HUG THE RAIL

IOT

Trent Zeller, Jacob Wood, Steven Hoang, Roland Tumbokon

Table of Contents

Section 1.0 **3**
- 1.1 Introduction 3
- 1.2 Team 4
- 1.3 Solution Abstract 4

Section 2.0 **4**
- 2.1 Statement of Problem 4
- 2.2 Required Data 4
- 2.3 Expected Output of IoT 5
- 2.4 Sensors 5
- 2.5 Conceptual Architecture 5
- 2.6 Timeline 6

Section 3.0 **7**
- 3.1 Non-Functional Requirements 7
    - 3.1.1 Security 7
    - 3.1.2 Performance 7
    - 3.1.3 Reliability 7
- 3.2 Functional Requirements 8
    - 3.2.1 Standing Objects 8
    - 3.2.2 Moving Objects 9
    - 3.2.3 Gates 9
    - 3.2.4 Global Positioning System 9
- 3.3 Hardware and Operating System 10

Section 1:

1.1 Introduction

The internet of things or IoT for short describes a collection of sensors and other technologies that communicate with each other. Currently, trains use cellular networks to transmit their data, such as speed and location. The problem we currently face is the consequences of losing connections with the trains. Thus, we are developing systems in order to keep the trains running smoothly and safely when connection is lost. We are doing this by utilizing all the sensors on the train and directing the information and guidance to the train operators via an interface.

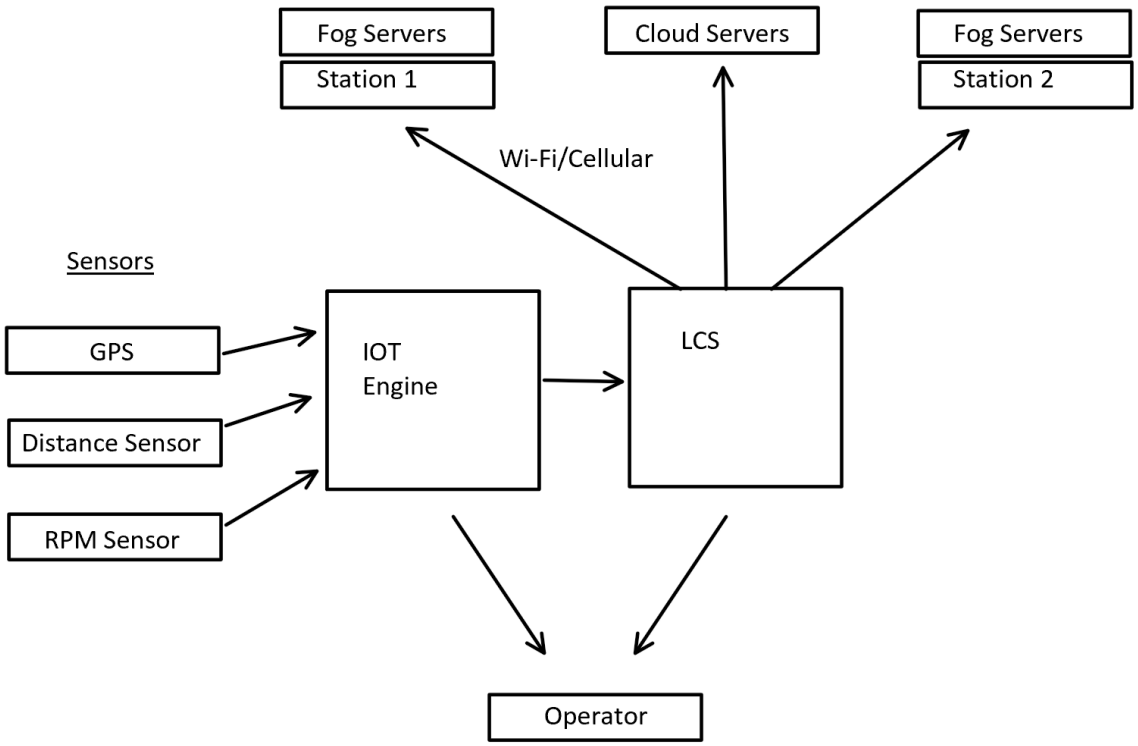


Diagram 1.1

1.2 Team

Our team consists of four members each with their own roles, but we will be working together closely to complete the project. Our team leader will be Steven. He has a lot of experience with this and is a certified scrum master. He will be in charge of dividing up the work for the team and changing the requirements or tasks depending on how the project is going. Trent will be the coordinator who is in charge of making sure everyone is getting their assigned work done on time and that nobody is falling behind. He will work with Steven if work assignments need to be adjusted or changed. Roland will be in charge of design. He will be working with software such as Adobe Illustrator and Photoshop to create the visuals and designs for the system. He will make the models to show the investors the progress that is being made and show them different possibilities for the future. Jacob will be the lead coder. He will decide the main structure of the system and how it will be split with pseudocode. While he will be in charge of making the overall outline of the code, all the team members will help to make it.

1.3 Solution Abstract

We will be using many different types of software for our solution. We have decided to use Java to code the project. This is the language we have the most experience with and feel like we can make the best solution in. We will also be using Trello and Gitlab to keep track of our work and to make sure that we are staying on time.

We will be using a unified process model to complete this project. We decided this would be the best way to make sure that any change in requirements could be completed and it would allow us to review and correct any problems we have. It will also allow us the most interaction with the stakeholders.

Section 2:

2.1 Statement of the problem:

The train operation depends on live data received from the Central Operation Center Servers via WiFi/Cellular network. We need to be able to operate the train when we lose WiFi/Cellular connection. We need to develop a system to allow us to continue the trip safely based on local data that we can collect in real time. This system will also have to be secure and streamline all the information to the train operators.

2.2 Required Data:

The IoT system will need to collect data in order to properly advise train operators on safe actions to take. The system will need data on the train's obstacles and physical surroundings. In particular, it will need to know if there are objects in front or back of the train, especially other trains or moving objects, as well as the status of crossing gates. The IoT system will also keep track of the train's wheel speed in order to detect when the wheels are slipping. The train will also need to collect Global Positioning System data for this purpose, to

determine its speed. The system will also need to be aware of inclement weather, so it must detect snow, ice, rain, temperature, and wind speed.

2.3 Expected output of IoT:

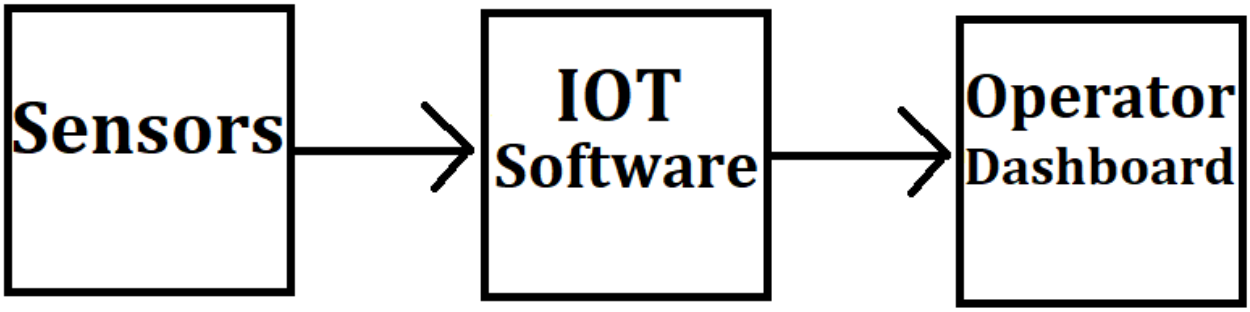
- IoT interfaces with sensors to collect data, analyze the data, use a rule-based logic to issue recommendation or action to the operator via the IoT screen.
- If the system detects a hazard on the rails, the IoT screen will make a recommendation (based on speed and distance) that the operator change speed or stop to avoid a collision.
- If the system detects the wheels slipping from sensors that detect wheel rpm, the system will recommend the operator slow down and pour sand on the rail. Once the rpm of the wheels have slowed and the slipping stops, the train will inform the operator.
- If the system detects inclement weather, the IoT screen will recommend that the operator will slow down until the inclement weather stops.

2.4 Sensors:

- Obstacle detection sensor:
    - Railway Optical Detection of Intrusions and Obstructions. This sensor searches for obstacles that are on the tracks.
- Rail Friction Sensor
    - Detects revolutions per minute of the wheels. This is compared with the speed of the train using Global Positioning System data and the revolutions per minute of the wheels. If there is a discrepancy between the information given, it is determined that the train is slipping.
- Time of Flight Sensor:
    - Detects the speed of surrounding objects
- Camera:
    - This is used to examine objects on the side of the train, such as gates.3

2.5 Conceptual Architecture

The Iot Software receives information and data from the sensors. Once it has this information, it will test to see if there are any problems or information that the operator requires. Any critical or requested information will be sent to the operator’s screen. This will allow the operator to 3receive any information that will be relevant to his needs without a wifi connection.



2.6 Timeline

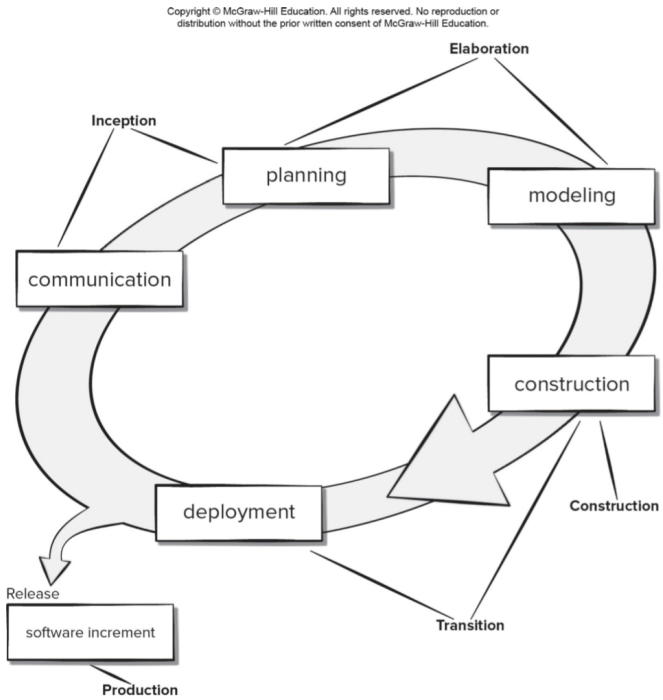


Diagram 2.6

Project Start: February 1st

Communication: February 1st - February 10th

Planning: February 10th - March 1st

Modeling: March 1st - March 10th

Construction: March 10th - March 31st

Deployment: April 1st

Section 3:

3.1 Non-Functional Requirements

3.1.1 Security: (list:no more than one per req)

R-1:To make sure our system is secure we will require a login screen on startup that will require the operator to input their username and password.

R-2: We will also have an administrator who will have their own separate username and password and they will be the only ones with access to the sensors and equipment.

R-3: Our network shall also be secured by LoRaWan protocol since it is well integrated for IOT and because it will work in the remote areas that some trains go through.

3.1.2 Performance:

R-4: Our system standard for response time for any event shall be no slower than half a second.

R-5: Our system will also need to be able to process at least 1000 events per second without slowing down the system or stopping it from doing something else.

3.1.3 Reliability:

R-6: Our system will need to operate with 99.99% accuracy.

3.2 Functional Requirements:

3.2.1: Standing objects

R-7: Our system will suggest to apply the brakes within .1 second of detecting an object on the track.

R-8: Our system will be able to detect when something is on the track 99.99% of the time.

R-9: The conductor will be able to turn on and off the brakes by the push of the button.

R-10: The system will give the conductor a suggestion on keeping the same speed when it determines it necessary to do so if something is on the track.

R-11: The system will give the conductor a suggestion on decreasing the speed when it is necessary to ensure safety because something is on the track.

R-12: The system will alert the driver on the screen when something is on the track.

R-13: The system will correctly determine the best way to avoid a collision with something standing on the track.

3.2.2: Moving objects

R-14: The system will determine if something crossing the track will make it across without a need for the train to brake.

R-15: The system will determine if the train needs to slow down because of something the train is approaching from the front.

R-16: The system will suggest when to apply the brakes if the train sees something run across the track in front of the train.

R-17: The system will be able to determine the speed of objects on the track and determine if they will interfere with the train.

R-18: The system will be able to determine the distance of objects moving on the track and determine if they will interfere with the train.

R-19: The system will correctly determine when to begin slowing down if something is stopped in front of the train.

R-20: The system will give the operator a new suggested speed if it determines a moving object will interfere with the train.

3.2.3: Gates

R-21: The system will accurately determine if the gates are open or closed and signal the gates to close if they are open.

R-22: The system will determine if a vehicle is crossing the tracks when the gates are closing and slow the train down to avoid collision.

R-23: The system will detect if a vehicle goes through the gates and the train needs to brake to avoid collision.

R-24: Upon detecting a gate crossing, the system will inform the conductor to blow the horn for 15 seconds while at 1 mile away from gate crossing and again when it reaches the gate for 5 seconds.

3.2.4: Global Positioning System and Wheel Revolutions per Minute

R-25: The system will detect if the wheels are slipping on the tracks and tells the conductor to deploy sand to avoid slipping.

R-26: The system will determine if the tracks are wet and suggest the train to slow down to ensure safety.

R-27: Determine using the wheel revolutions per minute if the wheels are slipping.

R-28: Determine using the Global Positioning System whether the wheels are slipping.

R-29: Determine if the train is able to speed up because the wheels are no longer slipping.

R-30: Determine if the train needs to brake because it cannot get traction on the track.

R-31: Our system should be able to read data from every one of its sensors simultaneously.

3.3 Hardware & Operating System:

R-32: Our IoT hardware will be able to support 1,000 sensors.

R-33: The IoT system will support 5 terabytes of data everyday.

R-34: The system will be run on Zephyr, a highly configurable linux operating system with a strong security system.

Section 4:

4. Requirement Modeling

4.1 Use Cases

Case 1: Conductor turns on IoT Engine.

Primary Actor: Train conductor
Goal in Context: Start up the functions of the IoT engine
Precondition: The train is powered on
Trigger: The conductor powers up the IoT engine
Scenario:

1. IoT engine turns on
2. A valid username and password is passed
3. IoT functions and sensors are enabled
4. The train is ready for use

Exceptions:

1. An invalid username or password is passed. Login is unsuccessful.
2. Sensors are malfunctioning and a warning is shown to the conductor.
3. IoT Engine startup fails and the console logs are shown and recorded.

Channel to Actor: Via IoT Engine Display

Case 2: Administrator turns on IoT Engine.

Primary Actor: Administrator
Goal in Context: Start up the functions of the IoT engine
Precondition: The train is powered on
Trigger: The Admin powers up the IoT engine
Scenario:

5. IoT engine turns on
6. A valid username and password is passed
7. IoT functions and sensors are enabled
8. The train is ready for use

Exceptions:

4. An invalid username or password is passed. Login is unsuccessful.

5. Sensors are malfunctioning and a warning is shown to the conductor.
6. IoT Engine startup fails and the console logs are shown and recorded.

Channel to Actor: Via IoT Engine Display

Case 3: Train detects a standing object on the tracks.

Primary Actor: Obstacle Detection Sensor
Goal in Context: To avoid collision with a stationary object
Precondition: The IoT engine and train are active.
Trigger: A stationary obstacle is on the track, in range of the obstacle detection sensor
Scenario:

1. The sensor detects an obstacle on the track, which is not moving.
2. Sensor data is passed to the Time Sensitive Networking Router, which passes the data to the display.
3. The IoT engine uses an audio cue to get the conductor's attention.
4. The IoT engine displays information from the obstacle detection sensor about the distance of the object, the projected time until collision with the object, and recommends that the conductor stops the train in order to avoid collision with the stationary object.
5. The conductor decides to stop to avoid collision with the object.

