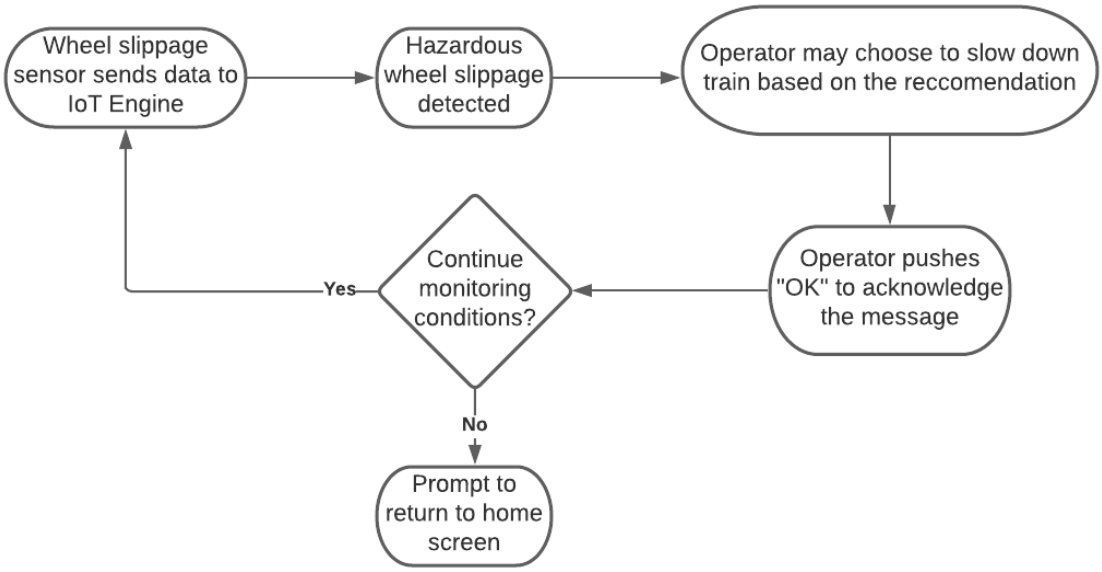
Use Case 4.1.3



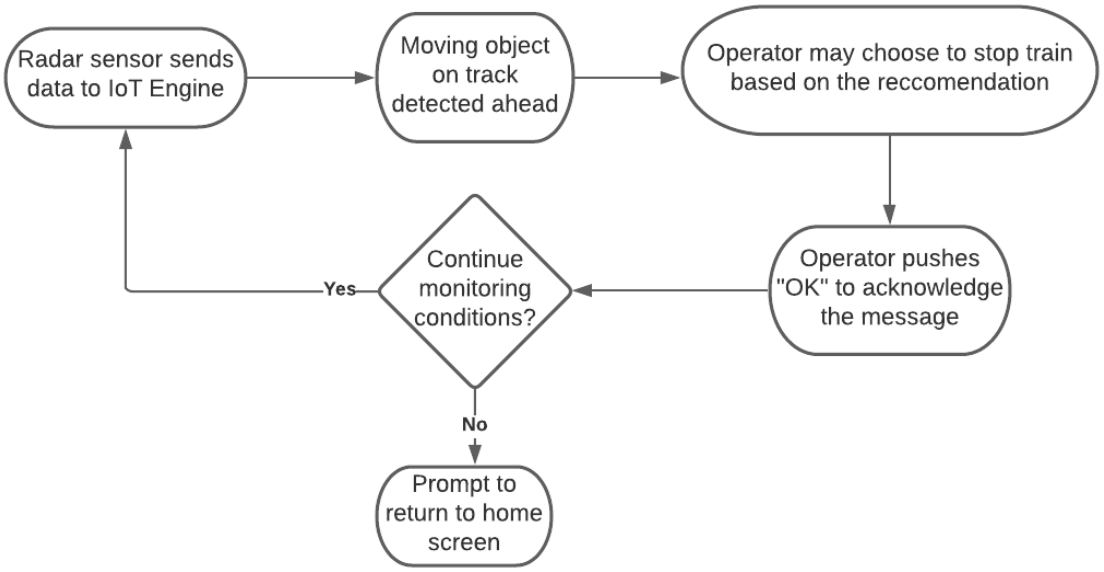
Use Case 4.1.4



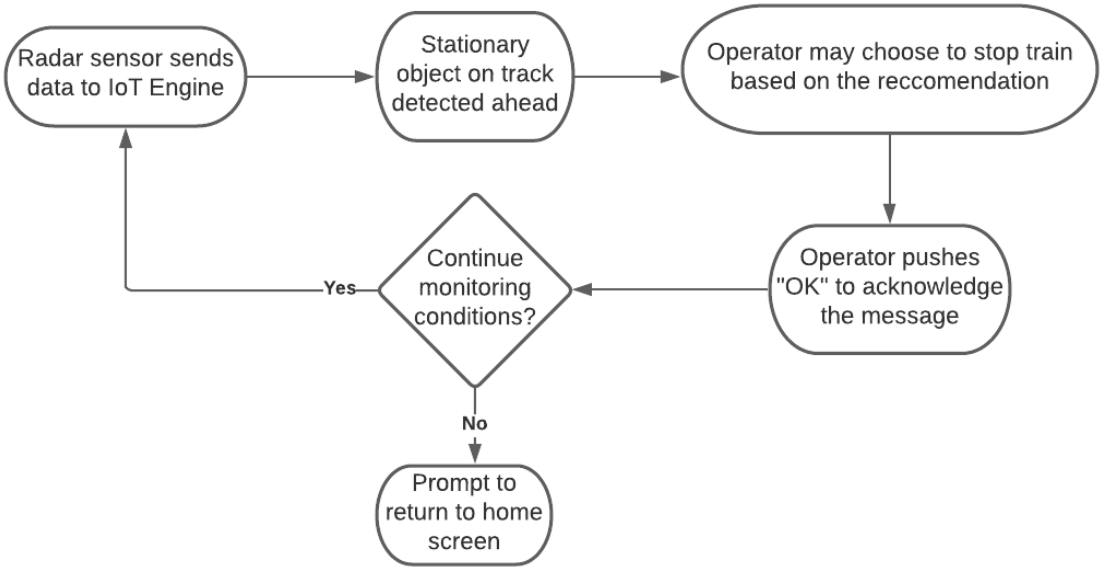
Use Case 4.1.5


Wheel slippage sensor sends data to IoT Engine
Hazardous wheel slippage detected
Operator may choose to slow down train based on the reccomendation
Operator pushes "OK" to acknowledge the message
Continue monitoring conditions?
Yes
No
Prompt to return to home screen

