Hug The Rail IoT Project CS347B

Team 9

“Enter Team Name Here”

Christopher Ward
Franklin Shack
Joseph Sofia
Sean Finnie

Table of Contents

1. Introduction

1.1: First Week

To start our Communication and Planning phase on 2/9, we discussed our schedules and set a weekly meeting time of 6 PM on Friday. We discussed the scope of the project and made a shared Google Drive to contain our shared documents. We then discussed and decided upon a team name, “Enter Team Name Here”, and wrote the introduction to this document. Following this, we continued to add to and refine our document as specified by Professor Reza, establishing our routine for document creation and revision.

1.2: Stakeholders

Hug The Rail Train Company (HTR) noticed safety concerns and issues with cost/inefficiency that can be addressed with IoT technologies. Specifically, the trains are susceptible to hazards on the rail when connection to cloud/fog servers is lost.

Our team, named “Enter Team Name Here”, was formed to develop IoT solutions to improve the Hug The Rail train system in CS347. Stakeholders in this project include the team members (who will earn a grade), the CAs (who will provide consulting for the group), and Professor Peyrovian (CEO of Hug the Rail Industries).

During the development of our solutions, we will focus chiefly on the safety of the product, as the systems must be robust in order to be trusted with control of the train.

1.3: Development Model

We chose to adopt the prototyping model of development for this project. We chose it because we will need to constantly iterate on our designs to make sure the product is as safe as possible. We chose this model of design because it allows us to go back and check over our work frequently as we develop new prototypes and develop our document with thorough class-based design and implementable models.

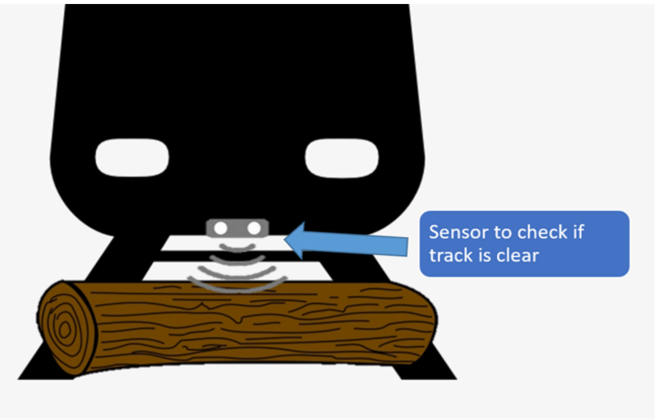
2. Overview

2.1: What is IoT?

The internet of things is the concept of individual systems (gadgets, appliances, etc) that have embedded software that allows them to connect with each other through the internet. This concept is mostly known from smart home technologies, which allow different devices in the home to network with each other and control certain functions such as temperature, turning the lights on and off, etc. IoT can upgrade normal technologies/objects to make them more “smart”—performing actions automatically and based on control systems. We will leverage IoT to create an offline network, where a central IoT computer will communicate with sensors on the train to gather information about hazards on the rail and train, as well as detect the position of railroad crossing gates, all in order to deliver warnings, and speed change/brake suggestions at a glance.

2.2: Problem Statement

HTR trains depend on WiFi/cellular network availability to receive information from Fog/Cloud servers on travel conditions, traffic, etc. The purpose of this project is to use IoT to make decisions locally on the train when WiFi/cellular has failed. This would allow trains to operate regardless of whether or not they have a network connection, and in the absence of such a connection allow them to relay the most recent information to the human conductor. If the Fog/Cloud servers detect obstructions in the railway to warn the trains, losing WiFi/cellular will leave the train vulnerable.



2.3: How IoT can help when WiFi/cellular is lost

When the train loses connection to the main server, the operator of the train will not receive important safety warnings from central fog/cloud servers. When this happens, safety and efficiency is compromised. Our IoT hazard detection sensor system can monitor safety conditions and display warnings at a glance to the operator to inform about emergencies and brake/speed change suggestions.

Our system is planned to have the following features:

- A proximity/object sensor to detect standing obstructions on track, displaying speed change/brake warnings.
- A proximity/object sensor to detect moving obstructions on track, displaying speed change/brake warnings allowing a following speed.
- A camera system to detect whether a railroad crossing gate is open or closed.
- A wheel RPM sensor working with a GPS speed sensor to detect when the wheels are slipping and display a warning/suggestion.
- Sensors on the roof to detect if the top of the train car is clear and display a warning.

The operator will have a touchscreen monitor showing sensor readings and system warnings. The IoT Computer will take in all the sensor data and, by continuously checking sensor data for hazards, will detect hazards based on our algorithms and display warnings with speed change/brake suggestions for the operator at a glance.

Without our system, trains are susceptible to hazardous conditions when stuck without a connection. Our system is passive and can always run so the operator becomes very familiar with it. Then, in the situation where the operator loses connection to fog/cloud, they will have our system to depend on for safe/efficient operation of the train.

3. Requirements

3.1: Non-Functional Requirements

3.1.1: Security

-Security is important for the system because it is mission critical when fog/cloud is lost. Therefore, the system will have a login screen so that the system can be locked when not in use. We will also use encryption to protect the IoT sensor data streams and IoT computer capabilities.

R-1: IoT HTR shall be accessed only by User ID/Password for operator usage.

R-2: IoT HTR shall have administrator login who will have control over the whole IoT Computer and sensor programs.

R-3: IoT HTR shall use encrypted connections wherever possible.

3.1.2: Reliability Requirements

-Being a hazard detection system, it is important that the sensors can work reliably. Here, we define the expected capabilities of the sensors, so that the warning system can be reliable overall.

R-4: IoT HTR should be operable under extreme weather conditions in temperatures ranging from -10 to 150 F (-24 to 66 C)

R-5: IoT HTR shall be able to withstand drops from a height of up to 5 feet (1.5 meters)

R-6: IoT HTR sensors shall be connected to the train's power

R-7: IoT HTR system shall have reliability of 0.999

3.1.3: Performance

- It is important that the warnings generated by the system can appear quickly. Therefore, it is important that the system has fast performance so the operator has time to react to the hazards.

R-8: IoT HTR shall have a response time of 0.5 second, assuming IoT has been on

R-9: IoT HTR shall be able to support up to 1,000 sensors

3.1.4: Operating System

- The IoT Computer will use an open source, reliable operating system which is widely used. This allows the IoT system to be expanded upon later since it is on a popular platform. We look for reliability, security, and performance when evaluating the operating system for the IoT Computer.

R-10: IoT HTR will use the Ubuntu distribution of Linux, using a raspberry pi for the IoT Computer.

3.2: Functional Requirements

3.2.1: Detect standing objects on the path with distance and suggest speed changes or brake.

- The IoT computer will continuously fetch data from the obstacle sensor on the front of the train. As soon as an obstacle is detected on the track, the speed of the object is fetched. If the obstacle is still, the IoT computer calculates the appropriate speed/braking suggestion based on the distance to the object and speed of the train. This suggestion is sent to the display. A warning is sent to slow down if the object is between 800 and 1,000 feet away, and to brake if it is closer than 800 feet. This suggestion is sent to the display.

R-11: IoT HTR shall use a proximity/object sensor that shall detect obstructions at a distance of at least 1,000 ft and generate speed change/brake suggestion to avoid hitting the obstacle.

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

- The IoT computer will continuously fetch data from the obstacle sensor on the front of the train. As soon as an obstacle is detected on the track, the speed of the object is fetched. If the obstacle is moving, the IoT computer calculates the appropriate speed/braking suggestion based on the distance to the object and speed of the train to maintain a safe distance of 200 ft, matching the speed of the train ahead. A warning is sent to slow down if the object is between 500 and 1,000 feet away, and to brake if it is closer than 500 feet. This suggestion is sent to the display.

R-12: IoT HTR shall use LIDAR to detect the distance and speed of the object at a distance of at least 1,000 ft.

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

- The IoT computer will continuously fetch data from the camera and compare the image to a sample of training images to differentiate between an open and closed railroad crossing gate. When a railroad crossing gate is detected, its position (open/closed) is fetched and a speed change/braking suggestion is generated. This suggestion is sent to the display.

R-14: IoT HTR shall use a camera system with at least 10 training images to provide reliable detection of whether a railroad crossing gate is open or closed at a range of 500 ft.

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

- The IoT computer will continuously fetch data from the wheel RPM and GPS speed sensors and the speed of the train is calculated from wheel length and RPM. The IoT computer continuously compares the speeds. If a difference is detected, the IoT computer will evaluate the difference and a speed change/braking suggestion is generated. This suggestion is sent to the display.

R-15: IoT HTR shall use GPS and wheel RPM detector to detect when the wheels are slipping within 1 second to generate the speed change/brake suggestion.

3.2.5: Detect obstructions on the roof of the vehicle using an object sensor.