Exceptions:

1. The object is too small to pose an issue: The conductor ignores the IoT and continues moving at the same speed.
2. The object begins moving towards the train: See case 4.
3. The object begins moving away from the train, on the track: See case 5.
4. The object moves off of the track: The IoT greys out the notification of an object on the tracks, and removes it from the display.

Channel to Actor: Via internet of things connection to IoT Engine.
Secondary Actor: Conductor
Channels to Secondary Actor: IoT Engine display

Case 4: Train detects an object moving towards the train, on the tracks.

Primary Actor: Time of Flight Sensor
Goal in Context: Warn conductor of a possible upcoming collision with an object moving towards the train on the tracks.
Precondition: IoT Engine and train are active.
Trigger: An obstacle is on the track, moving towards the train.
Scenario:

1. The Time of Flight Sensor identifies an object moving towards the train, and calculates its speed.
2. Sensor data is passed to the Time Sensitive Networking Router, which passes the data to the display.
3. Sensor issues an audible/visual cue for an upcoming object in front or the sides of the train.

4. The IoT engine displays information from the time of flight sensor about the distance of the object, the speed of the object, the projected time until collision with the object, and recommends that the conductor stops the train in order to avoid collision with the stationary object.
5. The conductor will assess whether the train should speed up or slow down

Exceptions:

1. If the object is small, the warning will be ignored.
2. The object stops moving: See case 3.
3. The object begins moving away from the train, on the track: See case 5.
4. The object moves off of the track: The IoT greys out the notification of an object on the tracks, and removes it from the display.

Channel to Actor: Via internet of things connection to IoT Engine.
Secondary Actor: Conductor, Time of Flight Sensor
Channels to Secondary Actor:

- Conductor: Via IoT Engine display
- Time of Flight Sensor: Via internet of things connection to IoT Engine.

Case 5: Train detects an object moving away from the train, on the tracks.

Primary Actor: Time of Flight Sensor
Goal in Context: Warn the conductor in case the object starts moving toward the train or allow speeding up if the object is a safe distance away.
Precondition: IoT Engine and train are active
Trigger: Time of Flight Sensor detects an object on the tracks moving away from the train.
Scenario:

1. The Time of Flight Sensor identifies an object moving away from the train, and determines its speed.
2. Sensor data is passed to the Time Sensitive Networking Router, which passes the data to the display.
3. The train will suggest maintaining speed or speeding up if the object is moving away from the train at a safe distance.
4. The conductor will assess how to control the speed of the train.
5. If the object starts moving toward the train, refer to Use Case 4

Exceptions:

1. If the object is small, the warning will be ignored.
2. The object stops moving: See case 3.
3. The object moves off of the track: The IoT greys out the notification of an object on the tracks, and removes it from the display.

Channel to Actor: Via internet of things connection to IoT Engine.
Secondary Actor: Conductor
Channels to Secondary Actor: Via IoT Engine display

Case 6: Train detects a gate

Primary Actor: Camera
Goal in Context: Provide a cautionary warning to the conductor when moving across railway crossings.
Precondition: IoT Engine, train, gate are active.
Trigger: Camera detects a gate ahead of the train
Scenario:

1. Camera identifies a gate ahead of the train.
2. The camera will inspect the area ahead, and determine if the gate is up or down, and the distance from the gate.
3. Sensor data is passed to the Time Sensitive Networking Router, which passes the data to the display.
4. The display makes an audible/visual cue to the conductor about an upcoming gate, and prompts them to blow the horn for 15 seconds at 1 mile away from the gate crossing.
5. The conductor will assess whether to stop, slow down, or maintain speed, and blow the horn.
6. The display makes an audible/visual cue for the conductor to blow the horn for 5 seconds when it reaches the gate.

Exceptions:

1. Fog or rain may hinder the camera's ability to see an upcoming gate.

Case 7: Train detects when the wheels are slipping

Primary Actor: Rail Friction Sensor
Goal in Context: Provide a warning to the conductor about wheel slippage
Precondition: IoT Engine and train are active.
Trigger: Wheel rotations per second do not match the GPS's speed of the train
Scenario:

1. The Rail Friction Sensor calculates the speed of the wheels compared to the speed the GPS says
2. Sensor data is passed to the Time Sensitive Networking Router, which passes the data to the display.
3. The display will light up a color on the controls to warn the conductor about the wheel slipping. An audio cue will play if the wheel slippage is heavy.
4. A green light will appear under normal and very light wheel slippage.
5. A yellow light will appear and an audio will play under light to medium wheel slippage.
6. A red light will appear and an audio will play under heavy wheel slippage.

Exceptions:

1. An object on the tracks may change how the conductor changes the speed of the train
2.

Case 8: Conductor turns off the IoT Engine

Primary Actor: Conductor
Goal in Context: To deactivate the IoT Engine
Precondition: The IoT Engine is on.
Trigger: Conductor hits the off button on the IoT Engine display
Scenario:

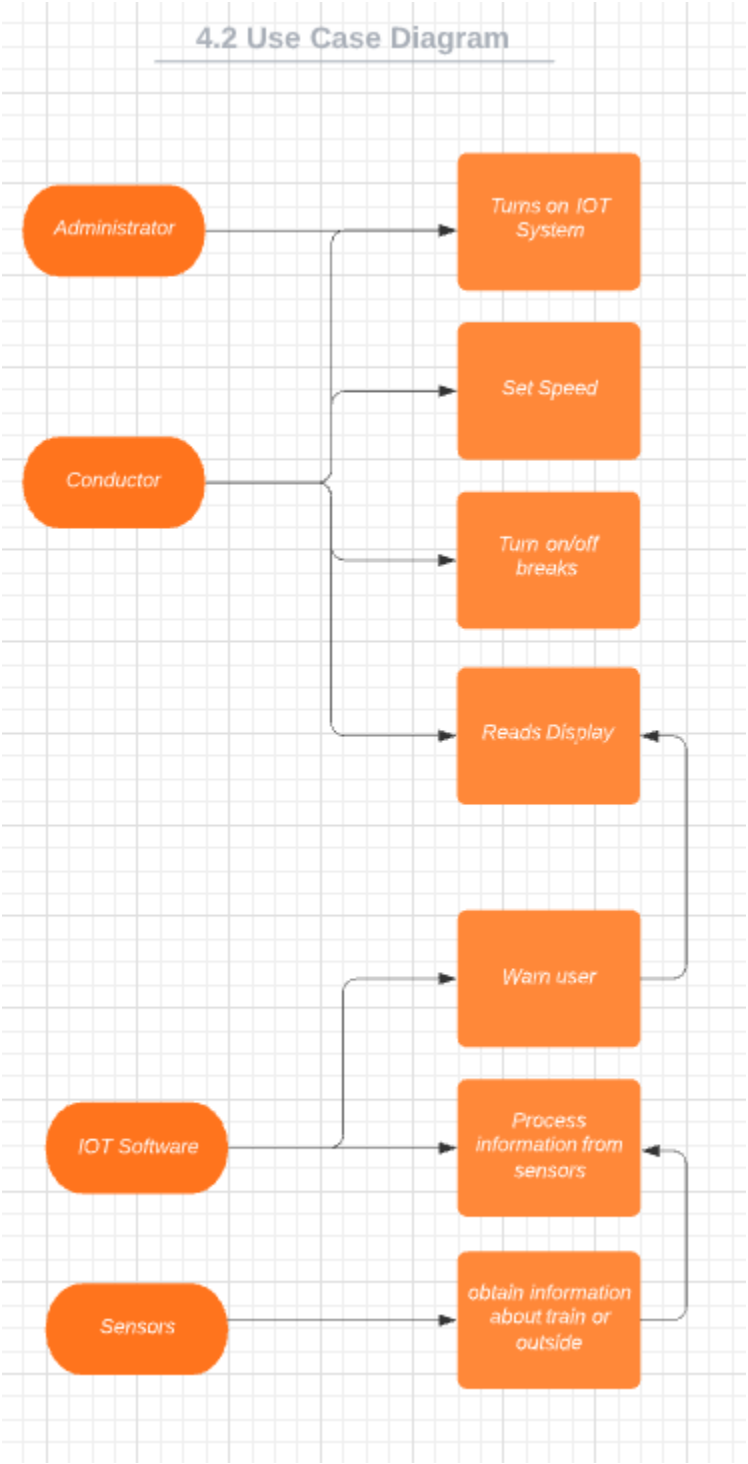
1. Conductor is prompted to confirm that they want to deactivate the IoT Engine
2. Conductor confirms they want to deactivate the IoT Engine by hitting the off button again.

Exceptions:

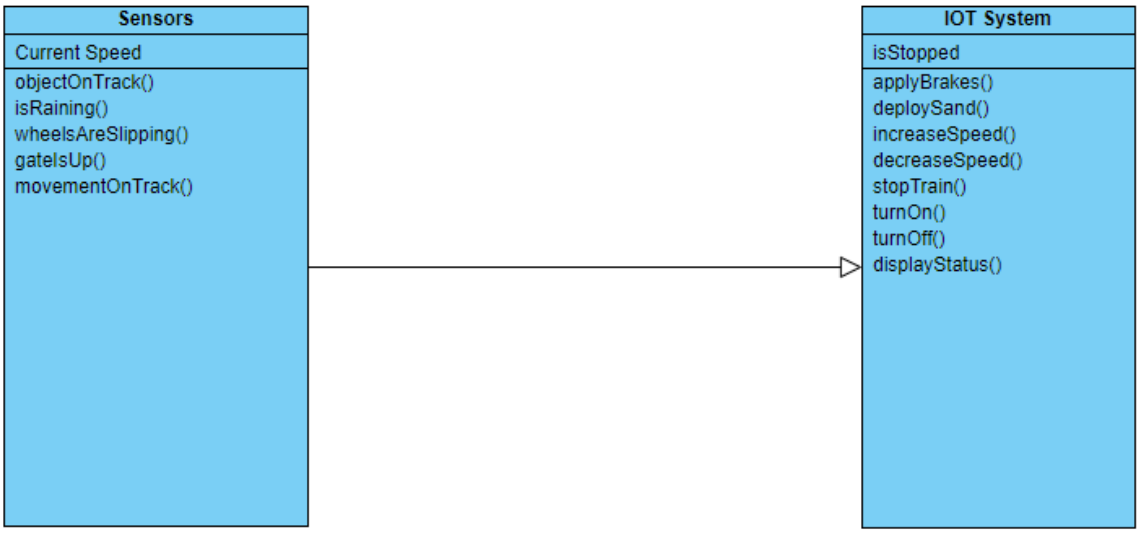
1. The train is in motion: The IoT Engine will not allow the conductor to shut it down, so that it is able to warn the conductor about dangerous conditions.

Frequency of Use: Once every day, when the train returns to the yard.
Channel to Actor: Via the IoT Engine display.

4.2 Use Case Diagram



4.3 Class-Based Modeling



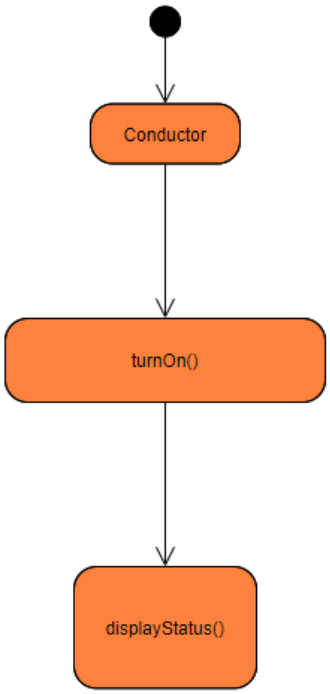
4.4 CRC Modeling/Cards

Class: Sensors
Description: Obtains data for various IOT systems processes and algorithms.
Responsibilities: - Return current speed when requested - Detect motion on the tracks - Detect rain and debris on tracks - Determine if gate is up
Collaborators: - IOT System - Conductor - System Admin

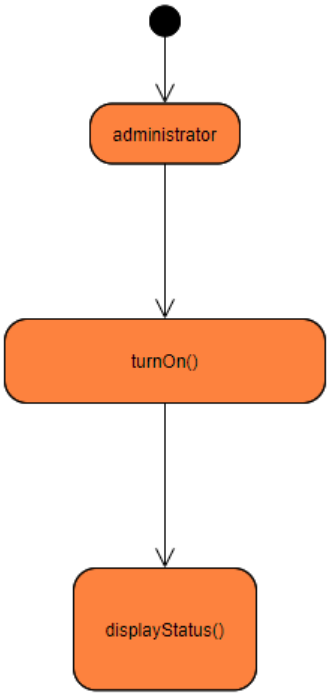
Class: IOT System
Description: Interprets input from sensors and determines how to adjust train speed
Responsibilities: - Apply brakes - Increase or decrease train speed - Release sand on the tracks - Prompt the conductor to blow the horn at gates - Stop train
Collaborators: - Sensors - Conductor - System Admin