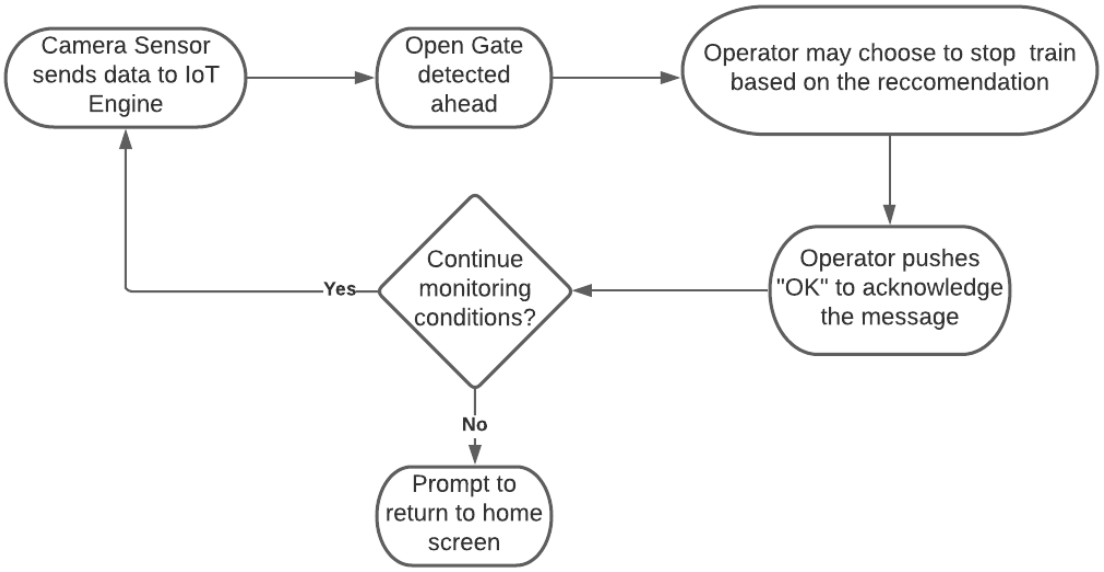
Use Case 4.1.6



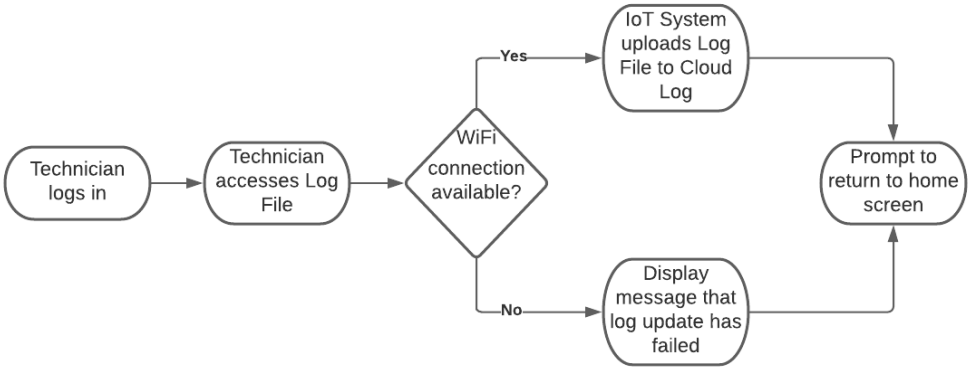
Use Case 4.1.7



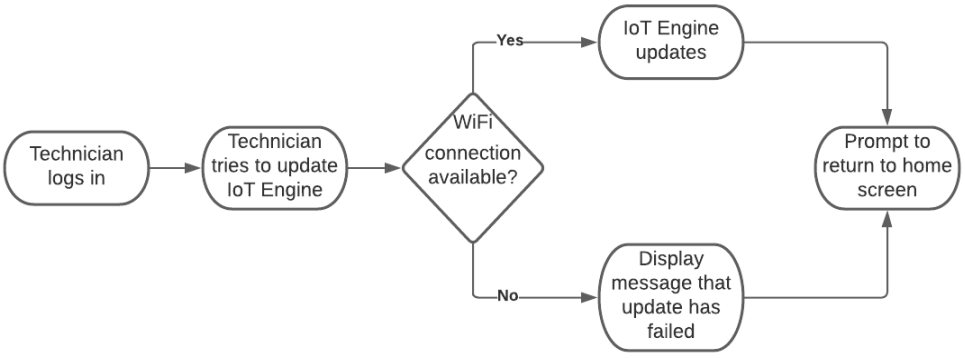