- The IoT computer will continuously fetch data from the object sensor on the roof of the train. This sensor will detect if there is an object on top of the train, as this is dangerous. When an object is detected, the IoT Computer sends a warning to the display.

R-16: IoT HTR shall have object sensors on the roof to detect if the top of the train car is clear with 99% accuracy.

3.2.6: Recommend to honk the horn when detecting a railroad crossing.

- The IoT computer will continuously fetch data from the railroad crossing camera. When there is a railroad crossing detected within 1 mile, recommend honking the horn for 15s. Then, when the crossing is near, recommend honking the horn for 5s.

R-16: IoT HTR shall recommend to honk the horn for exactly 15 seconds at 1 mile distance, and 5 seconds at 1,000ft distance.

4. Requirements Modeling

4.1 Use Cases

4.1.1 **Use Case:** The operator logs in/out.

Primary Actor: The operator.

Secondary Actor: IoT Computer.

Goal: Enables operation of the IoT Computer and Display.

Preconditions: N/A

Trigger: The operator types their username or password, or presses a small “log out” button during operation.

Scenario:

1. The train has started up, and the IoT computer boots, displaying a login screen.
2. The operator logs in with a username/password.
3. The IoT computer begins operation.

or

1. The IoT Computer and Display are in operation.
2. The operator presses a small “logout” button.
3. The IoT Computer and Display are locked and put to a resting state.

4.1.2 **Use Case:** Lidar sensor detects obstruction on the rail ahead.

Primary Actor: Lidar Sensor.

Secondary Actor: Display, IoT Computer.

Goal: Detect an obstruction on the track within 1,000 ft and display a suggestion to slow down or to brake depending on the distance of the object and speed of the train.

Preconditions: An object is on the track. The system is powered on. The lidar sensor can reach the obstruction.

Trigger: The obstruction enters the 1,000 ft range of the lidar sensor.

Scenario:

1. An obstruction is on the train tracks.
2. The train approaches the obstruction within the range of the lidar sensor.
3. The lidar sensor detects the obstruction.
4. The speed of the train is considered by the IoT computer and sends either a brake or speed change suggestion to the display.
5. The display makes a notifying sound and shows the warning at a glance.

4.1.3 **Use Case:** Lidar sensor detects a moving object on the rail ahead.

Primary Actor: Lidar sensor.

Secondary Actor: Display, IoT Computer.

Goal: Detect a moving object on the track within 1,000 ft and display a suggestion to slow down or to brake depending on the distance of the object and speed of the train.

Preconditions: A moving object is on the track. The system is powered on. The lidar sensor can reach the moving object.

Trigger: The moving obstruction enters the 1,000 ft range of the lidar sensor.

Scenario:

1. There is another train on the tracks ahead and it is going slower.
2. The train enters the range of the lidar sensor as it is approaching.
3. The lidar sensor detects the moving object.
4. The speed of both trains are considered by the IoT computer and sends either a brake or speed change suggestion to the display such that the ahead train can be followed.
5. The display makes a notifying sound and shows the suggestion at a glance.

4.1.4 **Use Case:** Camera detects a railroad crossing ahead.

Primary Actor: Camera, GPS.

Secondary Actor: Display, IoT Computer.

Goal: Use a camera to detect if an ahead railroad crossing is open or closed, and suggest a change of speed/brake if it is closed based on data from the GPS, as well as a suggestion to blow the horn..

Preconditions: There is a railroad crossing within 1 mile. The system is powered on. The camera can see the gate. GPS data is available.

Trigger: The camera determines, using a 99.9% confidence threshold, that an ahead railroad crossing gate is either open or closed.

Scenario:

1. There is a railroad crossing ahead.
2. The crossing enters the 1 mile range of the camera.
3. If a closed gate is detected ahead, the computer sends a warning to the display suggesting speed change or brake, based on speed/position data retrieved from the GPS, as well as a suggestion to blow the horn for 15 seconds.
4. The display makes a notifying sound and shows the suggestion at a glance.

4.1.5 **Use Case:** Wheel RPM/GPS data detects wheel slippage

Primary Actor: Wheel RPM sensor, GPS.

Secondary Actor: Display, IoT Computer.

Goal: Detect when the wheels are slipping within 1 second of occurrence, and display a change of speed/brake suggestion.

Preconditions: The wheels have been slipping for at least 1 second. The system is powered on. GPS data is available.

Trigger: The IoT computer constantly checks speed from GPS and compares it to wheel RPM. Trigger when the disparity between speeds indicates wheel slippage.

Scenario:

1. The wheels have been slipping for at most 1 second.
2. The computer realizes that the speed from GPS data doesn’t match the speed from the wheel RPM sensor, and sends a change of speed or brake suggestion to the display.
3. The display makes a notifying sound and shows the suggestion at a glance.

4.1.6 **Use Case:** Object sensor detects an object on the roof of the train.

Primary Actor: Object sensor.

Secondary Actor: Display, IoT Computer.

Goal: Detect when there is an object on the roof of the train and display a warning.

Preconditions: There is an object on the roof of the car. The object is detectable by the sensor.

Trigger: The object sensor determines that there is an object on the roof.

Scenario:

1. There is an object on top of the train.
2. It is detected by the object sensor.
3. The sensor transmits a signal to the main computer.
4. The display makes a notifying sound and displays the warning at a glance.

4.1.7 **Use Case:** Recommend to honk the horn when detecting a railroad crossing.

Primary Actor: Camera.

Secondary Actor: Display, IoT Computer.

Goal: Detect when there is a railroad crossing within 1 mile and recommend to use horn for 15 seconds. Then, at 1,000 ft range, recommend honking for 5 seconds.

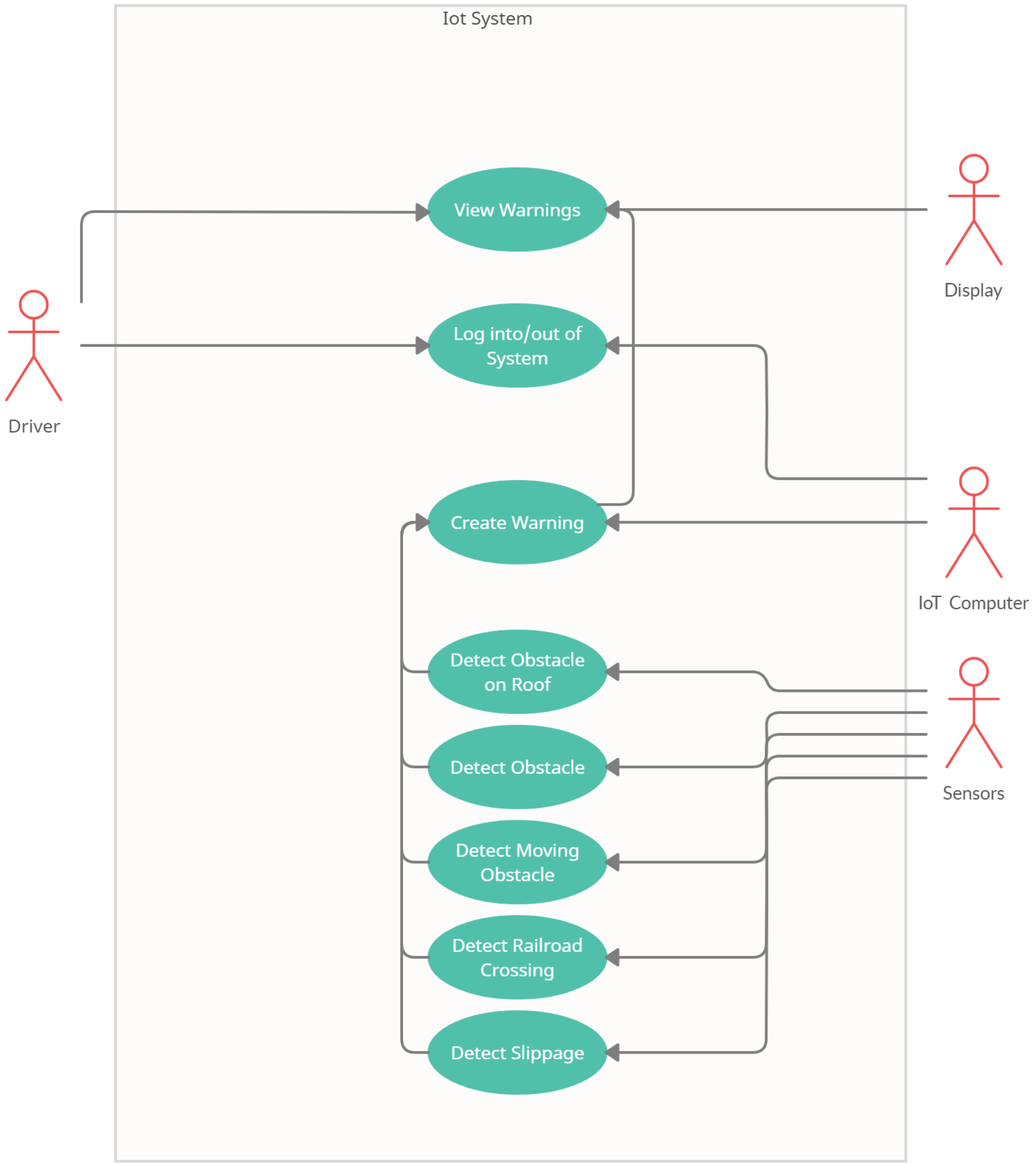
Preconditions: There is a railroad crossing within 1 mile, and the train is approaching it.