4.5 Activity Diagrams

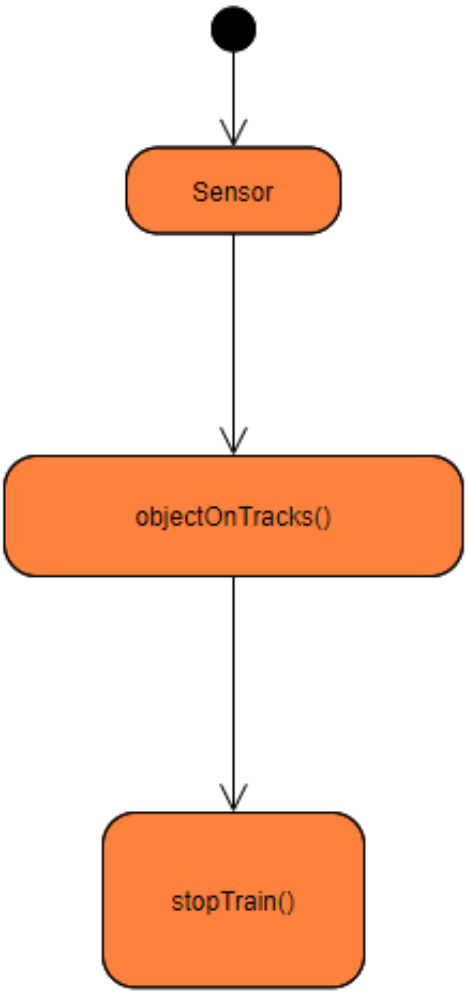
case: 1



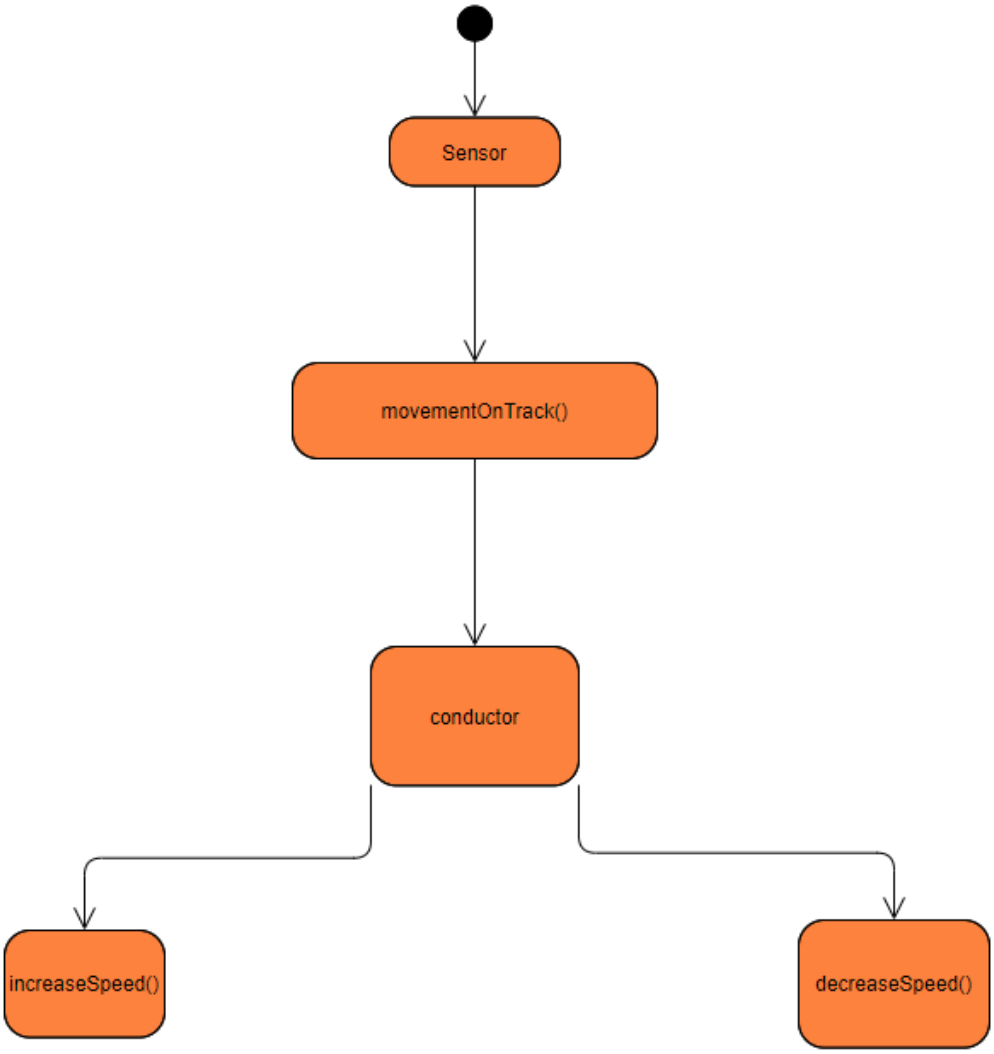
case: 2



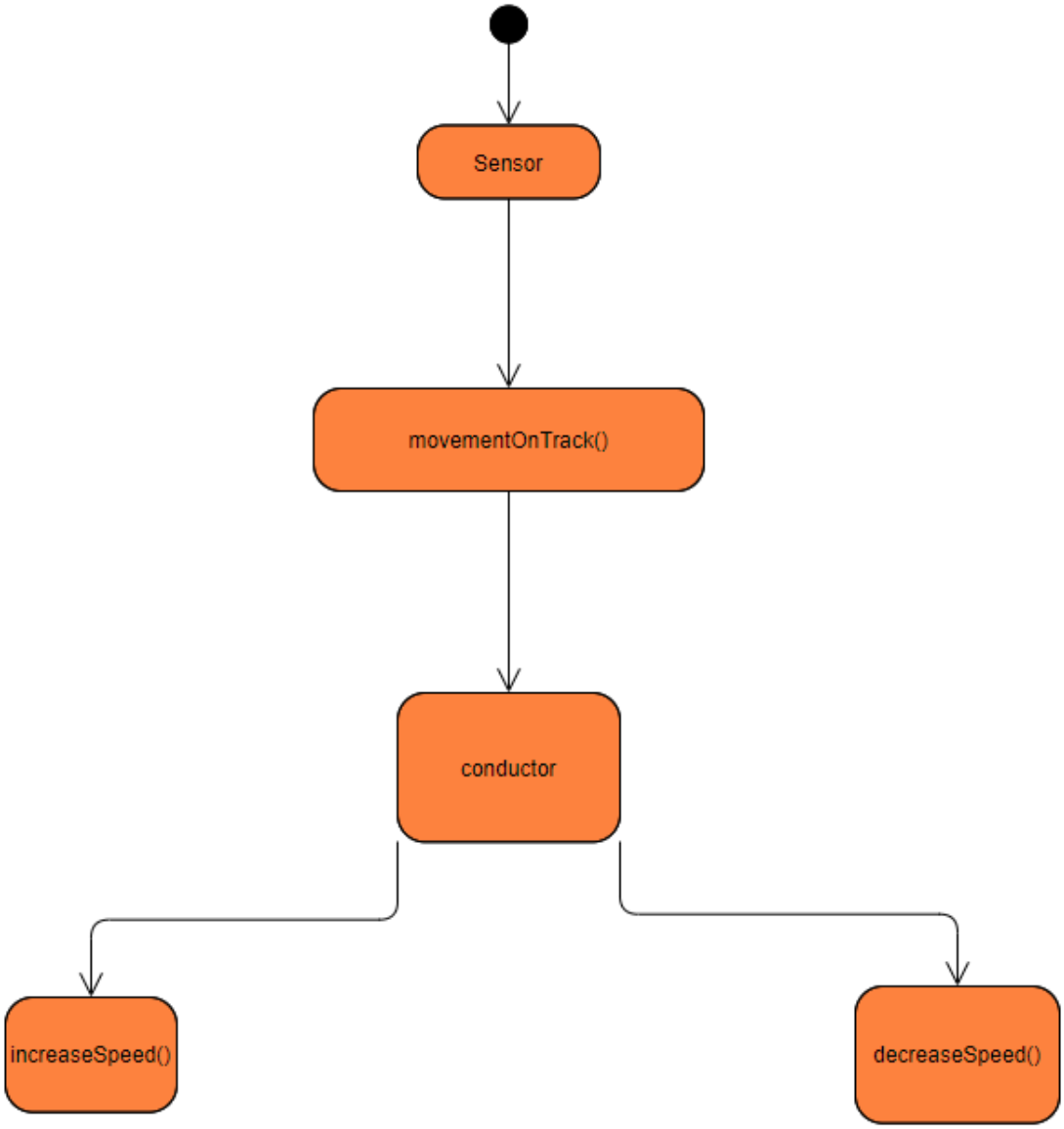
Case: 3

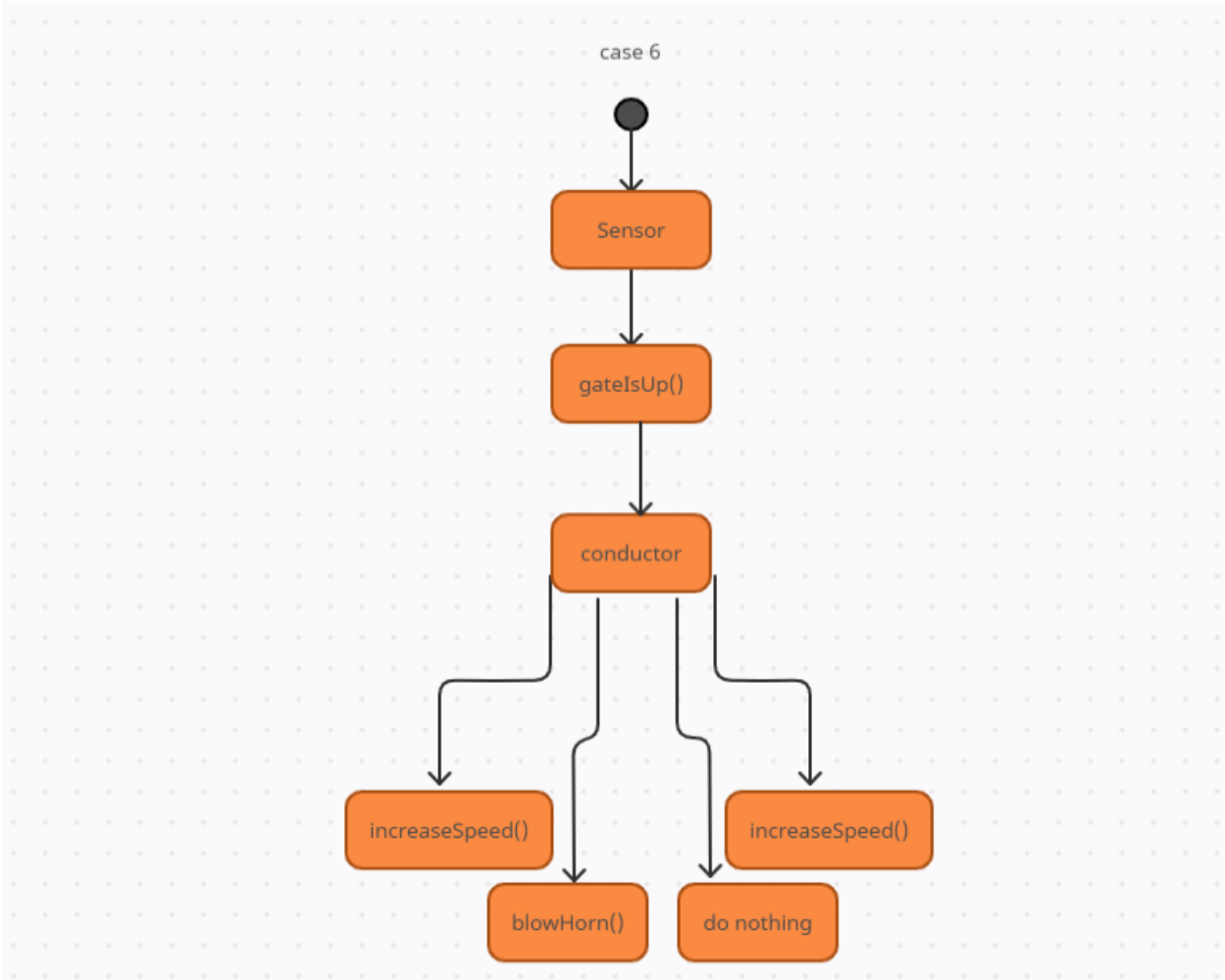


case: 4

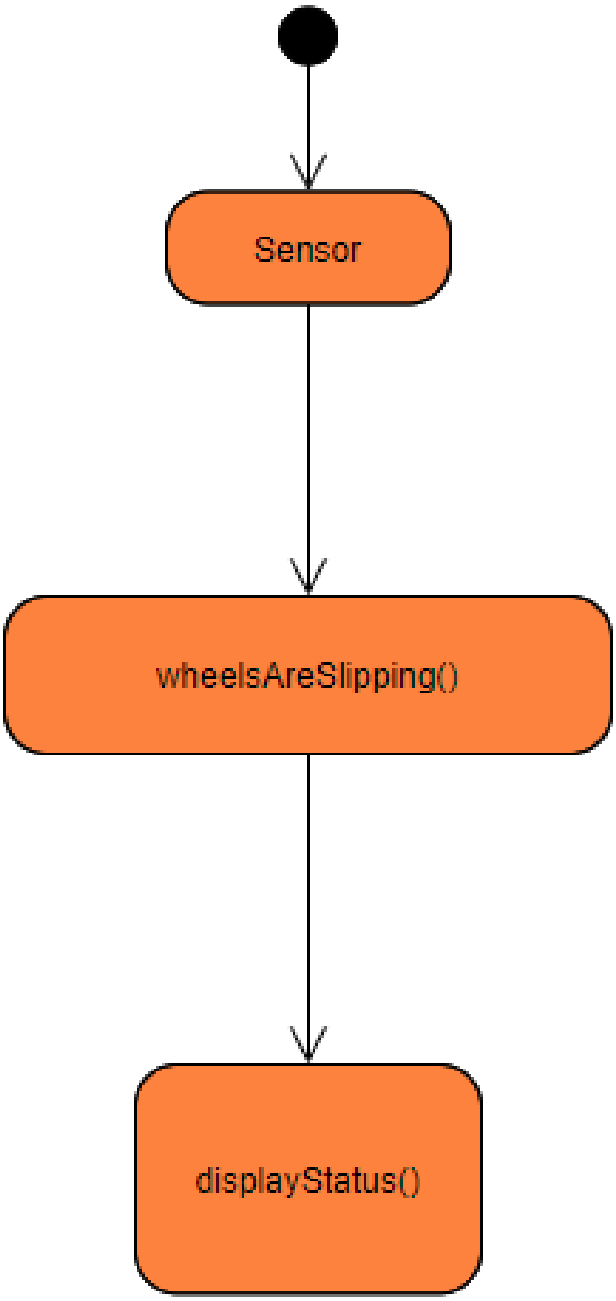

Sensor
movementOnTrack()
conductor
increaseSpeed()
decreaseSpeed()

case: 5

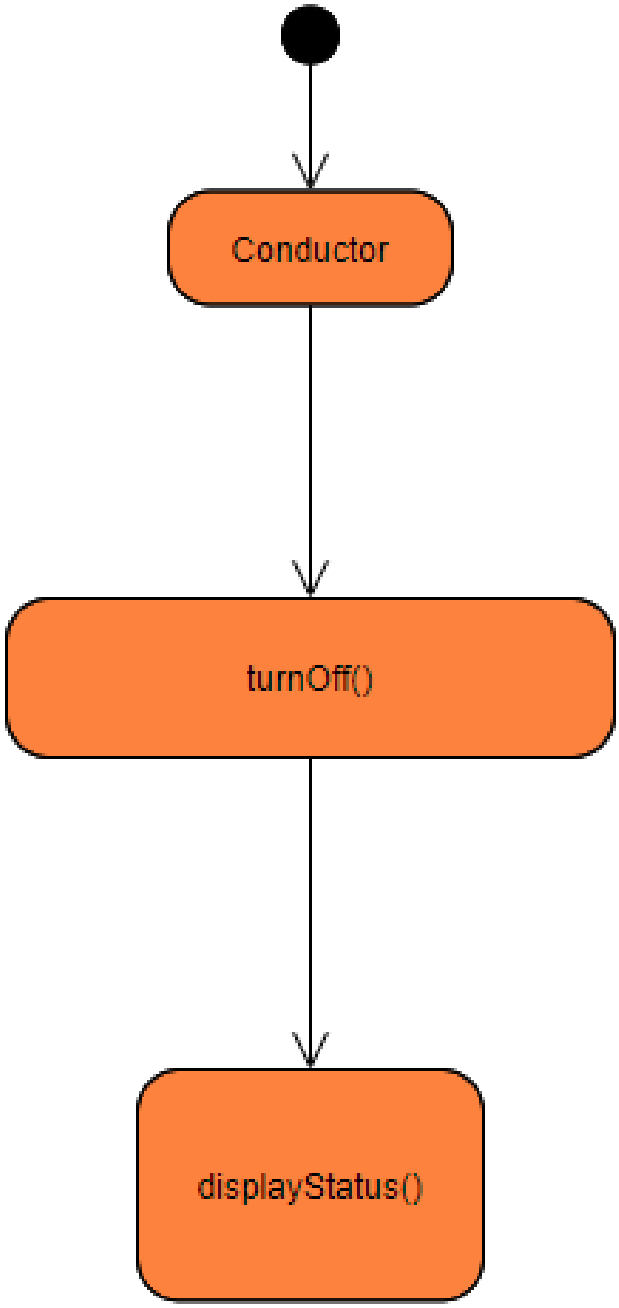

Sensor
movementOnTrack()
conductor
increaseSpeed()
decreaseSpeed()


case 6
Sensor
gateIsUp()
conductor
increaseSpeed()
blowHorn()
do nothing
increaseSpeed()