Use Case 4.1.8



Use Case 4.1.9

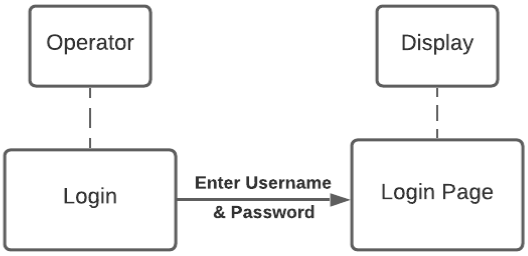


Use Case 4.1.10

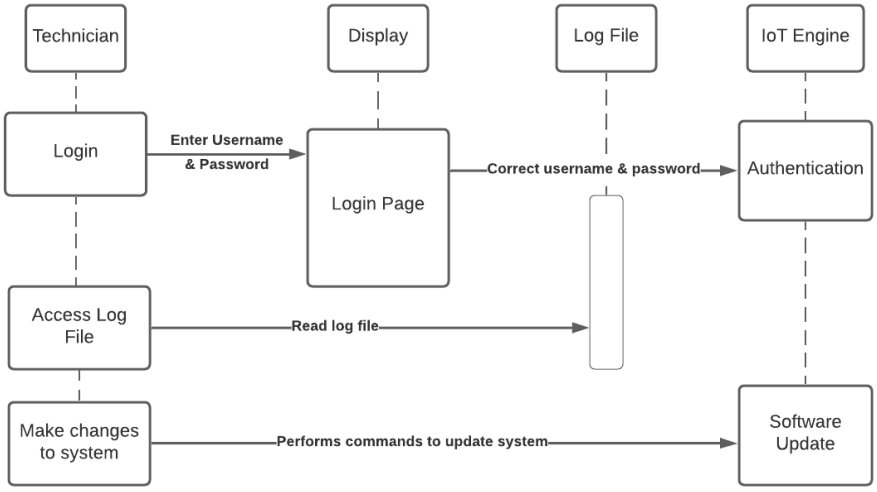


4.6 Sequence Diagrams