Trigger: The camera detects the presence of a railroad crossing within 1 mile.

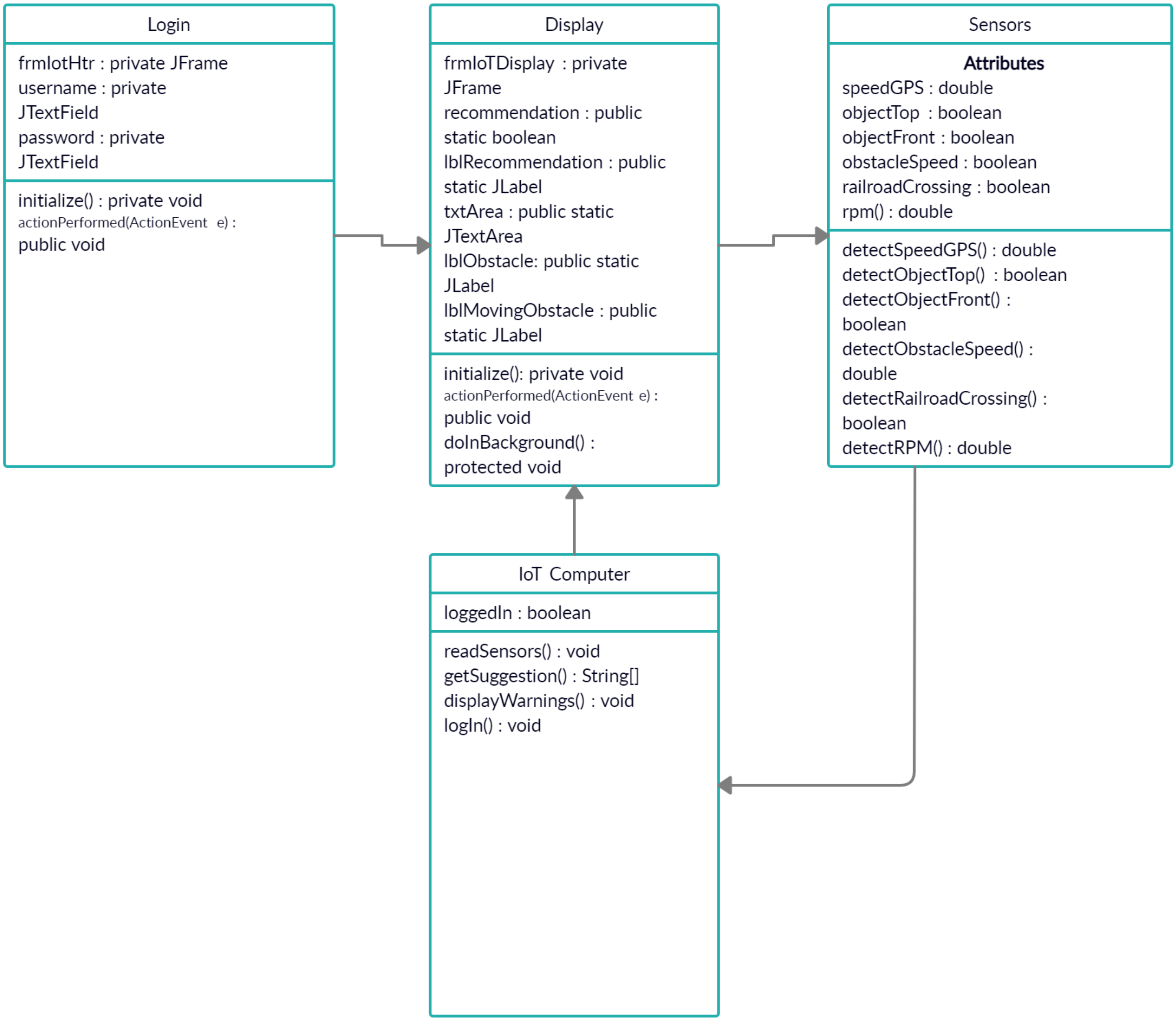
Scenario:

1. There is an object on top of the train.
2. It is detected by the object sensor.
3. The sensor transmits a signal to the main computer.
4. The display makes a notifying sound and displays the warning at a glance.

4.2 Use Case Diagram



4.3 Class-Based Modelling



4.4 CRC Modelling/Cards

Class: Sensors	
Sends sensor data to the system.	
Responsibility:	**Collaborator:**
Detect an obstacle	Proximity Sensor
Detect an obstacle on roof	Pressure Sensor
Detect a moving obstacle	LIDAR
Detect a railroad crossing	Camera
Detect train wheel slippage	GPS and wheel RPM detector

Class: IoTComputer	
Takes in data from the sensors to generate warnings and send them to the display.	
Responsibility:	**Collaborator:**
Enable / Disable system	Power Switch
Read sensor data	IoT Computer
Send warnings to display	

Class: Display	
Displays warnings and suggestions on the screen for the operator.	
Responsibility:	**Collaborator:**
Read warnings from system and displays them on screen	IoT Computer / Display

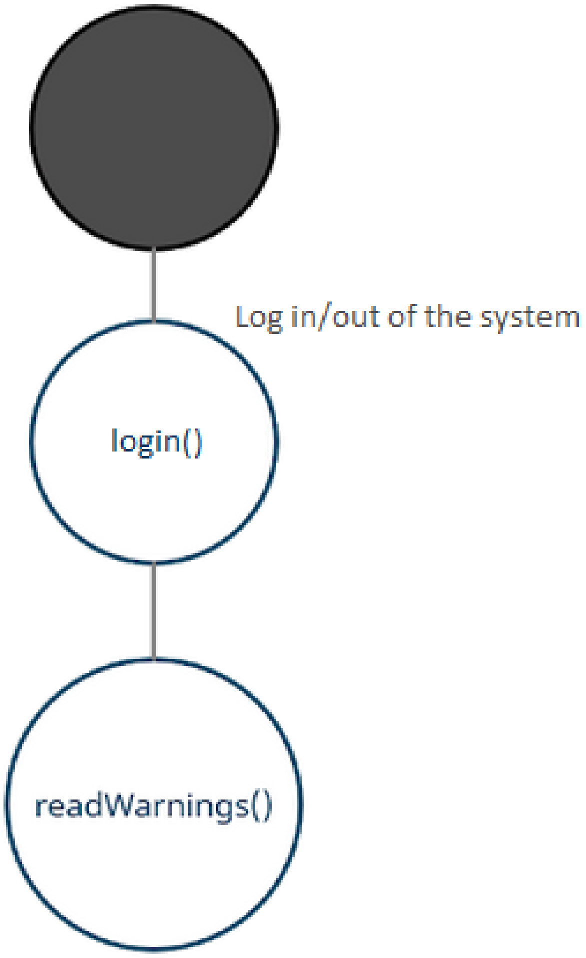
Class: Login	
Allows the operator to log in.	