case: 7

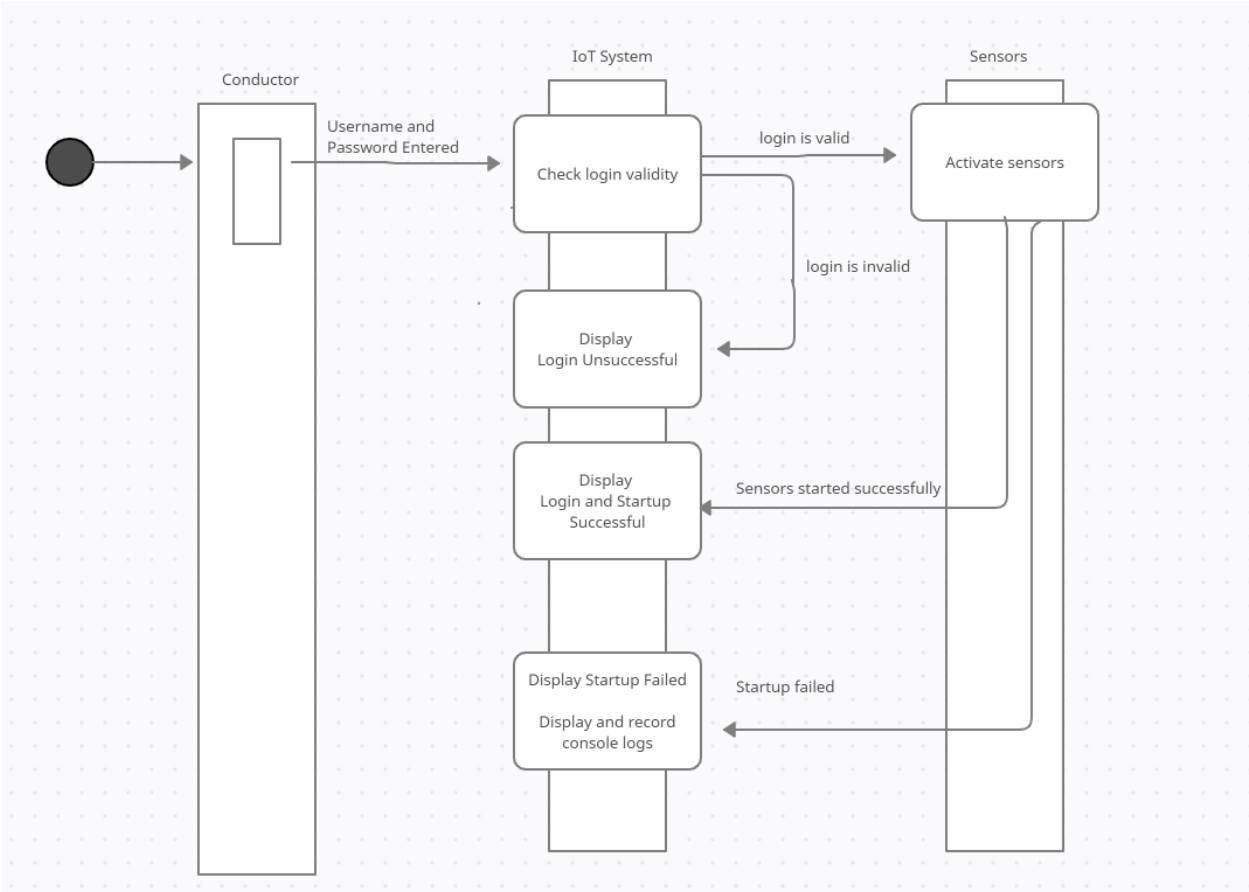


case: 8

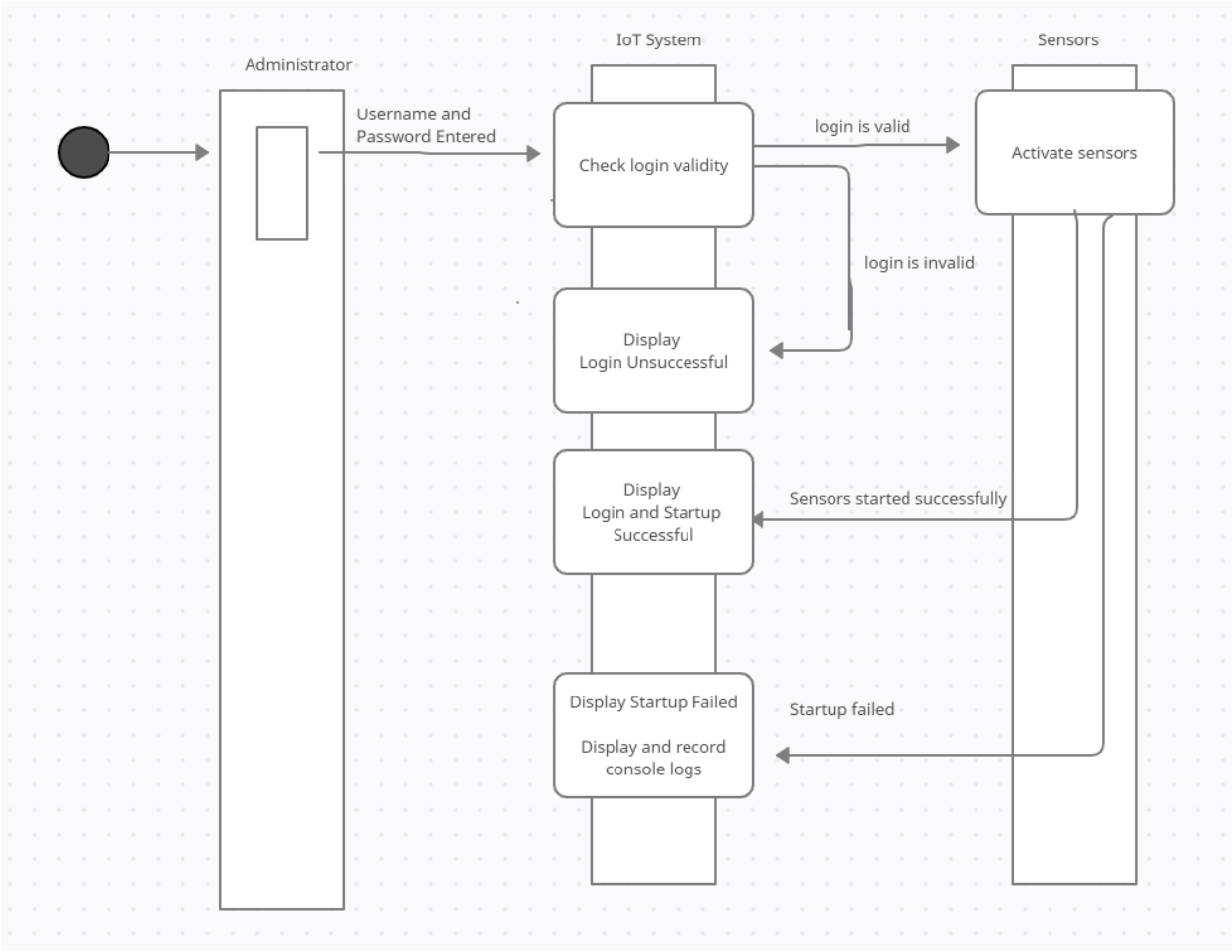


4.6 Sequence Diagrams

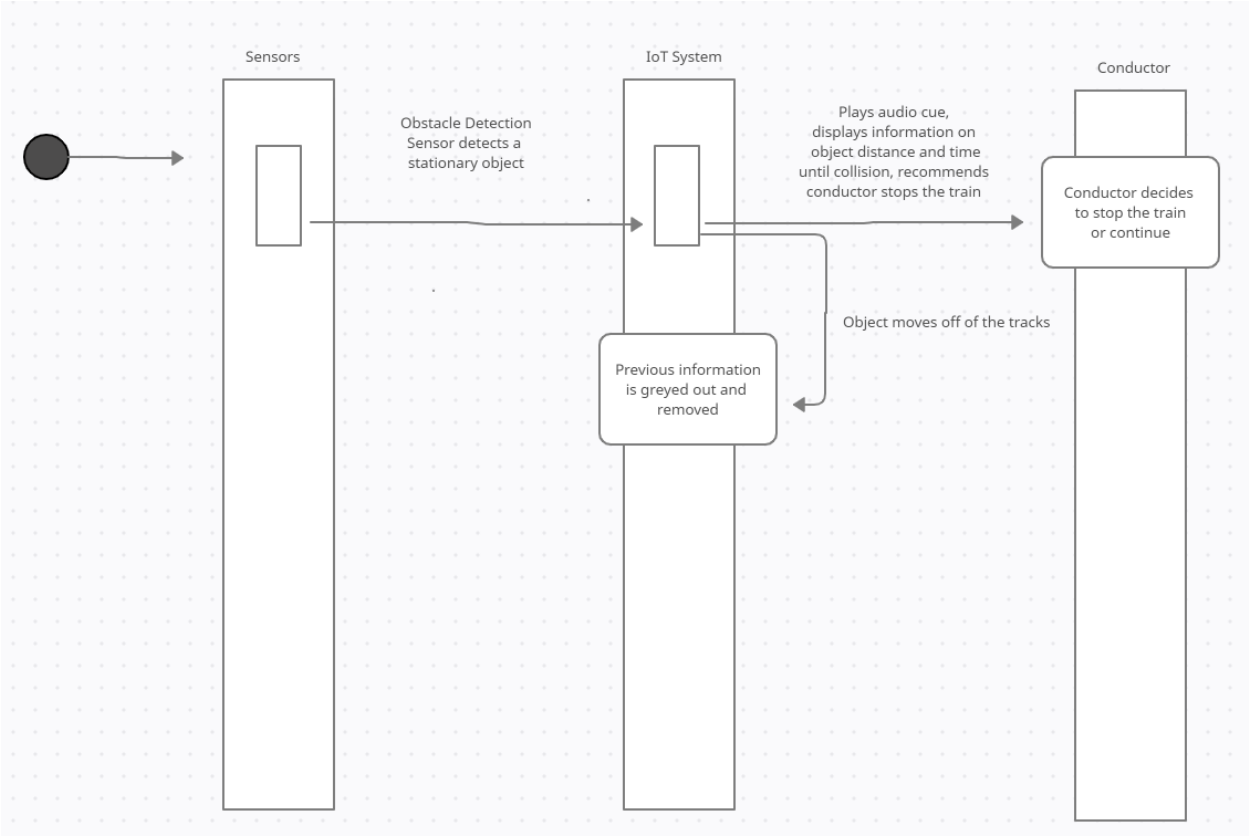
Case 1: Conductor turns on IoT Engine.



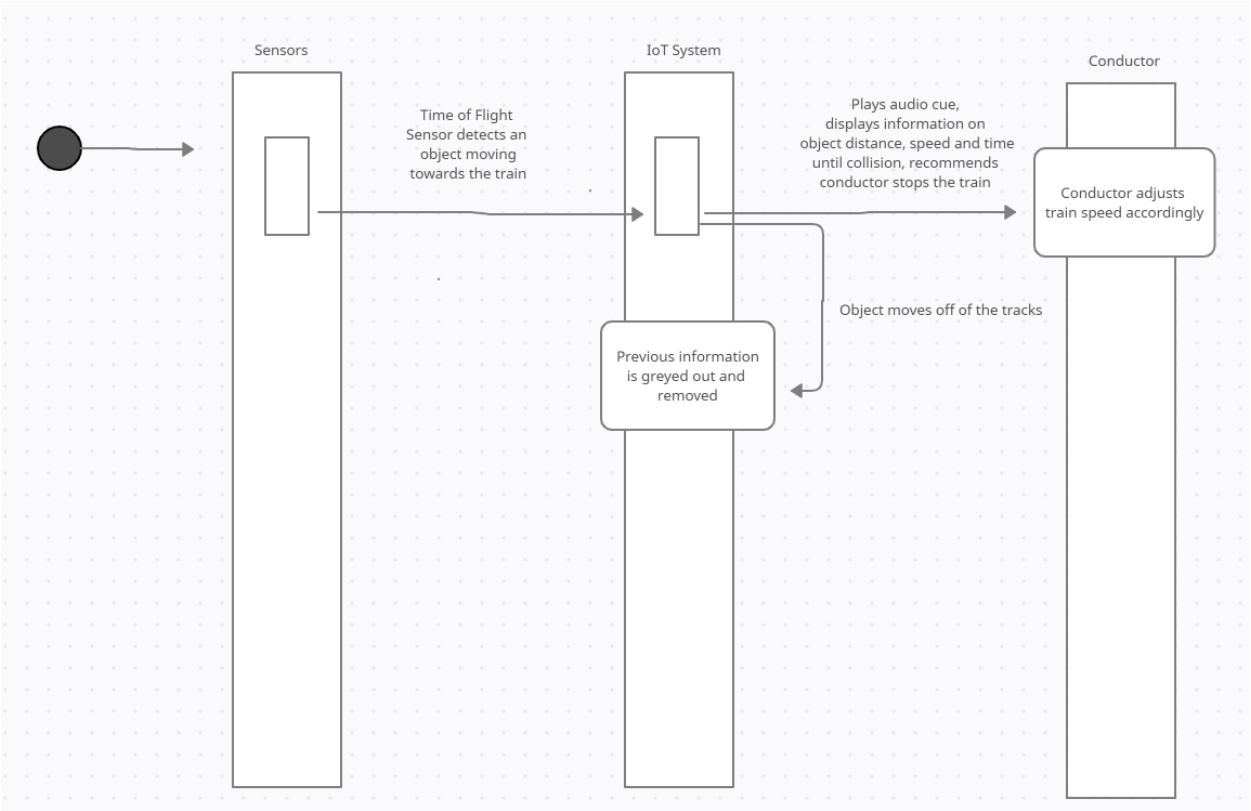
Case 2: Administrator turns on IoT Engine.



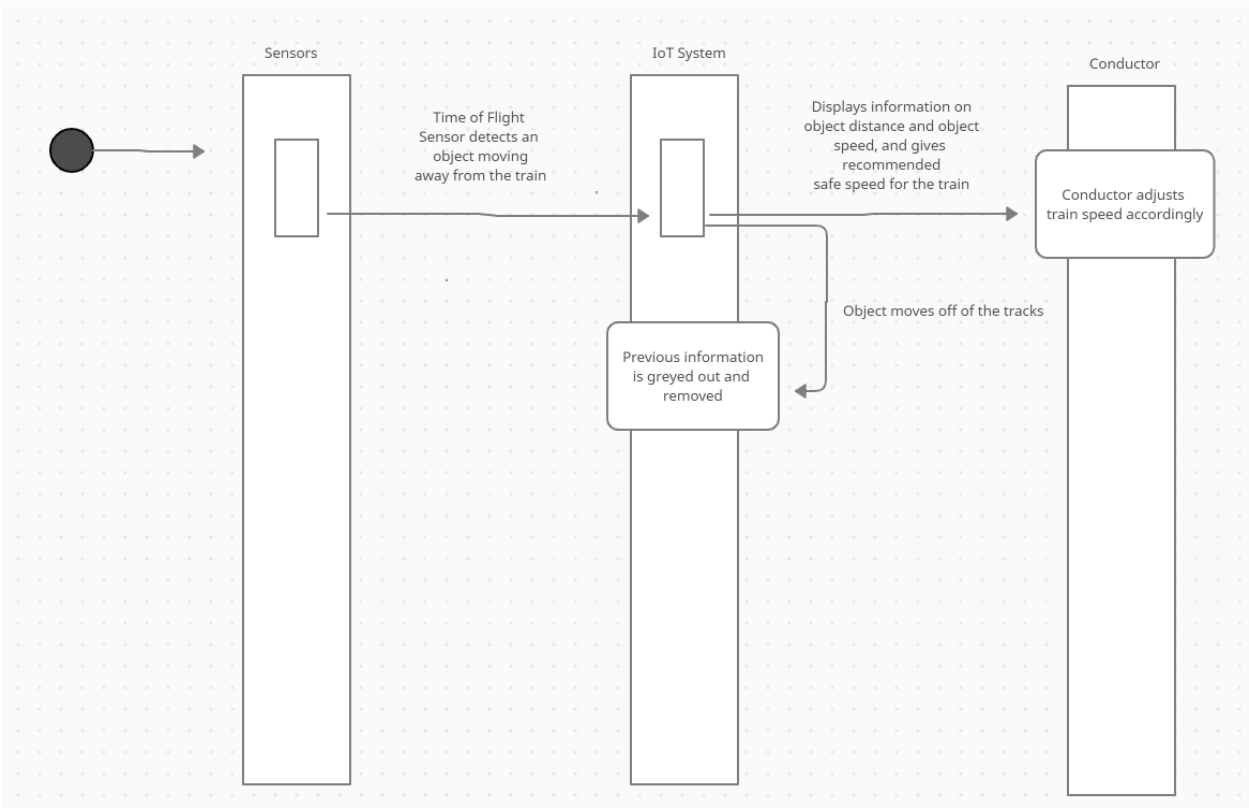
Case 3: Train detects a standing object on the tracks.