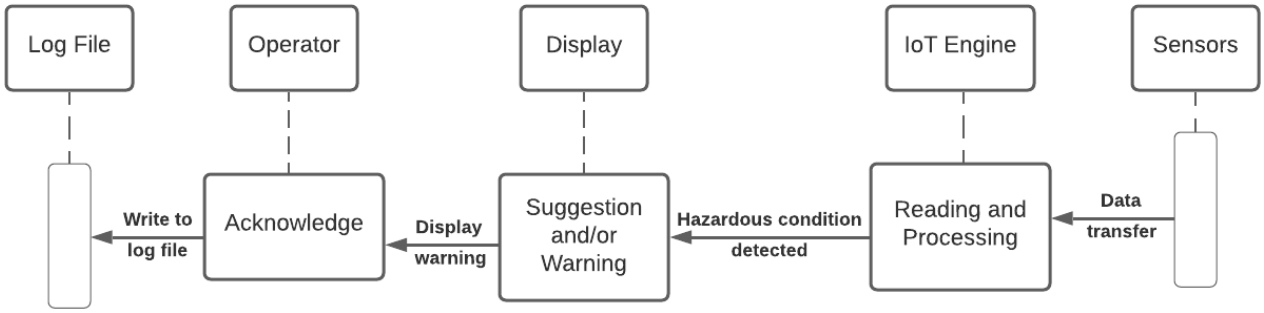
Operator Sequence Diagram:



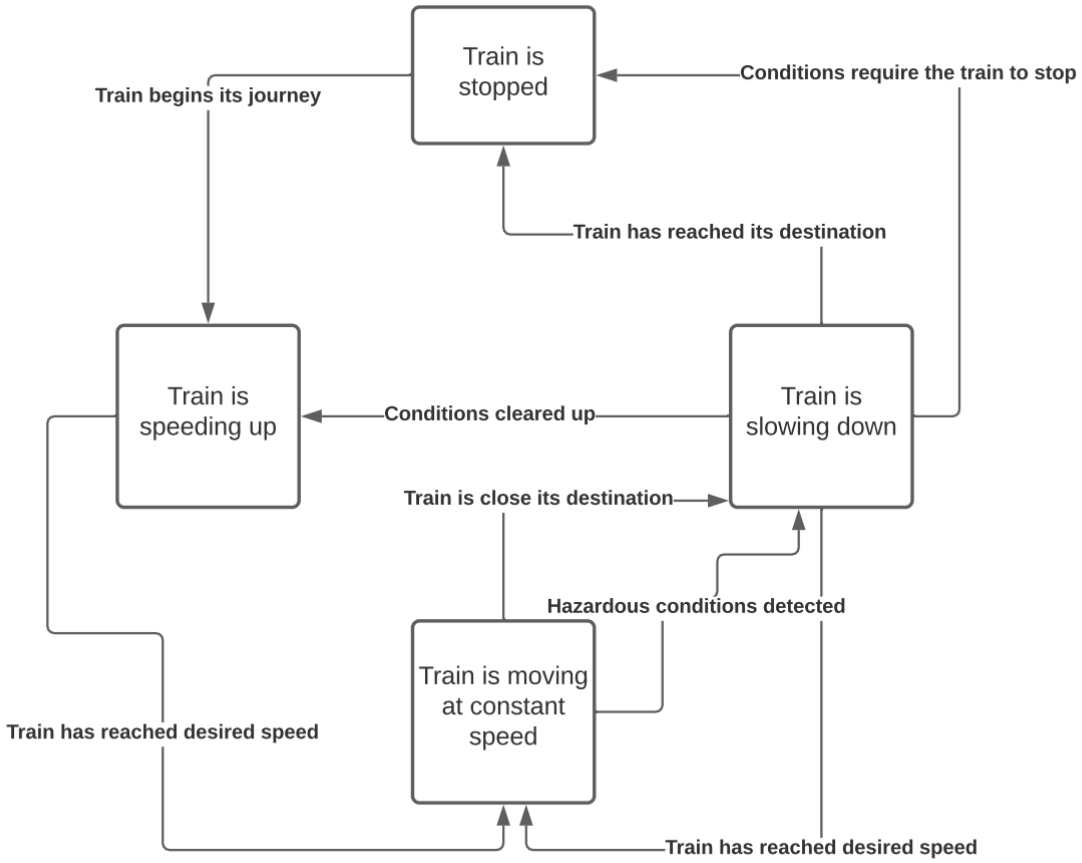
Technician Sequence Diagram:



Sensor Sequence Diagram:



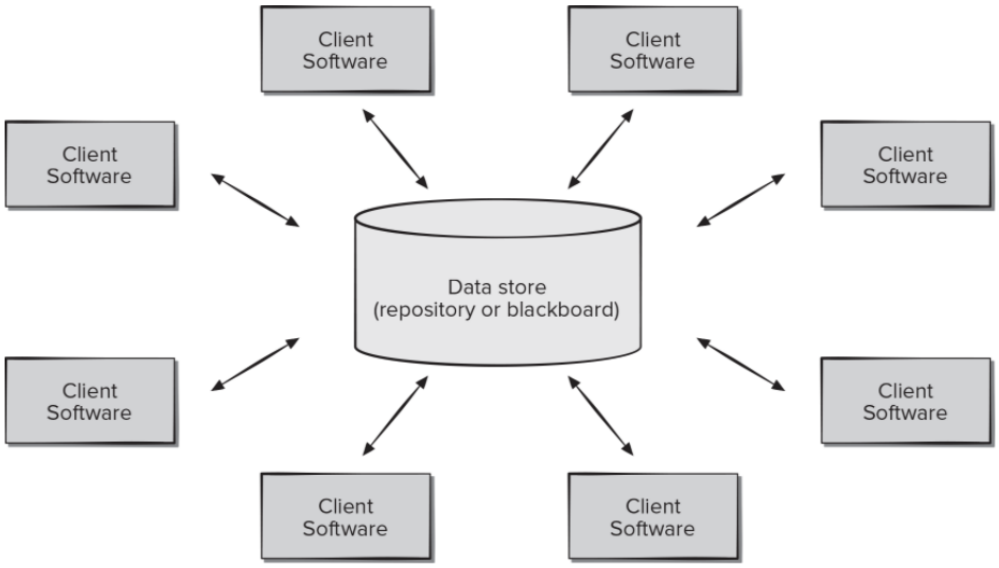
4.7 State Diagram



Section 5: Software Architecture

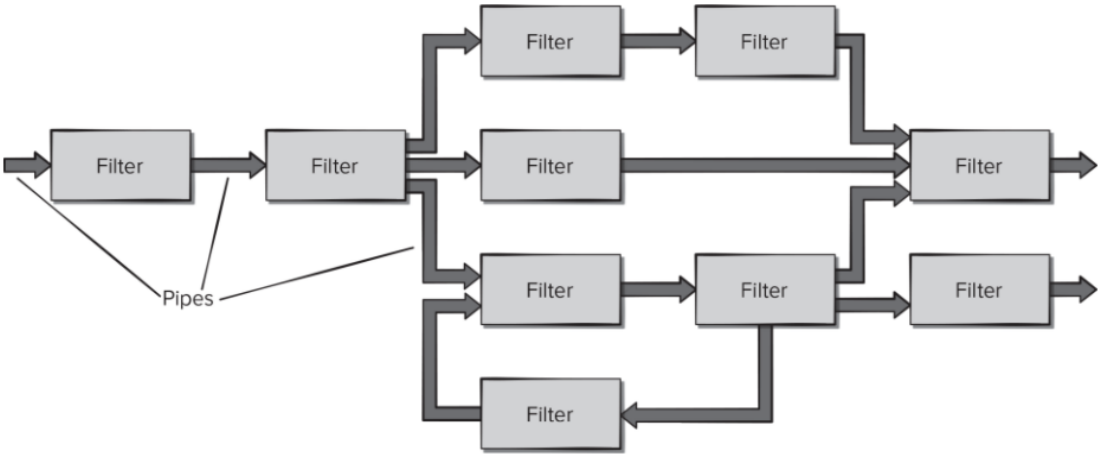
5.1 Software Architecture Models

5.1.1 Data-Centered Architecture



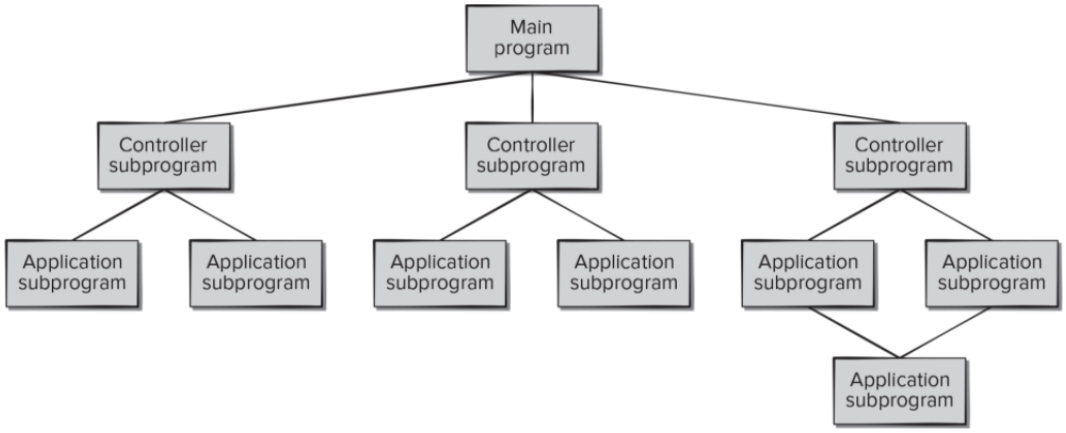
This architectural style is centered around a data store with independent components accessing the data store to update or modify the data. This is not a feasible architecture for IoT Hug the Rails because each sensor makes the system slower and there could be real time delay issues. Also, the data store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

5.1.2 Data-Flow Architecture



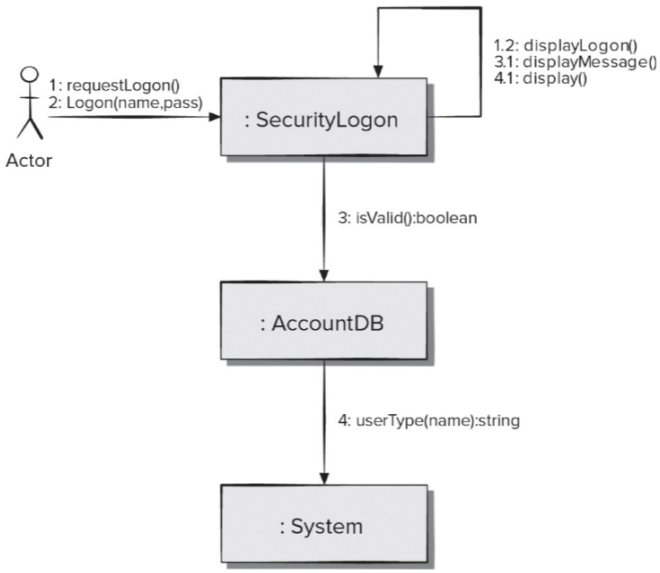