Responsibility:	**Collaborator:**
Log in the conductor	IoT Computer / Display

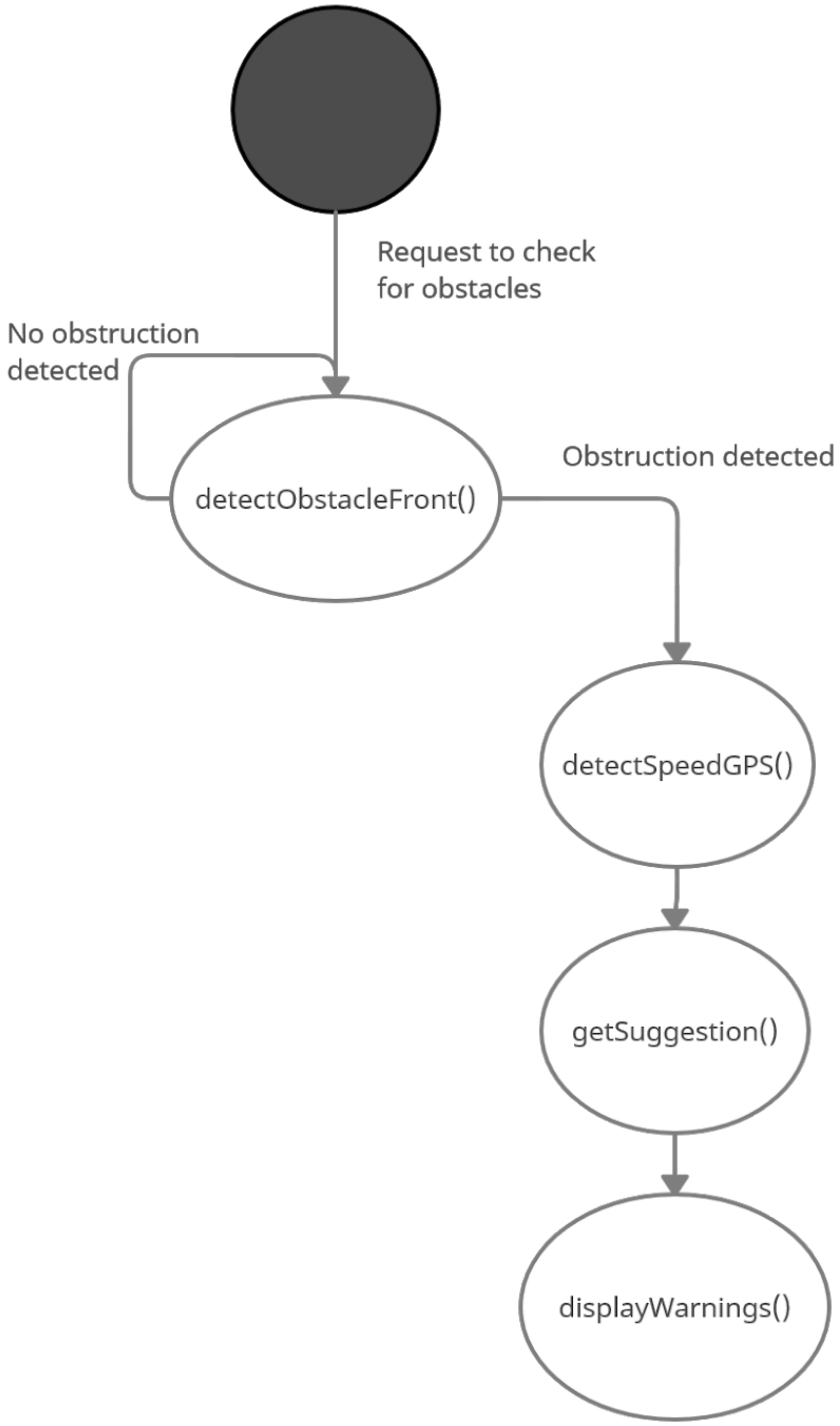
4.5 Activity Diagrams

The getSuggestion() method is a method that utilizes the Time Sensitive Networking Router in order to draw information collected from our sensors synchronously and send it to the IoT Computer in order to generate appropriate warnings in response to a variety of conditions.

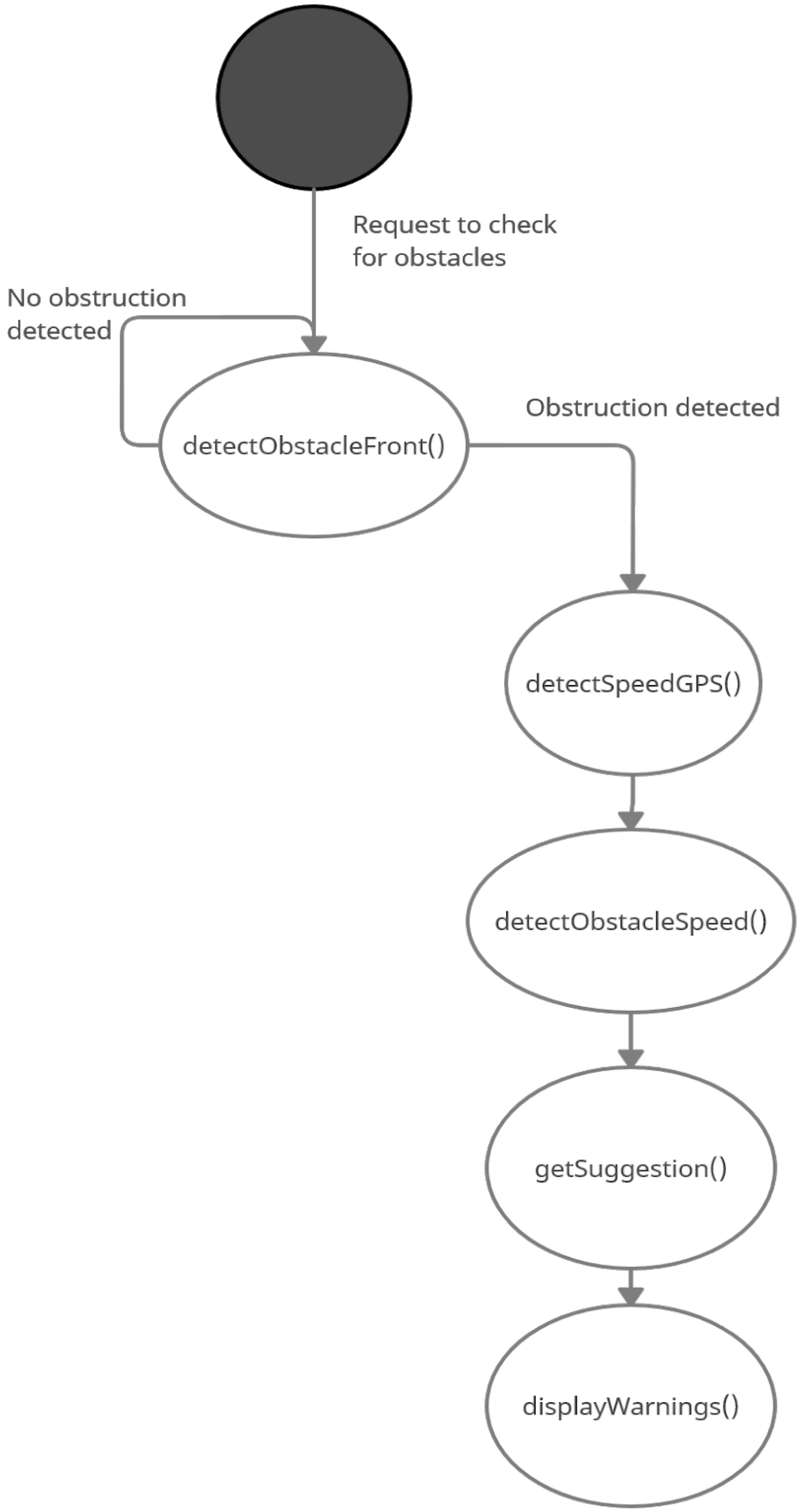
Use Case 4.5.1:



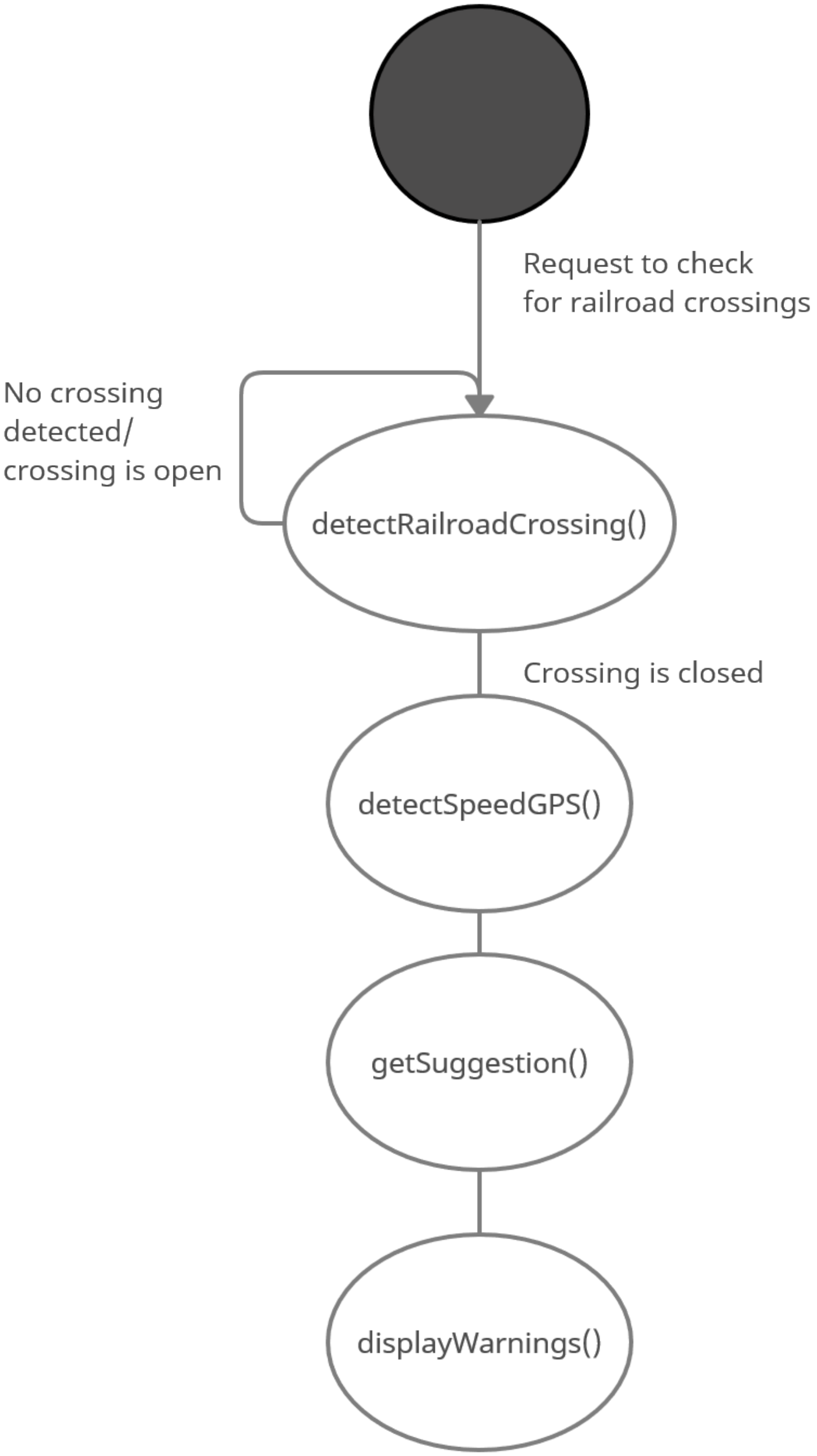
Use Case 4.5.2:


Request to check for obstacles
No obstruction detected
detectObstacleFront()
Obstruction detected
detectSpeedGPS()
getSuggestion()
displayWarnings()

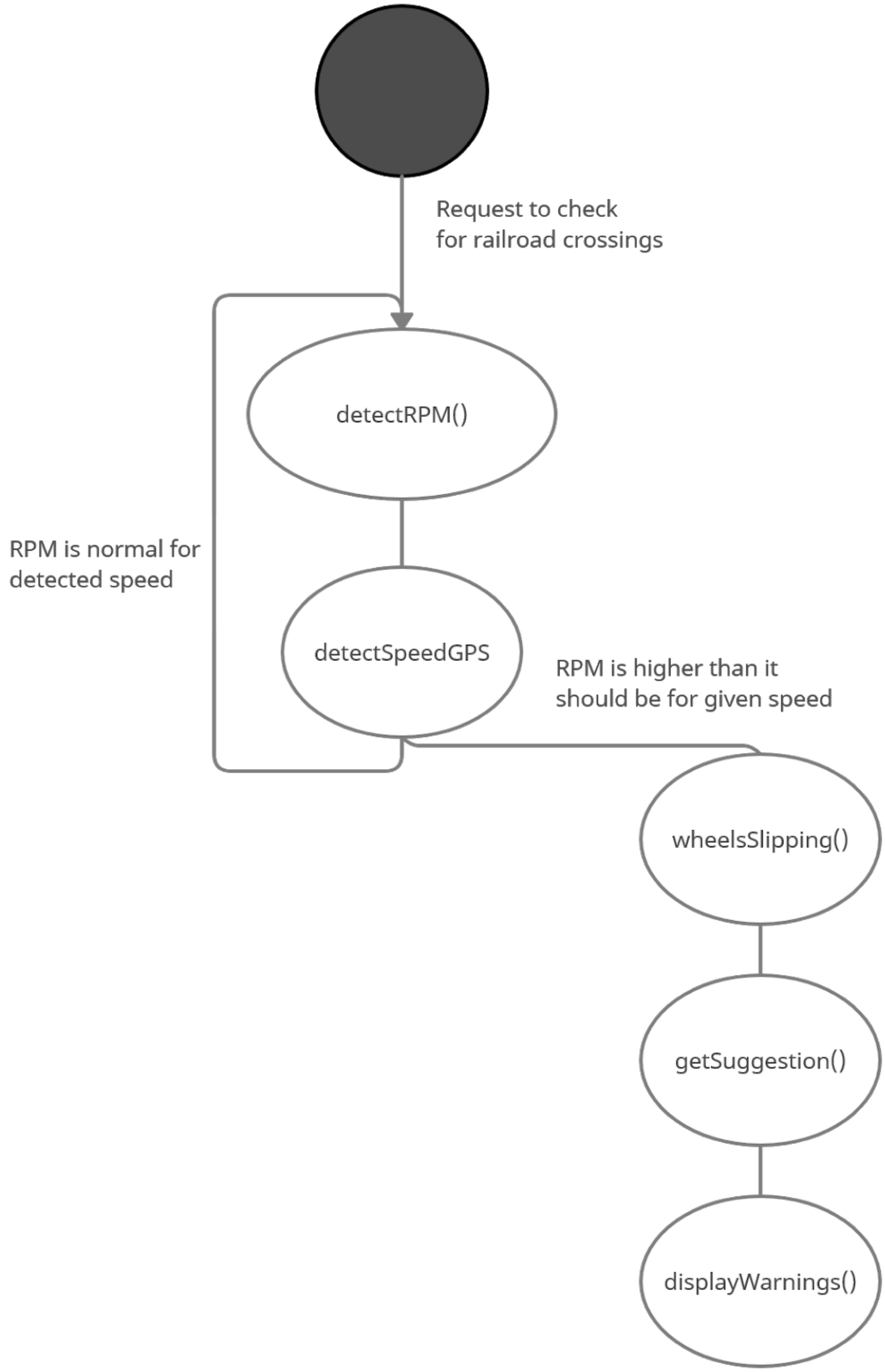
Use Case 4.5.3:


Request to check for obstacles
No obstruction detected
detectObstacleFront()
Obstruction detected
detectSpeedGPS()
detectObstacleSpeed()
getSuggestion()
displayWarnings()

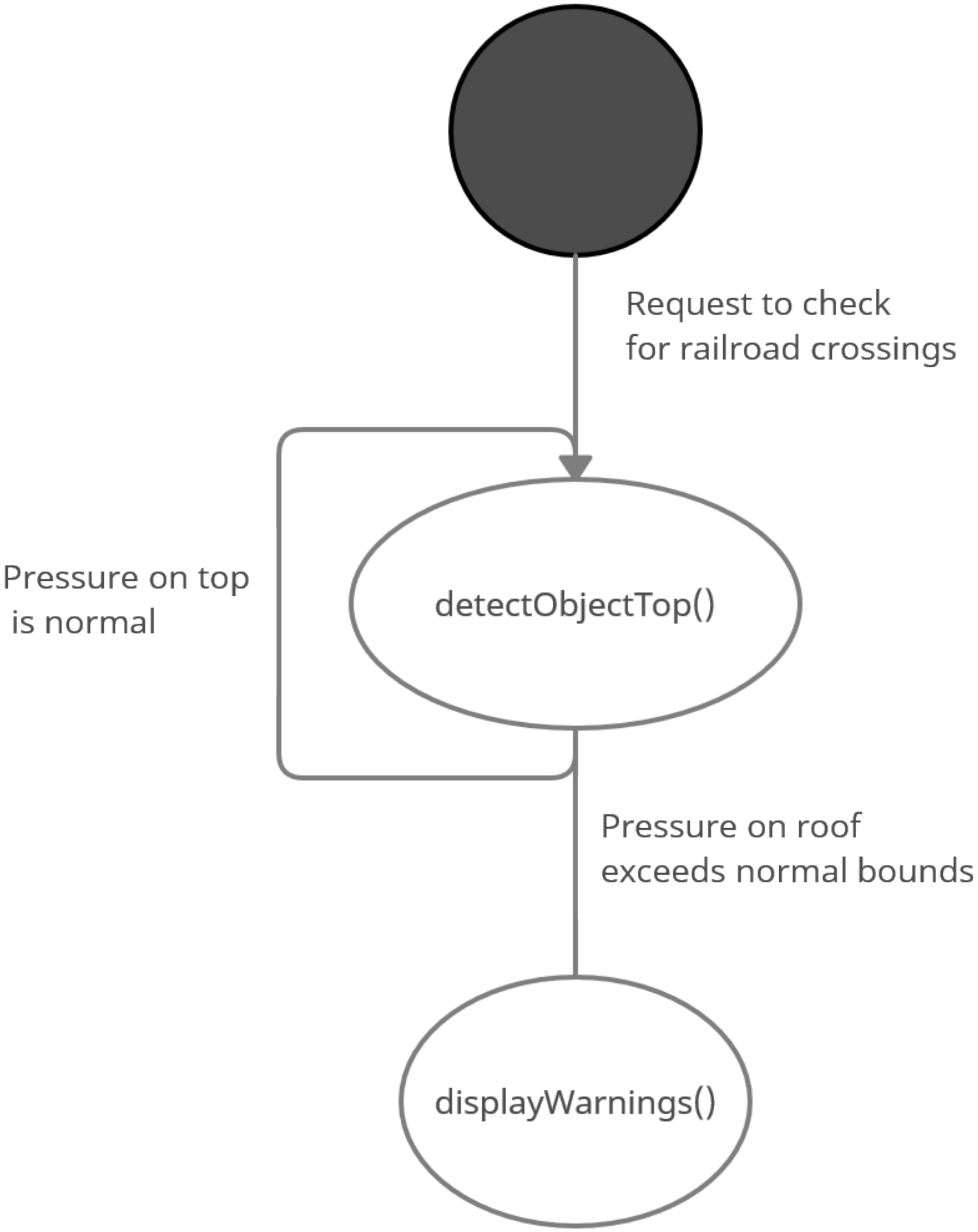
Use Case 4.5.4:


Request to check for railroad crossings
No crossing detected/ crossing is open
detectRailroadCrossing()
Crossing is closed
detectSpeedGPS()
getSuggestion()
displayWarnings()

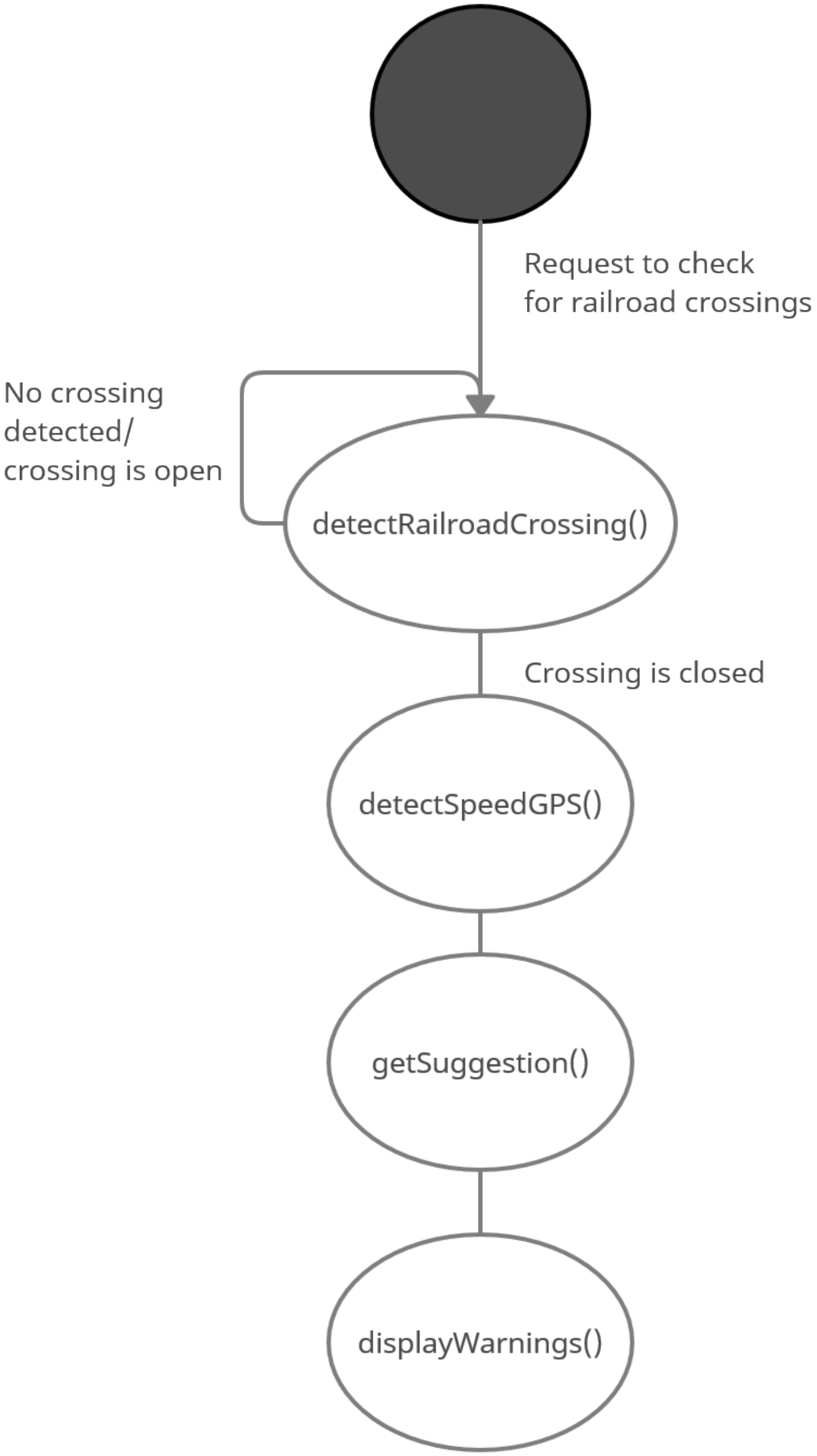
Use Case 4.5.5:


Request to check
for railroad crossings
detectRPM()
RPM is normal for
detected speed
detectSpeedGPS
RPM is higher than it
should be for given speed
wheelsSlipping()
getSuggestion()
displayWarnings()

Use Case 4.5.6:


Request to check
for railroad crossings
Pressure on top
is normal
detectObjectTop()
Pressure on roof
exceeds normal bounds
displayWarnings()

Use Case 4.5.7:

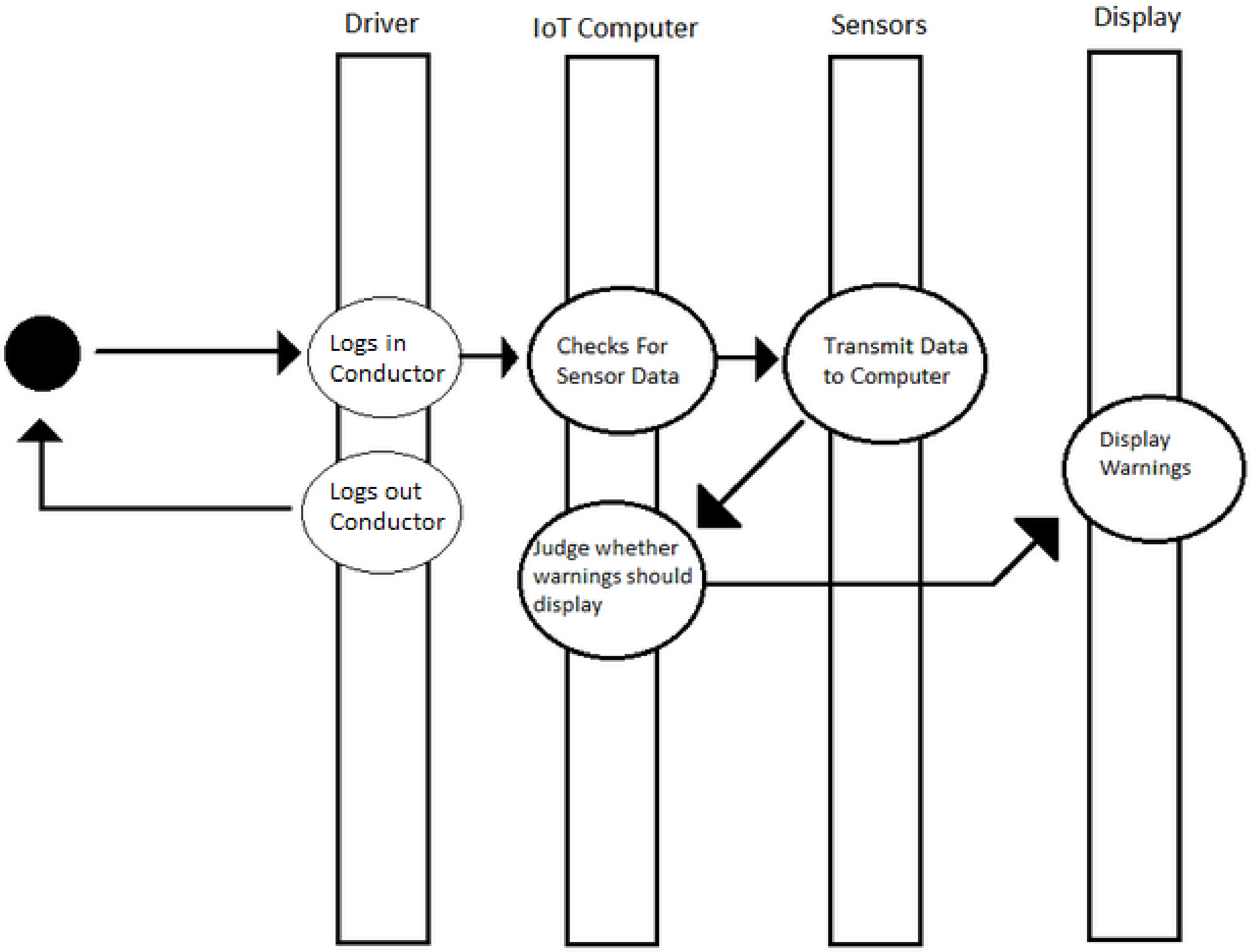


getSuggestion() handles the horn timers and distance check for the railroad crossing.