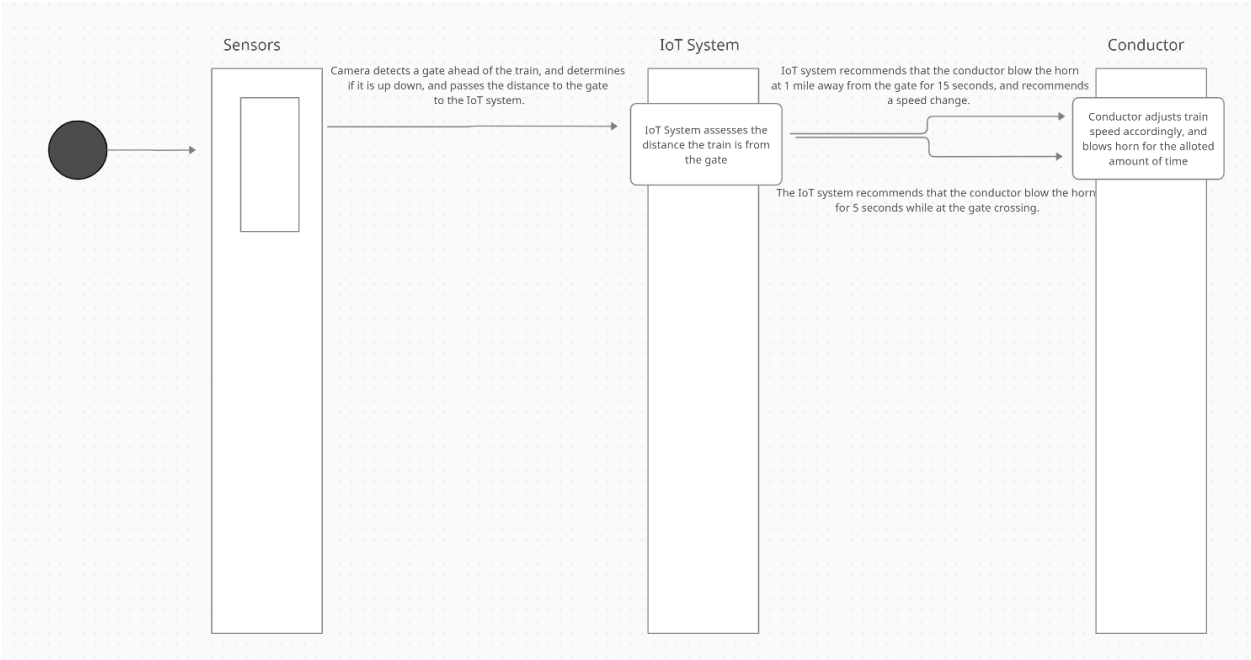
Case 4: Train detects an object moving towards the train, on the tracks.



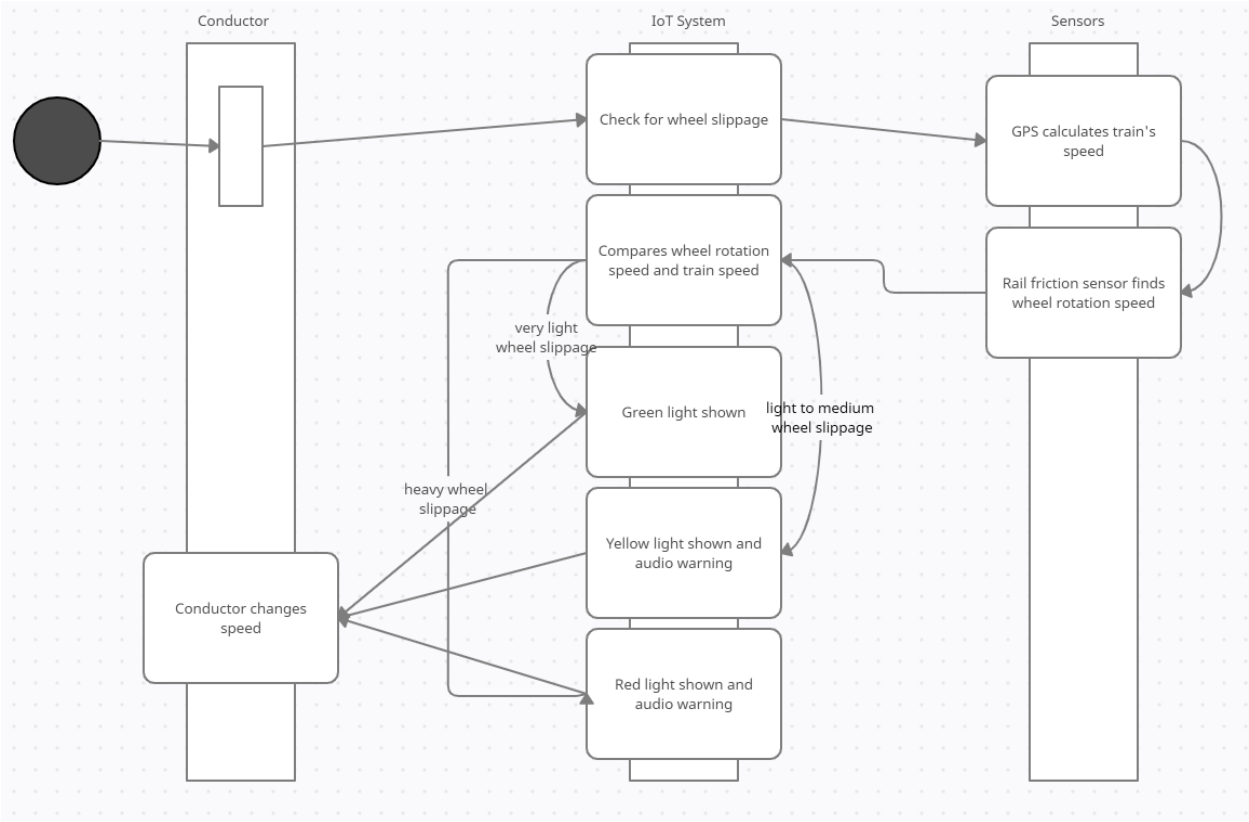
Case 5: Train detects an object moving away from the train, on the tracks.



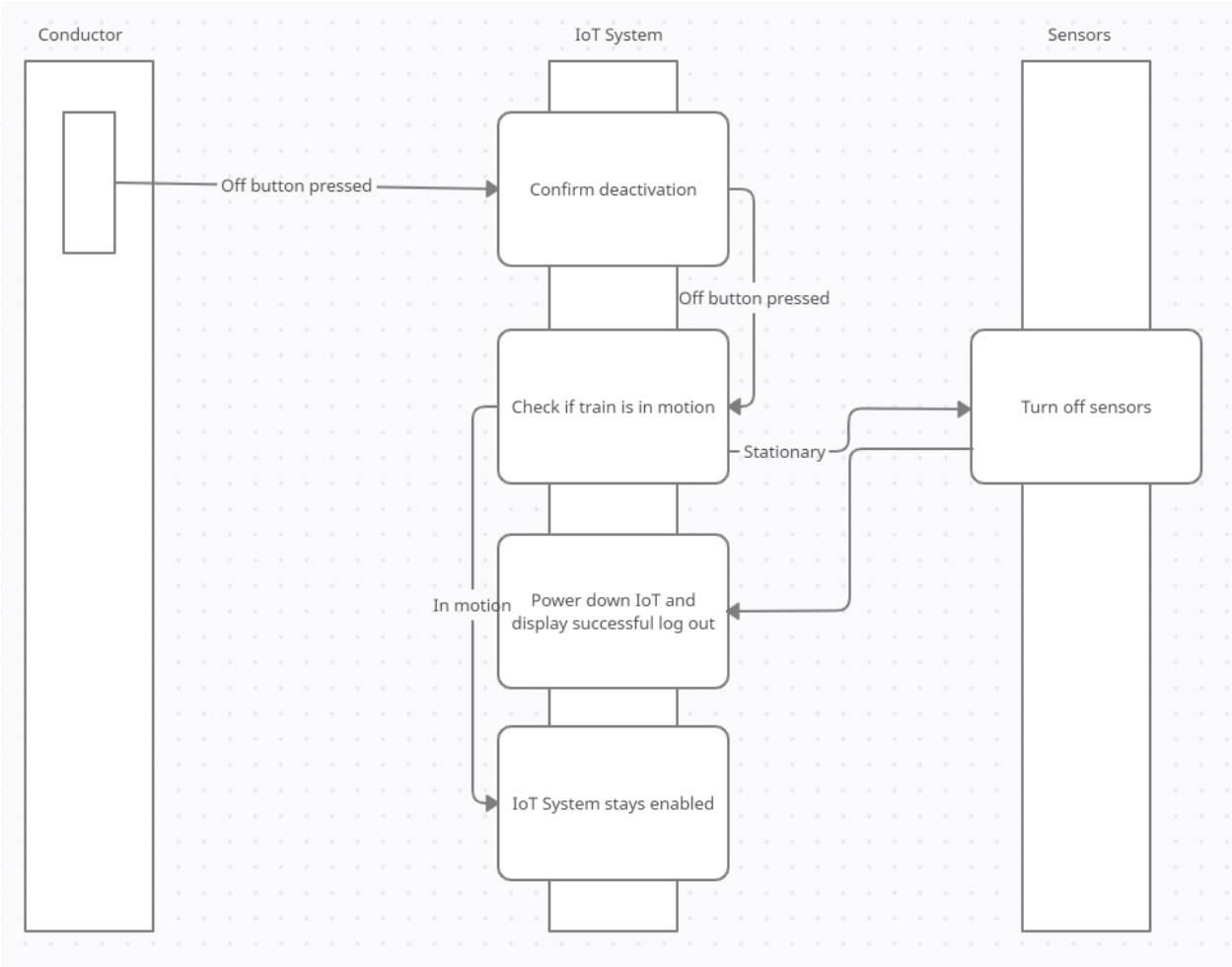
Case 6: Train detects a gate



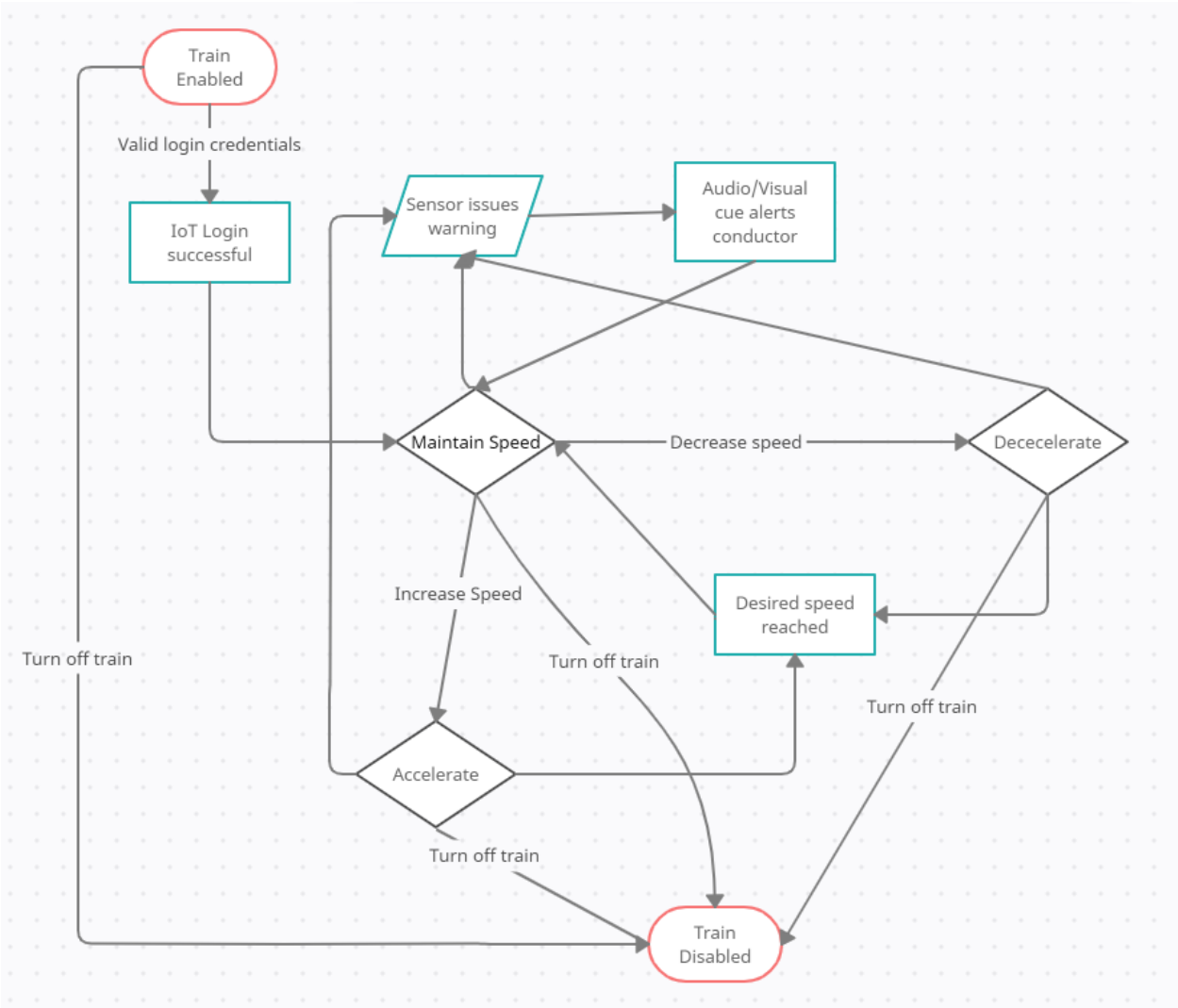
Case 7: Train detects when the wheels are slipping



Case 8: Conductor turns off the IoT Engine

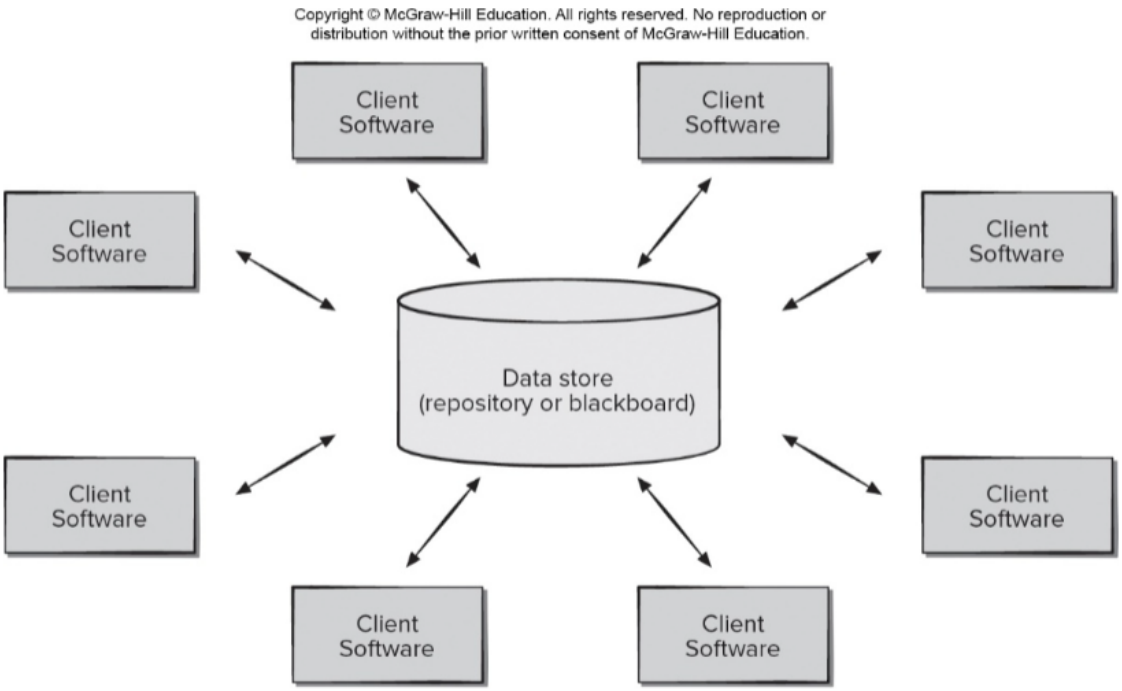


4.7 State Diagrams



Section 5:

5.1 Data Centered:



5.1.1 Architecture Description

A data store (e.g., a file or database) resides at the center of this architecture and is accessed frequently by other components that update, add, delete, or otherwise modify data within the store. The figure above illustrates a typical data-centered style. Client software accesses a central repository. In some cases the data repository is passive. That is, client software accesses the data independent of any changes to the data or the actions of other client software.