Data-Flow architecture is used when input data is to be manipulated and changed through computations. Each filter acts independently but expects some form of data as an input. The very first filter on the left hand side would be the Time Sensitive Network Router, in order to detect information from the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

5.1.3 Call-and-Return Architecture



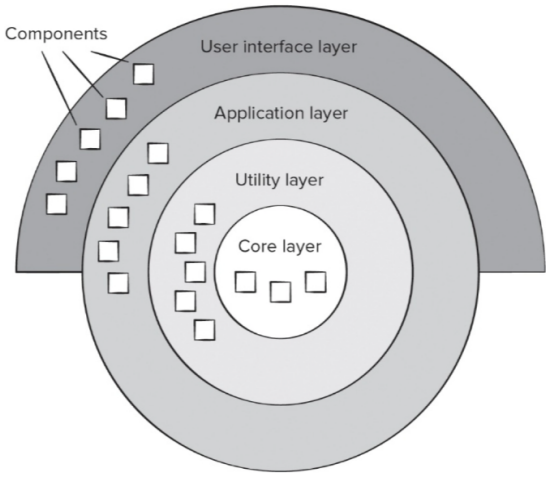
There are two categories of call-and-return architecture: main program/subprogram and remote procedure call. Both of these are used for a structure that can be easy to adjust and scale. This could be a feasible option for IoT Hug the Rails because the main program could call other functions to determine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

5.1.4 Object-Oriented Architecture



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

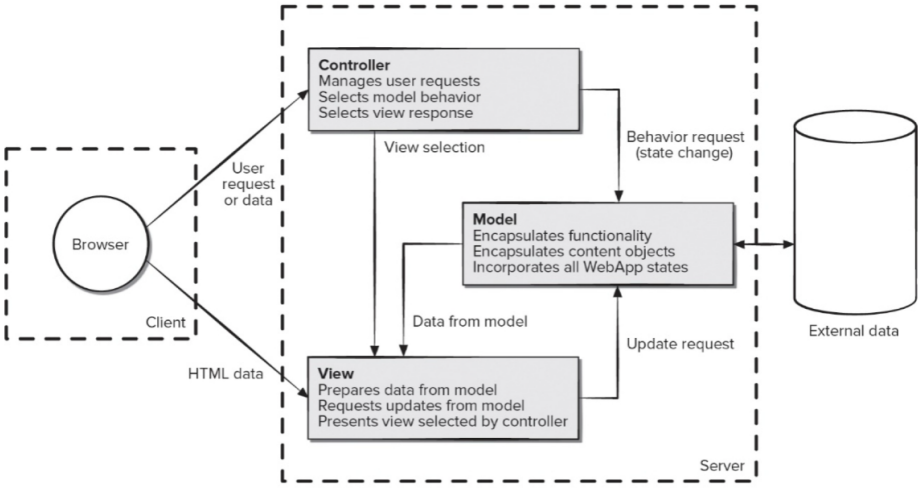
5.1.5 Layered Architecture



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

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

5.1.6 Model-View-Controller Architecture

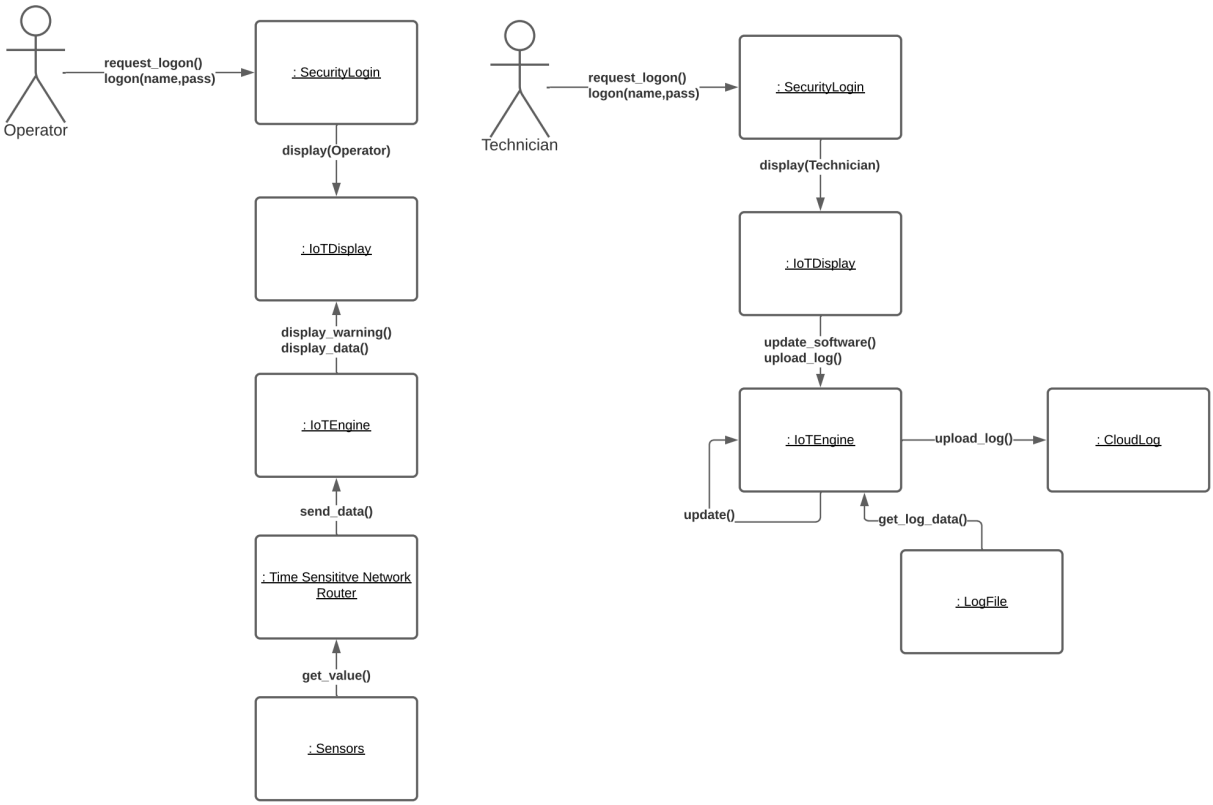


The MVC architecture model is a mobile infrastructure model, mainly used in web development. It has three components: the model, the view, and the controller. If we process data from right to left, the external data takes in the sensor data that has gone through the Time Sensitive Network Router.
This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

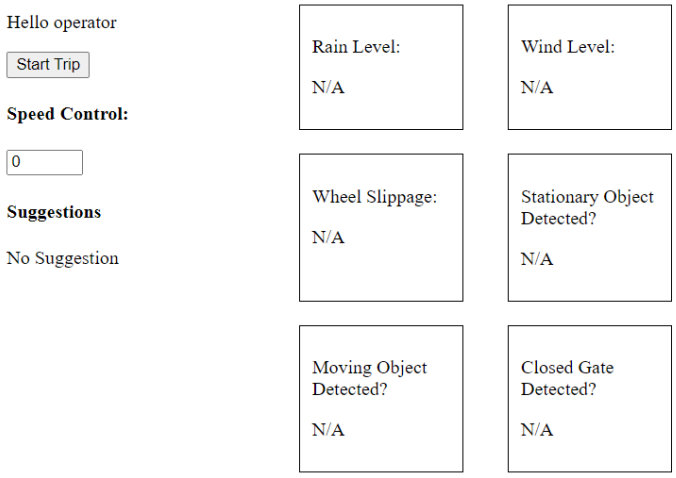
5.2 Pros and Cons

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

5.3 Choice of Architecture



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



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

Section 6: Code

Examples below, full code on: https://replit.com/join/uivsiouq-kaiqichee

```
import tsnr as tsnr

class IoTEngine:
  def __init__(self):
    self.recs = ['ok','ok','ok','ok','ok','ok']
    self.vals = {''}
    self.my_tsnr = tsnr.TSNR()

  def rain_gauge_rec(self, reading):
    return 'ok'

  def anemometer_rec(self, reading):
    return 'bad'

  def wheel_slip_rec(self, reading):
    return 'very bad'

  def moving_obj_rec(self, reading):
    return 'bad'

  def stat_obj_rec(self, reading):
    return 'ok'

  def gate_rec(self, reading):
    return 'very bad'
```

```
import csv

class Sensor:
  def __init__(self, sensorType, ID):
    self.sensorType = sensorType
    self.ID = ID
    self.value = self.get_value()

  def get_sensorType(self):
    return self.sensorType

  def get_ID(self):
    return self.ID

  def get_value(self):
    queue = []
    with open('data.csv') as File:
      readFile = csv.reader(File)
      for row in readFile:
        if(self.get_sensorType() == row[0]):
          queue=row[1:]
    return queue

  def get_reading(self):
    return self.value.pop()
```

```
import IoTEngine as engine
import csv

class Display:
    def __init__(self, user):
        my_engine = engine.IoTEngine()
        self.suggestions = my_engine.get_suggestion();
        #self.suggestions = ["ok","bad","very bad","ok","very bad","bad"]
        self.user = user

    #makes an array of colors that each warning should be
    def warning_states(self):
        warning = []
        for i in range(6):
            if(self.suggestions[i]=="ok"):
                warning.append("\033[92m")
            elif (self.suggestions[i]=="bad"):
                warning.append("\033[33m")
            elif (self.suggestions[i]=="very bad"):
                warning.append("\033[31m")
        return warning
```

Works Cited

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

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

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

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

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

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