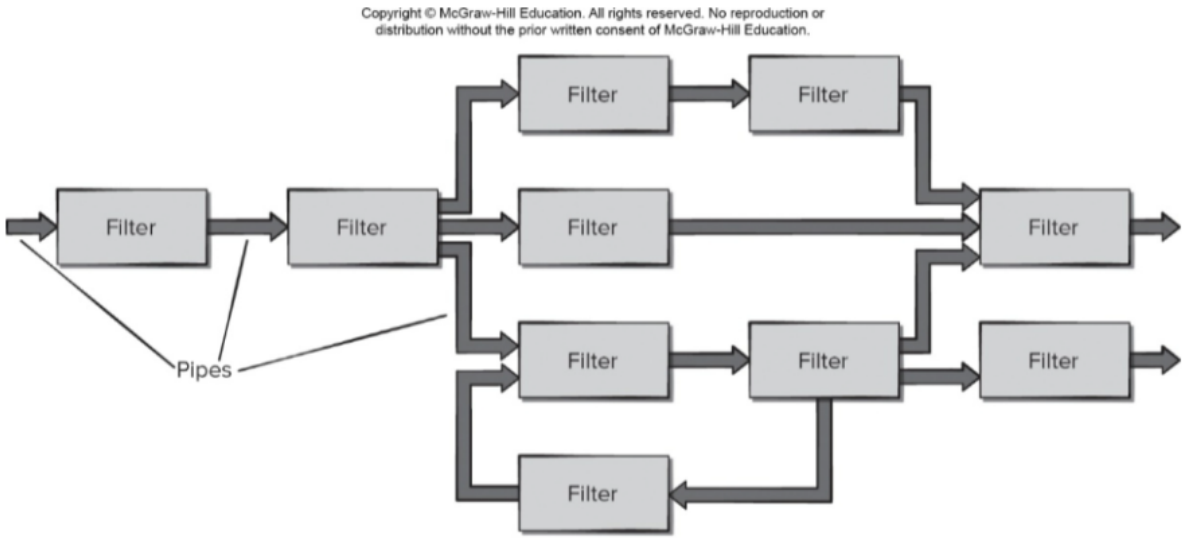
5.1.2 Pros

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

5.1.3 Cons

1. It is more vulnerable to failure and data replication or duplication is possible.
2. High dependency between data structure of data store and its agents.
3. Changes in data structure highly affect the clients.
4. Evolution of data is difficult and expensive.
5. Cost of moving data on network for distributed data.
6. Not a feasible architecture, because the IoT does not need to save the majority of its data to a database.

5.2 Data Flow:



5.2.1 Architecture Description

This architecture is applied when input data are to be transformed through a series of computational or manipulative components into output data. A pipe-and-filter pattern (Figure 13.2) has a set of components, called filters, connected by pipes that transmit data from one component to the next. Each filter works independently of those components upstream and downstream, is designed to expect data input of a certain form, and produces data output (to the next filter) of a specified form. However, the filter does not require knowledge of the workings of its neighboring filters.

5.2.1 Pros

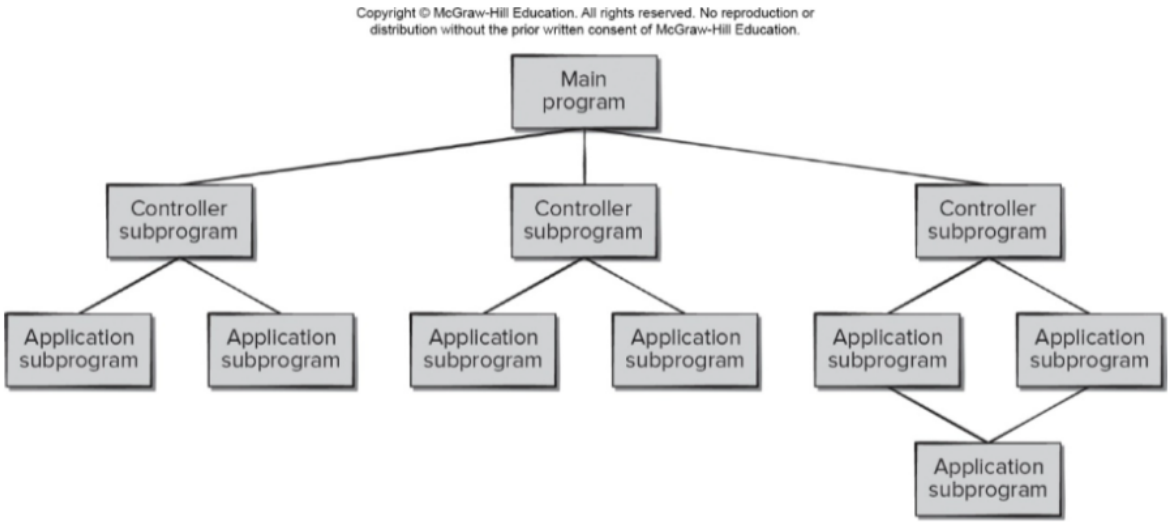
1. Doesn’t require information on neighbor filters
2. Low system maintenance

3. Supports parallel and sequential execution
4. Would allow all sensors to work on their own without depending on much from each other

5.2.2 Cons

1. Slow as each filter needs to file into the next
2. Doesn't allow filters to work together to solve a problem
3. No dynamic implementation

5.3 Call Return:



5.3.1 Architecture Description

This classic program structure decomposes function into a control hierarchy where a "main" program invokes a number of program components, which in turn may invoke still other components.

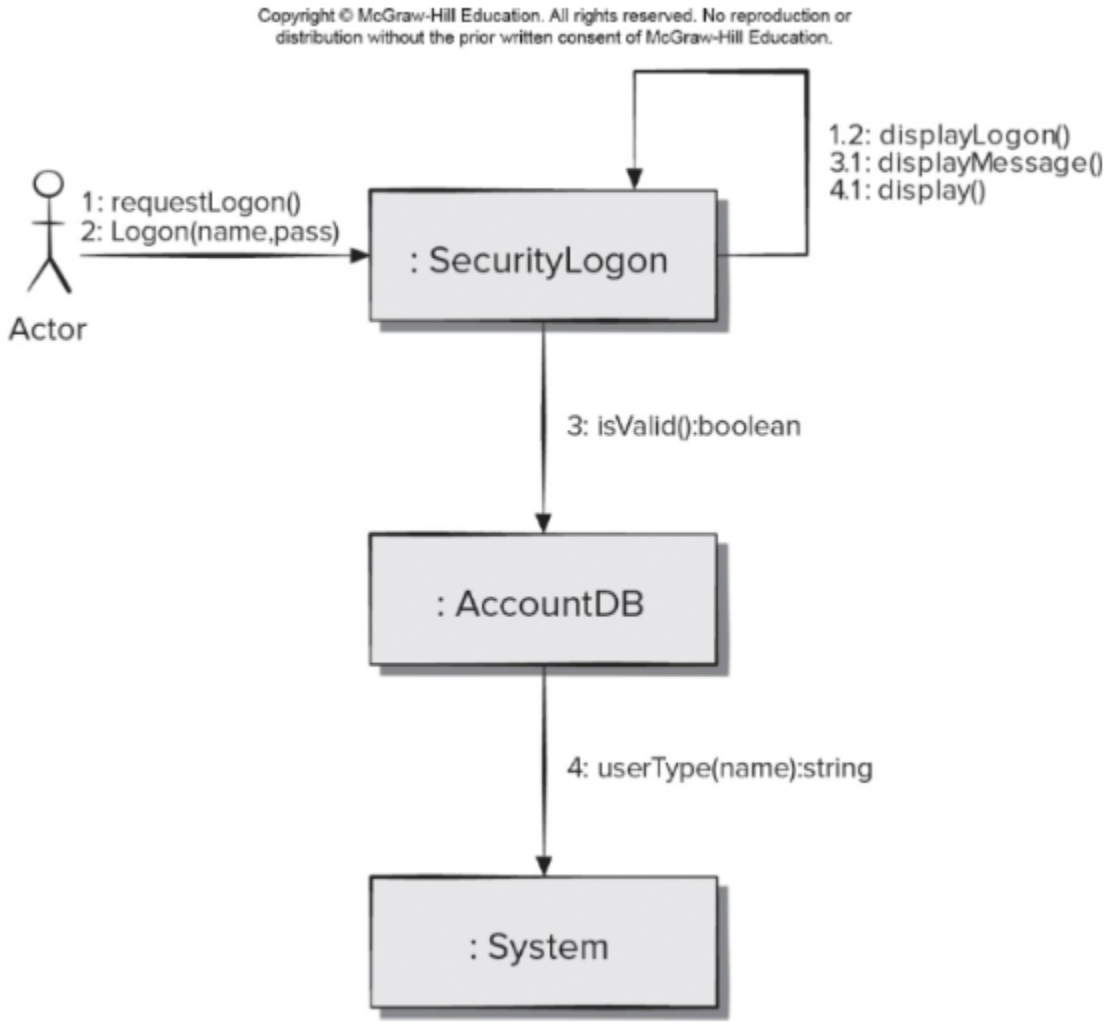
5.3.2 Pros

1. Simple to analyze
2. Easy to grow system by adding more modules
3. Would allow easy upgrades as new cameras or sensors are added to the system

5.3.3 Cons

1. Parallel processing is difficult
2. Exceptions are difficult to handle
3. Would be difficult to run the many sensors and get information from them all at the same time, because parallel processing is difficult

5.4 Object-Oriented:



5.4.1 Architecture Description

The components of a system encapsulate data and the operations that must be applied to manipulate the data. Communication and coordination between components are accomplished via message passing.

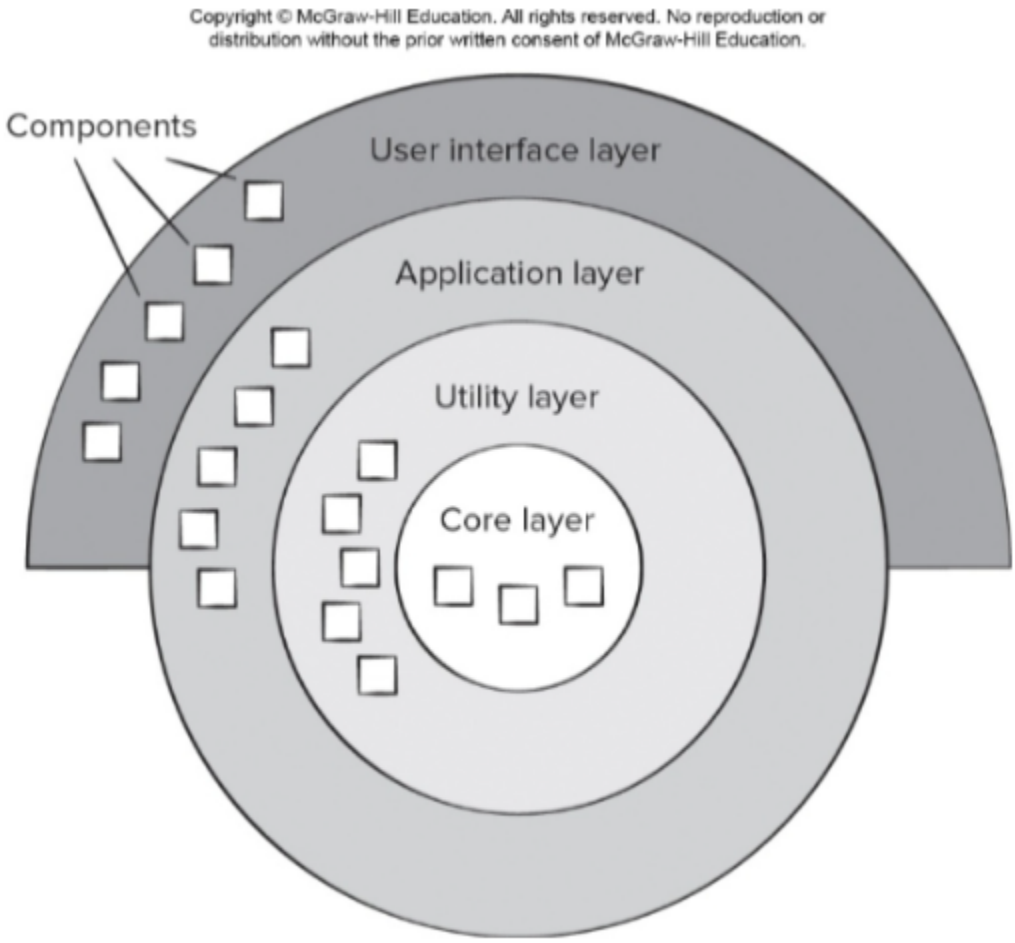
5.4.2 Pros

1. Allows for parallel processing
2. Code is easy to maintain
3. Code reusability
4. Would allow systems to use similar parts of code while running simultaneously
5. The team has all worked with object oriented programming before

5.4.3 Cons

1. Can become inefficient
2. Hard to keep costs and overhead down
3. Code for the different systems may end up being so similar it's almost a copy
4. Code may be two slow to make decisions for train fast enough

5.5 Layered:



5.5.1 Architecture Description

A layered architecture organizes the structure of the system in layers, with each layer being determined by their proximity to the machine instruction set. Outer layers would include user interface operations, whereas inner layers would include operation system interfacing, with utility services and application software functions in the intermediate lays between.

5.5.2 Pros

1. Separates the system into clear layers
2. Simple in abstraction
3. Easier to test because of separation into layers
4. Would separate sensors, user interface, and the decision making layers

5.5.3 Cons

1. May expand to have too many layers
2. Excess layers may slow performance, and fast performance is required for the IoT system

5.6 Choice of Software Architecture

Our choice of software architecture will be object oriented architecture. We chose this as it allows for parallel processing and it is easy to maintain. The structure of the IoT Engine outlined in our requirements section will most easily be transferred to an object oriented format, and the team has experience in object oriented programming. The team will be able to use our use case diagrams to create the components in this architecture, and use parallel processing to deal with multiple use cases at once.

Section 6:

6.1 Main.java:

```
import java.util.Scanner;
import javax.swing.*;
import java.awt.GraphicsEnvironment;
import java.awt.GraphicsDevice;
import java.awt.Color;

class Main{
static GraphicsDevice device = GraphicsEnvironment
       .getLocalGraphicsEnvironment().getScreenDevices()[0];
 public static void main(String[] args) {
   String username;
   String password;
   Scanner sc = new Scanner(System.in);
   String warn = "";
   String message = "";



   //login GUI
   GUI frame = new GUI();
   frame.setTitle("Login Form");
   frame.setVisible(false);
   frame.setBounds(10, 10, 370, 600);
   frame.setExtendedState(JFrame.MAXIMIZED_BOTH);
   frame.setUndecorated(true);
   device.setFullScreenWindow(frame);
   frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   frame.setResizable(false);



   while(frame.iot.getIsOn() == false) {
     System.out.println("");
     if (frame.stat) {
       System.out.println("");
       frame.displayStatus(frame.iot.turnOn());
       //frame.getContentPane().setBackground(Color.WHITE);
```

```
      }
    }
    boolean red = false;
    while(frame.iot.getIsOn()) {
      boolean warning = false;

      // Detects objects on the track
      if (frame.sens.objectOnTrack()) {
        if (!warning) frame.iot.warningSound();
        warning = true;

        if(frame.iot.getSpeed() > 0) {
          warn = "Warning: Object on the Track. Suggestion: Slow down.";
        }
        else {
          warn = "Warning: Object on the Track.";
        }
      }

      // wheels are slipping if RPM != GPS speed
      //Converts rpm to a speed to be compared to the GPS speed
      double wheelSlippage = 1.778 * 3.14159 * 60 * frame.sens.getRPM() /
1000 / frame.iot.getSpeed();

      // Detects heavy wheel slippage
      if (wheelSlippage > 1.25 || wheelSlippage < 0.75) {
        if (!warning) frame.iot.warningSound();
        warning = true;

        warn = "Warning: Heavy wheel slippage. Suggestion: Deploy sand,
slow down.";
        frame.getContentPane().setBackground(Color.RED);
        red = true;
      } else {
        red = false;
      }

      // Detects medium wheel slippage
      if ((wheelSlippage > 1.1 && wheelSlippage <= 1.25) ||
          (wheelSlippage < 0.9 && wheelSlippage >= 0.75)) {
```

```
        if (!warning) frame.iot.warningSound();
        warning = true;
        warn = "Warning: Medium wheel slippage. Suggestion: Deploy sand,
slow down.";
      }

      // Detects gates ahead
      if (frame.sens.getGateDistance() > .9 && frame.sens.getGateDistance()
< 1.60934) {
        if (!warning) frame.iot.warningSound();
        warning = true;
        warn = "Info: Gate found. Suggestion: Blow horn for 15 seconds.";
      }

      // Detects arriving at a gate
      if (frame.sens.getGateDistance() < .1) {
        message = "Info: Arrived at a gate. Suggestion: Blow horn for 5
seconds.";
      }

      // Detects incoming objects
      if (frame.sens.nearbyObjectSpeed() > 0) {
        if (!warning) frame.iot.warningSound();
        warning = true;
        warn = "Warning: Object incoming at "
                            + frame.sens.nearbyObjectSpeed() +
                            "kph. Suggestion: Slow down.";

      }

      // Detects objects moving away from the train
      if (frame.sens.nearbyObjectSpeed() < 0) {
        warning = true;
        warn = "Info: Object outgoing at "
                            + frame.sens.nearbyObjectSpeed();
      }
      frame.displaySpeed(frame.iot.getSpeed());
      frame.displayRain();
      frame.displayRPM();
      frame.displayObj();
```

```
      frame.displayGateOpen();
      frame.displayGateDistance();

      if (warning == false) {
        frame.fullyOperational("Status: Fully Operational.");
        frame.getContentPane().setBackground(Color.WHITE);
      } else {
        frame.displayStatus(warn);
        if (!red) frame.getContentPane().setBackground(Color.YELLOW);
      }
    }
  }
}
```

6.2 IOTSystem.java:

```
import java.util.*;
import java.io.*;

public class IOTSystem{
 private boolean is_on;
 private double speed;
 private Map<String,String> login;

 public IOTSystem(){
   is_on = false;
   speed = 0;
   login = new HashMap<String,String>();
   login.put("conductor","password1");
   login.put("admin","password2");
 }

 public double getSpeed() {
   return speed;
 }

 //Applies brakes
 public String applyBrakes(double amount){
   double cur_speed = getSpeed();
   if(amount > speed){
```

```
    amount = speed;
  }
  decreaseSpeed(amount);

  if (getSpeed() == cur_speed) {
    return "Warning: Break failure.";
  }
  return "Brakes applied.";
}

//Suggests that the conductor deploy sand
public String deploySand(){
  return "Deploy Sand.";
}

//Measures increase in speed
public String increaseSpeed(double amount){
  double origSpeed = getSpeed();
  speed += amount;
  if (getSpeed() == origSpeed) {
    return "Warning: No acceleration.";
  }
  return "Speed increased.";
}

//Measures decrease in speed
private String decreaseSpeed(double amount){
  double origSpeed = getSpeed();
  if(amount == 0){
    return "Cannot slow down, already stopped.";
  }
  speed -= amount;
  if (getSpeed() == origSpeed) {
    return "Warning: No acceleration.";
  }
  return "Speed decreased.";
}

//Measures the train breaking
public String stopTrain(){
```

```
    return applyBrakes(getSpeed());
  }

  //Turns on the IoT system
  public String turnOn(){
    is_on = true;
    return "Turning on.";
  }

  //Turns off the IoT system
  public String turnOff(){
    is_on = false;
    return "Turning off.";
  }

  //Checks if the IoT system is on
  public boolean getIsOn(){
    return is_on;
  }

  //Checks login
  public String containsLogin(String username, String password) {
    String psd = login.get(username);
    if (psd != null && psd.equals(password)) {
      return username;
    }
    else
      return "";
  }

  //Creates warning sound
  public void warningSound(){
    System.out.println("beep beep");
  }
}
```

6.3 Sensors.java:

```
import java.util.*;

public class Sensors {
 private boolean data;
 private boolean rain;
 private double currentDistance;
 private double prevDistance;
 private double RPM;
 private double gateDistance;
 private boolean gateOpen;
 private boolean obstacleDetectionSensor;
 private double railFrictionSensor;
 private double timeOfFlightSensor;
 private boolean camera;

 //Initializes sensors
 public Sensors() {
   prevDistance = 0;
   currentDistance = 0;
   gateDistance = 10000000;
   data = false;
 }

 public void reset() {
   boolean data = false;
   boolean rain = false;
   double currentDistance = 0;
   double prevDistance = 0;
   double RPM = 0;
   double gateDistance = 0;
   boolean gateOpen = false;
   boolean obstacleDetectionSensor = false;
   double railFrictionSensor = 0;
   double timeOfFlightSensor = 0;
   boolean camera = false;
 }

 //Detects object on track
```

```
public boolean objectOnTrack() {
  return obstacleDetectionSensor;
}

//Sets whether there is an object on track, for testing purposes
public void toggleObjectOnTrack() {
  obstacleDetectionSensor = !obstacleDetectionSensor;
}

//Detects speed of nearby object via Time of Flight Sensor
public double nearbyObjectSpeed() {
  double speed = prevDistance - currentDistance;
  return speed;
}

//Checks if it is raining
public boolean isRaining() {
  return rain;
}

//Sets whether it is raining, for testing purposes
public void toggleRain() {
  rain = !rain;
}

//Sets the RPM for testing purposes
public void setRPM(double r) {
  RPM = r;
}

//Gets the RPM of the wheels
public double getRPM() {
  return RPM;
}

//Gets the distance from the gate
public double getGateDistance() {
  return gateDistance;
}
```

```
 //Sets the distance from the gate, for testing purposes
 public void setGateDistance(double g){
   gateDistance = g;
 }

 //Checks if gate is open
 public boolean isGateOpen(){
   return gateOpen;
 }

 //Sets if the gate is open, for testing purposes
 public void toggleGate(){
   gateOpen = !gateOpen;
 }

 //Checks for movement on track
 public double movementOnTrack() {
   return timeOfFlightSensor;
 }

 //Sets data for testing purposes
 public void setDistance(double d) {
   currentDistance = d;
 }

 //Sets RPM for testing purposes
  public void setRpm(double d) {
   RPM = d;
 }

 //Sets data, for testing purposes
 public void set_data(boolean data_) {
   data = data_;
 }
}
```

6.4 GUI:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.Dimension;
import javax.swing.JFrame;

public class GUI extends JFrame implements ActionListener {
   //sensors
   Sensors sens = new Sensors();
   IOTSystem iot = new IOTSystem();



   Container container = getContentPane();
   Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
   //jlabels
   JLabel userLabel = new JLabel("   USERNAME");
   JLabel passwordLabel = new JLabel("   PASSWORD");
   JLabel status = new JLabel("IOT is running");
   JLabel speed = new JLabel("Speed: 0");
   JLabel rain = new JLabel("Speed: false");
   JLabel obj = new JLabel("Object On Tracks: false");
   JLabel rpm = new JLabel("RPM: 0");
   JLabel gateOpen = new JLabel("Gate Open: false");
   JLabel gateDistance = new JLabel("Gate Distance: 0");

   JTextField userTextField = new JTextField();
   JTextField textField = new JTextField();
   JPasswordField passwordField = new JPasswordField();

   //jbuttons
   JButton loginButton = new JButton("LOGIN");
   JButton brakeButton = new JButton("BRAKE");
   JButton resetButton = new JButton("RESET");
   JButton logoutButton = new JButton("LOGOUT");
   JButton incSpeedButton = new JButton("Increase Speed");
   JButton decSpeedButton = new JButton("Decrease Speed");
   JButton setRpmButton = new JButton("Set RPM");
```

```
   JButton objOnTrack = new JButton("Toggle Obj. on Track");
   JButton rainButton = new JButton("Toggle Rain");
   JButton gateButton = new JButton("Set Gate Distance");
   JButton gateOpenButton = new JButton("Toggle Gate");
   JCheckBox showPassword = new JCheckBox("Show Password");
   boolean stat = false;
   public GUI() {
       setLayoutManager();
       setLocationAndSize();
       addloginComponentsToContainer();
       addActionEvent();
   }
   public void setLayoutManager() {
       container.setLayout(null);
   }
   public void setLocationAndSize() {
       userLabel.setBounds(screenSize.width/2 - 300,
screenSize.height/2-200, 100, 30);
       passwordLabel.setBounds(screenSize.width/2 - 300,
screenSize.height/2-100, 100, 30);
       userTextField.setBounds(screenSize.width/2 - 200,
screenSize.height/2-200, 150, 30);
       passwordField.setBounds(screenSize.width/2 - 200,
screenSize.height/2-100, 150, 30);
       showPassword.setBounds(screenSize.width/2 - 200,
screenSize.height/2- 75, 150, 30);
       //buttons
       loginButton.setBounds(screenSize.width/2 + 100,
screenSize.height/2, 100, 30);
       resetButton.setBounds(screenSize.width/2 - 200,
screenSize.height/2, 100, 30);

       logoutButton.setBounds(screenSize.width/2 + 100,
screenSize.height/2, 300, 30);
       brakeButton.setBounds(screenSize.width/2 + 100, screenSize.height/2
- 50, 300, 30);
       incSpeedButton.setBounds(screenSize.width/2 + 100,
screenSize.height/2 - 100, 300, 30);
       decSpeedButton.setBounds(screenSize.width/2 + 100,
screenSize.height/2 - 150, 300, 30);
```

```
        setRpmButton.setBounds(screenSize.width/2 - 400,
screenSize.height/2 - 200, 300, 30);
        rainButton.setBounds(screenSize.width/2 - 400, screenSize.height/2,
300, 30);
        gateButton.setBounds(screenSize.width/2 - 400, screenSize.height/2
- 50, 300, 30);
        gateOpenButton.setBounds(screenSize.width/2 - 400,
screenSize.height/2 - 100, 300, 30);
        objOnTrack.setBounds(screenSize.width/2 - 400, screenSize.height/2
- 150, 300, 30);
        //logged in
        speed.setBounds(screenSize.width/2 - 200, screenSize.height/2 + 50,
1000, 300);
        rain.setBounds(screenSize.width/2 - 50, screenSize.height/2 + 50,
1000, 300);
        rpm.setBounds(screenSize.width/2 + 100, screenSize.height/2 + 50,
1000, 300);
        gateOpen.setBounds(screenSize.width/2 - 200, screenSize.height/2 +
100, 1000, 300);
        gateDistance.setBounds(screenSize.width/2 - 50, screenSize.height/2
+ 100, 1000, 300);
        obj.setBounds(screenSize.width/2 + 100,screenSize.height/2 +
100,1000,300);

        status.setBounds(screenSize.width/2 - 100, screenSize.height/2 ,
1000, 300);
        textField.setBounds(screenSize.width/2 - 50, screenSize.height/2 +
50, 100, 30);

    }

    public void addloginComponentsToContainer() {
        reset();
        container.add(userLabel);
        container.add(passwordLabel);
        container.add(userTextField);
        container.add(passwordField);
        container.add(showPassword);
        container.add(loginButton);
        container.add(resetButton);
```

```
}
public void reset(){
  userTextField.setText("");
  passwordField.setText("");
  container.removeAll();
  container.revalidate();
  container.repaint();
}

public void addDisplayConductorComponentsToContainer() {
    reset();
    //tags
    container.add(status);
    container.add(speed);
    container.add(rain);
    container.add(rpm);
    container.add(gateOpen);
    container.add(gateDistance);
    //buttons
    container.add(logoutButton);
    container.add(incSpeedButton);
    container.add(decSpeedButton);
    container.add(brakeButton);


    stat = true;
}

public void addDisplayAdminComponentsToContainer() {
    reset();
    //tags
    container.add(status);
    container.add(speed);
    container.add(rain);
    container.add(rpm);
    container.add(gateOpen);
    container.add(gateDistance);
    //buttons
    container.add(logoutButton);
    container.add(brakeButton);
```

```
    container.add(incSpeedButton);
    container.add(decSpeedButton);
    container.add(setRpmButton);
    container.add(rainButton);
    container.add(objOnTrack);
    container.add(obj);
    container.add(gateButton);
    container.add(gateOpenButton);
    //text fields
    container.add(textField);
    stat = true;
}

public void addActionEvent() {
    loginButton.addActionListener(this);
    resetButton.addActionListener(this);
    showPassword.addActionListener(this);
    logoutButton.addActionListener(this);
    brakeButton.addActionListener(this);
    incSpeedButton.addActionListener(this);
    decSpeedButton.addActionListener(this);
    setRpmButton.addActionListener(this);
    rainButton.addActionListener(this);
    objOnTrack.addActionListener(this);
    gateButton.addActionListener(this);
    gateOpenButton.addActionListener(this);
}

//Functions for changing conditions of train
public void fullyOperational(String op){
    status.setText(op);
    container.revalidate();
    container.repaint();
}

public void displaySpeed(double num){
  speed.setText("Speed: " + num + " mph");
}

public void displayRain(){
```

```
      rain.setText("Rain: " + sens.isRaining());
    }

    public void displayRPM(){
      rpm.setText("RPM: " + sens.getRPM());
    }

    public void displayStatus(String change){
      status.setText(change);
    }

    public void displayGateOpen(){
      gateOpen.setText("Gate Open: " + sens.isGateOpen());
    }

    public void displayGateDistance(){
      if (sens.getGateDistance() > 100) {
        gateDistance.setText("Gate Distance: n/a");
      } else {
        gateDistance.setText("Gate Distance: " + sens.getGateDistance() + "
mi");
      }

    }

    public void displayObj(){
      obj.setText("Object On Track: " + sens.objectOnTrack());
    }

    public static double round(double value, int places) {
      double scale = Math.pow(10, places);
      return Math.round(value * scale) / scale;
    }
    // Interaction with the GUI
    @Override
    public void actionPerformed(ActionEvent e) {
        //Coding Part of LOGIN button
        if (e.getSource() == loginButton) {
            String userText;
            String pwdText;
```

```
            userText = userTextField.getText();
            pwdText = passwordField.getText();
            String login = iot.containsLogin(userText, pwdText);
            if (login.equals("conductor")) {
                JOptionPane.showMessageDialog(this, "Conductor Login
Successful");
                addDisplayConductorComponentsToContainer();
            } else if(login.equals("admin")) {
                JOptionPane.showMessageDialog(this, "Admin Login
Successful");
                addDisplayAdminComponentsToContainer();
            }
            else{
                JOptionPane.showMessageDialog(this, "Invalid Username or
Password");
            }
        }

        if (e.getSource() == logoutButton) {
          iot.turnOff();
          reset();
        }
        //Coding Part of RESET button
        if (e.getSource() == resetButton) {
            userTextField.setText("");
            passwordField.setText("");
        }
       //Coding Part of showPassword JCheckBox
        if (e.getSource() == showPassword) {
            if (showPassword.isSelected()) {
                passwordField.setEchoChar((char) 0);
            } else {
                passwordField.setEchoChar('*');
            }
        }
        //Coding Part of BRAKE button
        if (e.getSource() == brakeButton) {
          displayStatus(iot.stopTrain());
          sens.setRPM(0);
        }
```

```
if (e.getSource() == incSpeedButton) {
  iot.increaseSpeed(5);
  sens.setRPM(round(iot.getSpeed()*1000/60/1.778/3.14159, 3));
}

if (e.getSource() == decSpeedButton) {
  iot.applyBrakes(5);
  sens.setRPM(round(iot.getSpeed()*1000/60/1.778/3.14159, 3));
}

if (e.getSource() == setRpmButton) {
  String field;
  field = textField.getText();
  if (!field.equals("")) {
    double rpm = Double.parseDouble(field);
    if(rpm >= 0){
      sens.setRPM(rpm);
    }
    textField.setText("");
  }

}

if (e.getSource() == gateButton) {
  String field;
  field = textField.getText();
  if (!field.equals("")) {
    double gate = Double.parseDouble(field);
    if(gate >= 0){
      sens.setGateDistance(gate);
    }
    textField.setText("");
  }
}

if (e.getSource() == rainButton) {
  sens.toggleRain();
}
```

```
        if(e.getSource() == gateOpenButton){
          sens.toggleGate();
        }

        if (e.getSource() == objOnTrack) {
          sens.toggleObjectOnTrack();
        }
    }
}
```

Section 7:

7.1 Scenario-based Testing

7.1.1 Test 1

Use case:
Input:
- Username:
- Password:
- Train Speed:
- Wheel RPM:
- Rain:
- Gate Distance:
- Object:
- Object Speed:

Expected Output:
- Light colors:
- Messages:

7.1.1 Test 1

Use case: Case 1
Input:
- Username: Conductor
- Password: Password1
- Train Speed: None
- Wheel RPM: None
- Rain: None
- Gate Distance: None
- Object: None
- Object Speed: None

Expected Output:

- Light colors: None
- Messages:
    - Conductor Login Successful

7.1.2 Test 2

Use case: Case 1
Input:
- Username: Conductor
- Password: badPassword
- Train Speed: None
- Wheel RPM: None
- Rain: None
- Gate Distance: None
- Object: None
- Object Speed: None

Expected Output:
- Light colors: None
- Messages:
    - Invalid Username or Password

7.1.3 Test 3

Use case: Case 2
Input:
- Username: Admin
- Password: Password2
- Train Speed: None
- Wheel RPM: None
- Rain: None
- Gate Distance: None
- Object: None
- Object Speed: None
- Object Distance: None

Expected Output:
- Light colors: None
- Messages:
    - Admin Login Successful

7.1.4 Test 4

Use case: Case 2
Input:
- Username: notAdmin
- Password: Password2

- Train Speed: None
- Wheel RPM: None
- Rain: None
- Gate Distance: None
- Object: None
- Object Speed: None

Expected Output:

- Light colors: None
- Messages:
    - Invalid Username or Password

7.1.5 Test 5

Use case: Case 3

Input:

- Username: Conductor
- Password: Password1
- Train Speed: 0
- Wheel RPM: 0
- Rain: None
- Gate Distance: None
- Object: Yes
- Object Speed: 0
- Object Distance: 1 km

Expected Output:

- Light colors: Yellow
- Messages:
    - Warning: Object on the Track.

7.1.6 Test 6

Use case: Case 3

Input:

- Username: Conductor
- Password: Password1
- Train Speed: 50 kph
- Wheel RPM: 2,850
- Rain: None
- Gate Distance: None
- Object: Yes
- Object Speed: 0
- Object Distance: 50 meters

Expected Output:

- Light colors: Red
- Messages:
    - Warning: Object on the Track. Suggestion: Slow down.

7.1.7 Test 7

Use case: Case 4
Input:
- Username: Conductor
- Password: Password1
- Train Speed: 60 kph
- Wheel RPM: 3,420
- Rain: None
- Gate Distance: None
- Object: Yes
- Object Speed: -2 kph
- Object Distance: 1 km

Expected Output:
- Light colors: Yellow
- Messages:
    - Warning: Object incoming at 2 kpm. Suggestion: Slow down.

7.1.8 Test 8

Use case: Case 4
Input:
- Username: Conductor
- Password: Password1
- Train Speed: 60 kph
- Wheel RPM: 3,420
- Rain: None
- Gate Distance: None
- Object: Yes
- Object Speed: -50 kph
- Object Distance: 0.5 km

Expected Output:
- Light colors: Red
- Messages:
    - Warning: Object incoming at 50 kpm. Suggestion: Slow down.

7.1.9 Test 9

Use case: Case 5
Input:
- Username: Conductor
- Password: Password1
- Train Speed: 60 kph
- Wheel RPM: 3,420
- Rain: None
- Gate Distance: None

- Object: Yes
- Object Speed: 60 kph
- Object Distance: 2 km

Expected Output:

- Light colors: Yellow
- Messages:
    - Info: Object outgoing at 60 kpm

7.1.10 Test 10

Use case: Case 5

Input:

- Username: Conductor
- Password: Password1
- Train Speed: 60 kph
- Wheel RPM: 3,420
- Rain: None
- Gate Distance: None
- Object: Yes
- Object Speed: 10 kph
- Object Distance: 0.75 km

Expected Output:

- Light colors: Red
- Messages:
    - Warning: Object incoming at 10 kph. Suggestion: Slow down.

7.1.11 Test 11

Use case: Case 6

Input:

- Username: Conductor
- Password: Password1
- Train Speed: 60 kph
- Wheel RPM: 3,420
- Rain: None
- Gate Distance: 1.60934 km
- Object: No
- Object Speed: None
- Object Distance: None

Expected Output:

- Light colors: None
- Messages:
    - Info: Gate found. Suggestion: Blow horn for 15 seconds.

7.1.12 Test 12

Use case: Case 6
Input:
- Username: Conductor
- Password: Password1
- Train Speed: 60 kph
- Wheel RPM: 3,420
- Rain: None
- Gate Distance: 0
- Object: No
- Object Speed: None
- Object Distance: None

Expected Output:
- Light colors: None
- Messages:
    - Info: Arrived at a gate. Suggestion: Blow horn for 5 seconds.

7.1.13 Test 13

Use case: Case 7
Input:
- Username: Conductor
- Password: Password1
- Train Speed: 60 kph
- Wheel RPM: 3,761
- Rain: None
- Gate Distance: None
- Object: No
- Object Speed: None
- Object Distance: None

Expected Output:
- Light colors: Yellow
- Messages:
    - Warning: Medium wheel slippage. Suggestion: Deploy sand, slow down.

7.1.14 Test 14

Use case: Case 7
Input:
- Username: Conductor
- Password: Password1
- Train Speed: 60 kph
- Wheel RPM: 5,471
- Rain: None

- Gate Distance: None
- Object: No
- Object Speed: None
- Object Distance: None

Expected Output:

- Light colors: Red
- Messages:
    - Warning: Heavy wheel slippage. Suggestion: Deploy sand, slow down.

7.1.14 Test 15

Use case: Case 8

Input:

- Username: Conductor
- Password: Password1
- Train Speed: None
- Wheel RPM: None
- Rain: None
- Gate Distance: None
- Object: None
- Object Speed: None
- Object Distance: None
- Turn Off: Yes

Expected Output:

- Light colors: None
- Observe that the system shuts off after turning it off.

7.2 Validation Testing

7.2.1 Test 1

Requirement: R-1

Input:

- Username: Conductor
- Password: Password1
- Train Speed: None
- Wheel RPM: None
- Rain: None
- Gate Distance: None
- Object: None
- Object Speed: None

Expected Output:

- Light colors: None

- Messages:
    - Conductor Login Successful

7.2.2 Test 2

Requirement: R-1
Input:
- Username: Conductor
- Password: badPassword
- Train Speed: None
- Wheel RPM: None
- Rain: None
- Gate Distance: None
- Object: None
- Object Speed: None

Expected Output:
- Light colors: None
- Messages:
    - Invalid Username or Password

7.2.3 Test 3

Requirement: R-2
Input:
- Username: Admin
- Password: Password2
- Train Speed: None
- Wheel RPM: None
- Rain: None
- Gate Distance: None
- Object: None
- Object Speed: None
- Object Distance: None

Expected Output:
- Light colors: None
- Messages:
    - Admin Login Successful

7.2.4 Test 4

Requirement: R-2
Input:
- Username: notAdmin
- Password: Password2
- Train Speed: None

- Wheel RPM: None
- Rain: None
- Gate Distance: None
- Object: None
- Object Speed: None

Expected Output:

- Light colors: None
- Messages:
    - Invalid Username or Password

7.2.5 Test 5

Requirement: R-4, R-5, R-6, R-31
Repeat inputs and check output 10,000 times
Input:

- Username: Conductor
- Password: Password1
- Train Speed: 60 kph
- Wheel RPM: 6,471
- Rain: Yes
- Gate Distance: 1.60934 kilometers
- Object: Yes
- Object Speed: 0
- Object Distance: 50 meters

Expected Output:

- Light colors: Red
- Messages:
    - Info: Gate found. Suggestion: Blow horn for 15 seconds.
    - Warning: Heavy wheel slippage. Suggestion: Deploy sand, slow down.
- Observe that the time for the message to appear is less than half a second after updating the inputs.
- Observe that the system does not slow down with additional inputs.
- Observe that the system is able to read data from every sensor simultaneously.
- Observer that the system has the correct outputs at least 9,999 times

7.2.6 Test 6

Requirement: R-7, R-8, R-9, R-10, R-11, R-12, R-13, R-19
Repeat inputs and check output 10,000 times
Input:

- Username: Conductor
- Password: Password1
- Train Speed: 60 kph to 50 kph
- Wheel RPM: 5,471 to 2,850

- Rain: None
- Gate Distance: None
- Object: Yes
- Object Speed: 0
- Object Distance: 50 feet

Expected Output:

- Light colors: Red
- Messages:
    - Warning: Object on the Track. Suggestion: Slow down.
    - Brakes applied
- Observe that the brake is applied within 0.1 second of hitting the button.
- Observe that the system detects the object on the track 99.99% of the time.
- Observe that the brake is able to be turned on and off at the push of a button.
- Observe that the system will give the conductor a suggestion on decreasing the speed when it is necessary to ensure safety because something is on the track, and alerts the driver when something is on the track, and correctly determines the best way to avoid a collision with a standing or stopped object on the track.

7.2.6 Test 7

Requirement: R-14, R-17, R-18

Input:

- Username: Conductor
- Password: Password1
- Train Speed: 60 kph
- Wheel RPM: 3,420
- Rain: None
- Gate Distance: None
- Object: Yes
- Object Speed: 70 kph
- Object Distance: 3 kilometers

Expected Output:

- Light colors: None
- Messages:
    - Info: Object outgoing at 70 kph.
- Observe that the system determines that the train does not need to brake, and the object will not interfere with the track

7.2.7 Test 7

Requirement: R-14, R-15, R-16, R-17, R-18, R-20

Input:
- Username: Conductor
- Password: Password1
- Train Speed: 60 kph
- Wheel RPM: 3,420
- Rain: None
- Gate Distance: None
- Object: Yes
- Object Speed: -5 kph
- Object Distance: 60 meters

Expected Output:
- Light colors: None
- Messages:
    - Warning: Object incoming at 5 kph. Suggestion: Slow down.
- Observe that the system determines that the train will need to brake, and the object will interfere with the track.
- Observe that the system suggests a new speed for the train to travel.

7.2.8 Test 8

Requirement: R-21, R-22, R-23, R-24

Input:
- Username: Conductor
- Password: Password1
- Train Speed: 60 kph
- Wheel RPM: 3,420
- Rain: None
- Gate Distance: 1.60934 km
- Gate Open: Yes
- Object: Yes
- Object Speed: 0
- Object Distance: 1.60934 km

Expected Output:
- Light colors: None
- Messages:
    - Info: Gate found. Suggestion: Blow horn for 15 seconds.
    - Warning: Object on the Track. Suggestion: Slow down.
- Observe that the system determines if the gate is open, if there is an object on the tracks while the gate is closing, and whether or not the train has to change speed.
- Observer that the system informs the conductor to blow the horn for 15 seconds while at 1 mile away from gate crossing

7.2.9 Test 9

Requirement: R-24

Input:
- Username: Conductor
- Password: Password1
- Train Speed: 60 kph
- Wheel RPM: 3,420
- Rain: None
- Gate Distance: 0 km
- Gate Open: No
- Object: No
- Object Speed: None
- Object Distance: None

Expected Output:
- Light colors: None
- Messages:
    - Info: Arrived at a gate. Suggestion: Blow horn for 5 seconds.
- Observe that the system determines if the gate is open, if there is an object on the tracks while the gate is closing, and whether or not the train has to change speed.
- Observer that the system informs the conductor to blow the horn for 5 seconds when it reaches the gate crossing

7.2.10 Test 10

Requirement: R-25, R-26, R-27, R-28, R-29, R-30
Repeat inputs and check output 10,000 times
Input:
- Username: Conductor
- Password: Password1
- Train Speed: 50 kph
- Wheel RPM: 5,471 to 2,850
- Rain: Yes
- Gate Distance: None
- Object: No
- Object Speed: None
- Object Distance: None

Expected Output:
- Light colors: Red to Yellow to None
- Messages:
    - Warning: Heavy wheel slippage. Suggestion: Deploy sand, slow down.
    - Warning: Medium wheel slippage. Suggestion: Deploy sand, slow.
    - Fully Operational
- Observe that the system determines if the wheels are slipping using the GPS and wheel RPM, and informs the conductor, recommending that they slow down and deploy sand.

- Observe that the system goes to fully operational when the wheels are no longer slipping.