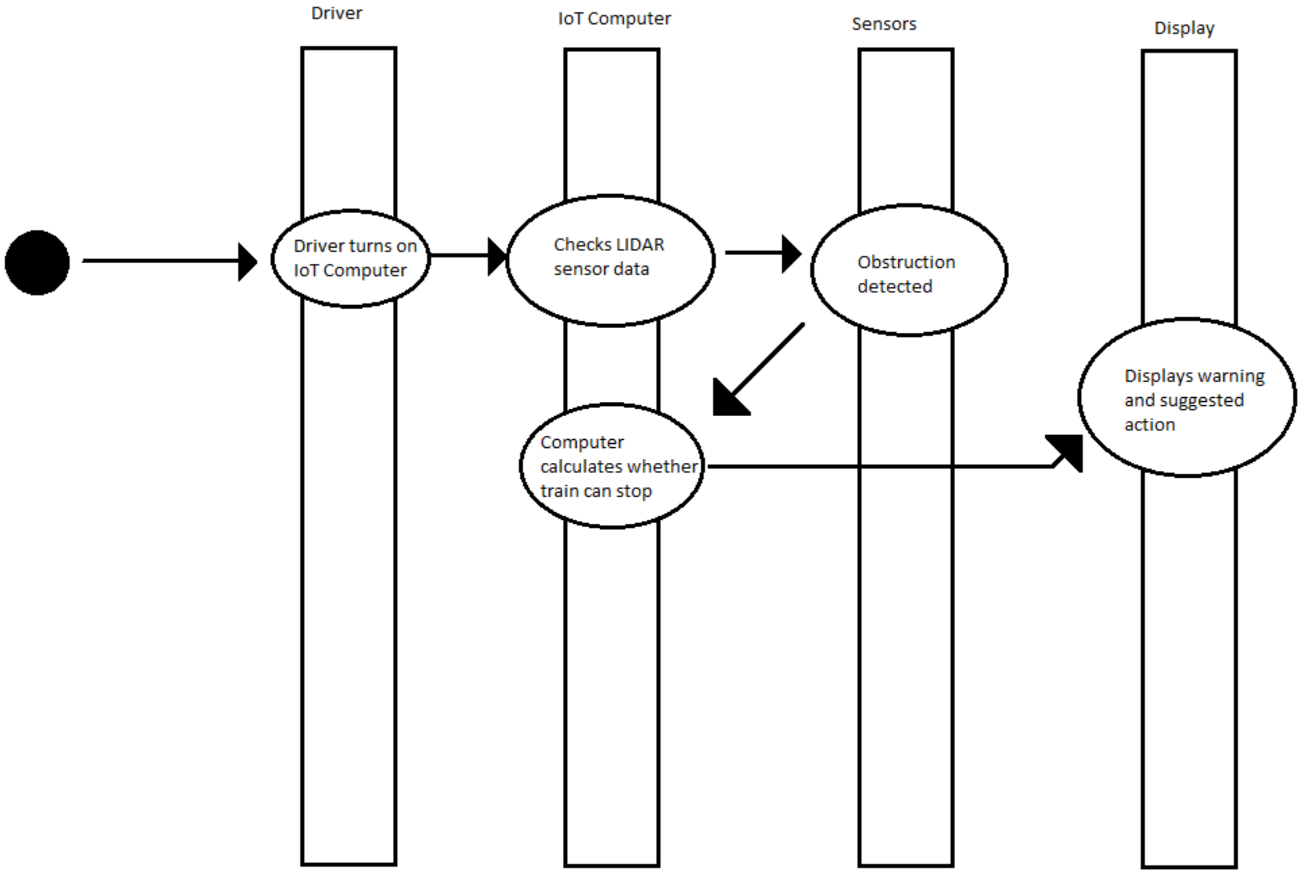
4.6 Sequence Diagrams

When the sensors transmit data to the computer, the timing is accounted for using the TSNR.

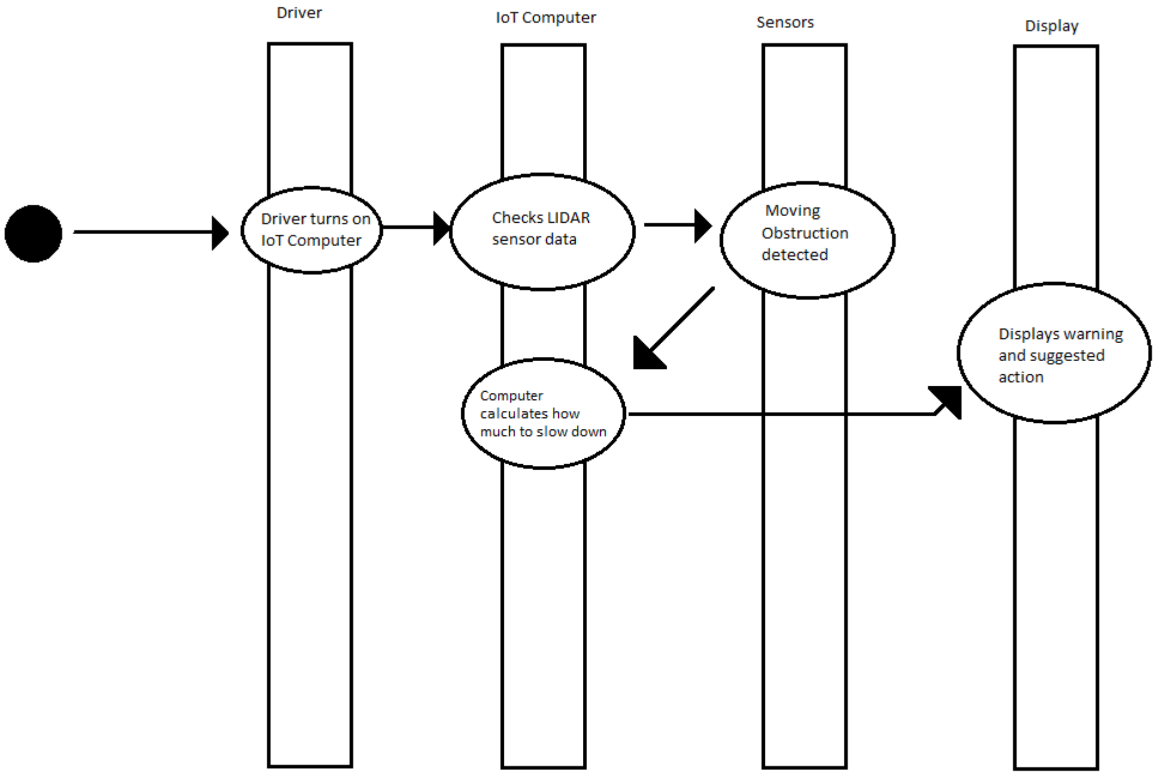
Use Case 4.6.1:



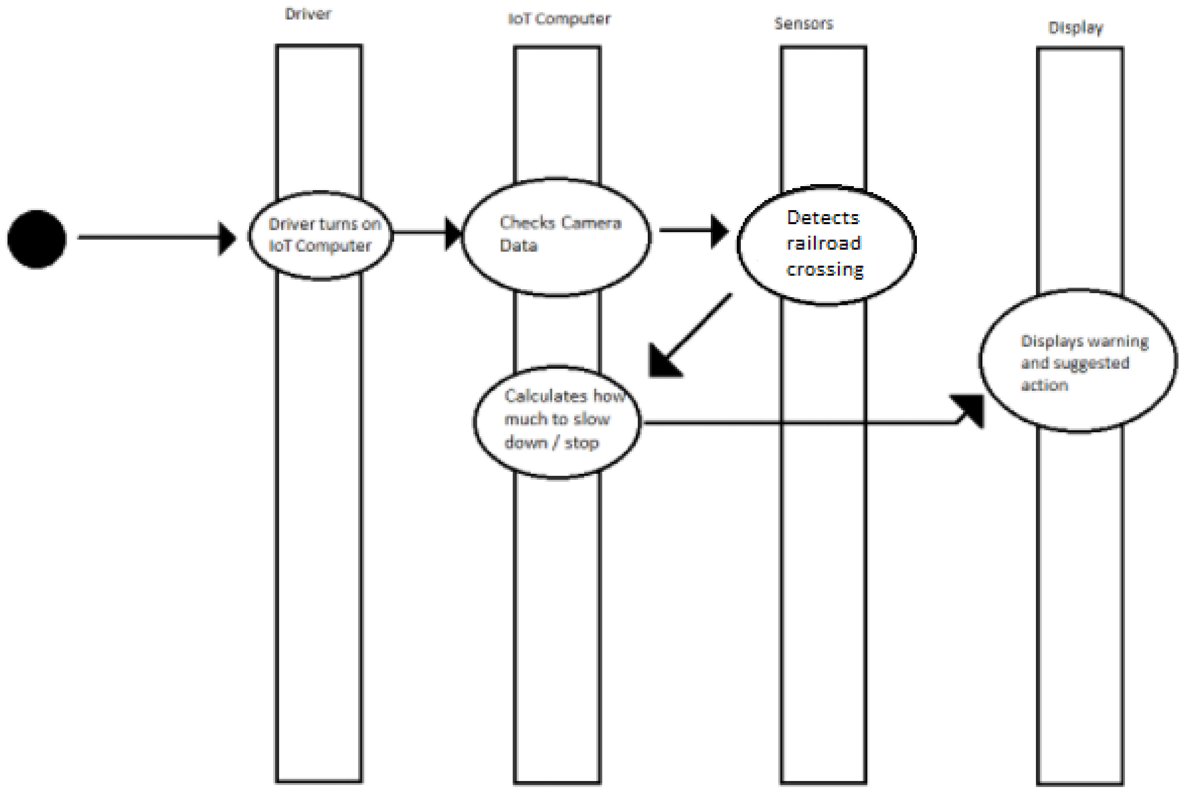
Use Case 4.6.2:


Driver
IoT Computer
Sensors
Display
Driver turns on IoT Computer
Checks LIDAR sensor data
Obstruction detected
Computer calculates whether train can stop
Displays warning and suggested action

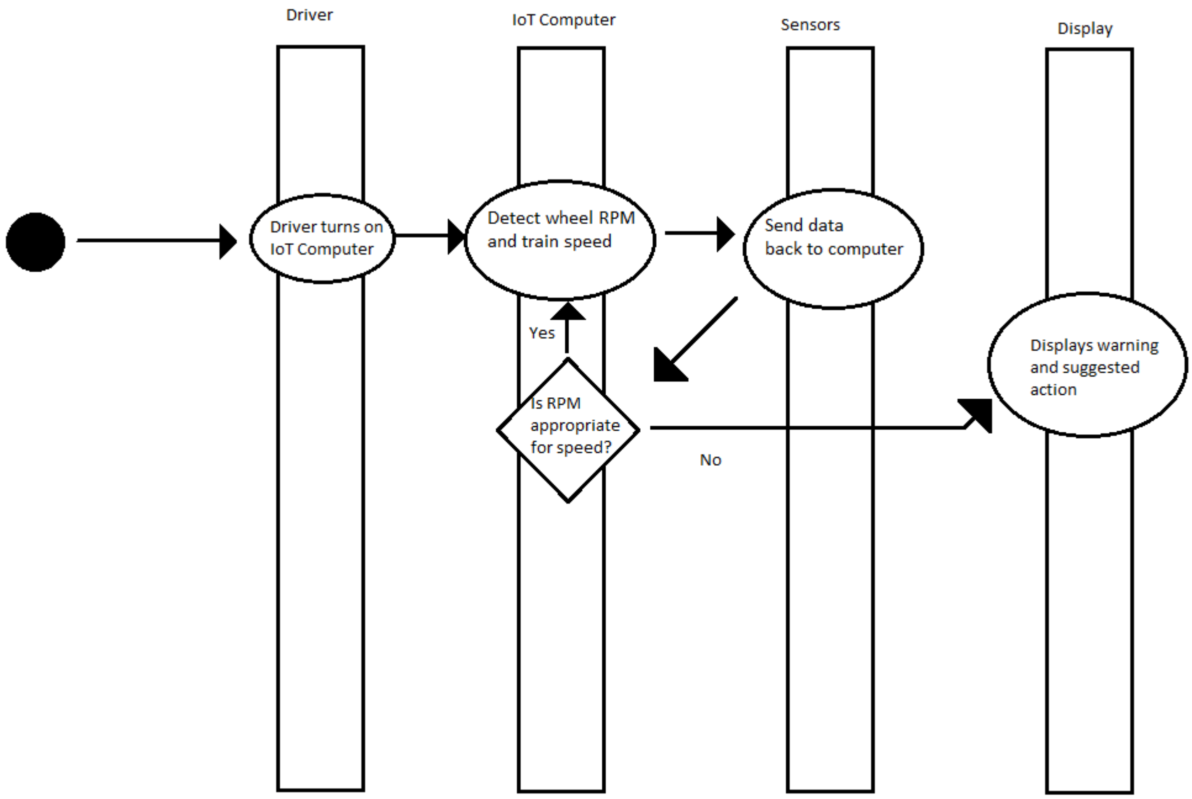
Use Case 4.6.3:


Driver
IoT Computer
Sensors
Display
Driver turns on IoT Computer
Checks LIDAR sensor data
Moving Obstruction detected
Computer calculates how much to slow down
Displays warning and suggested action

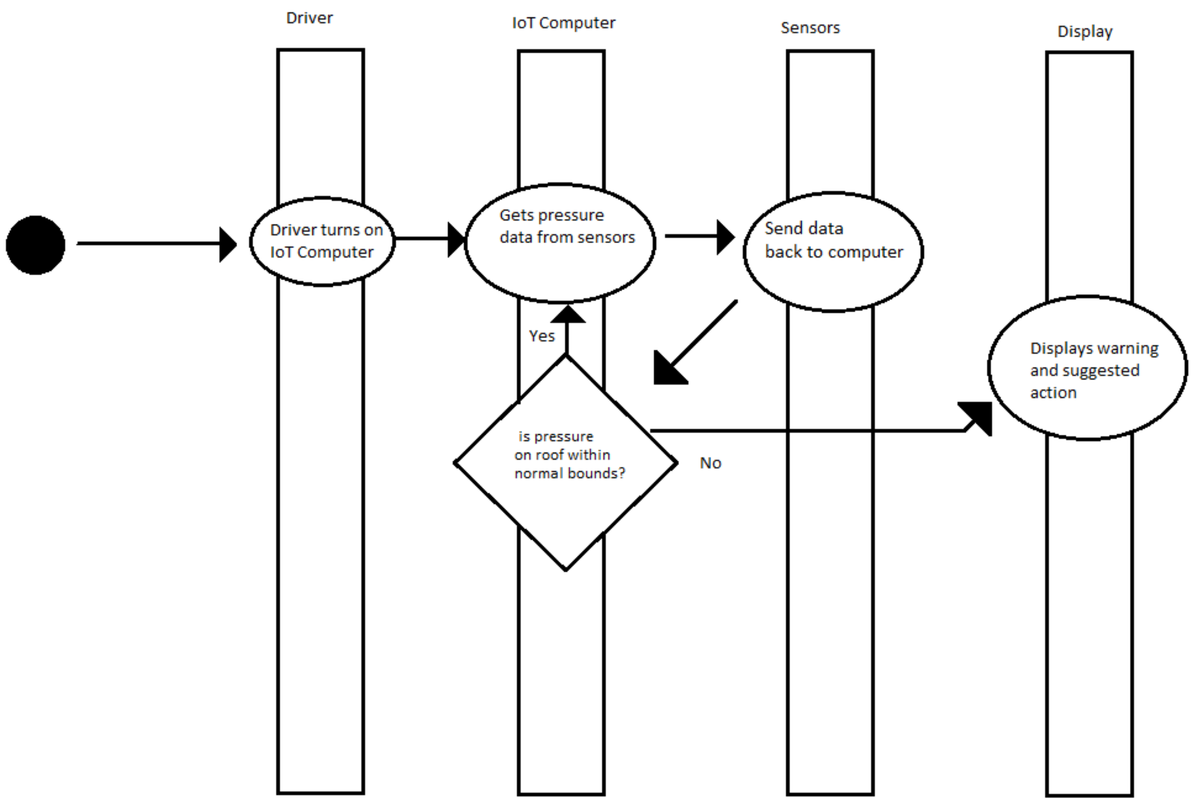
Use Case 4.6.4:


Driver
IoT Computer
Sensors
Display
Driver turns on IoT Computer
Checks Camera Data
Detects railroad crossing
Calculates how much to slow down / stop
Displays warning and suggested action

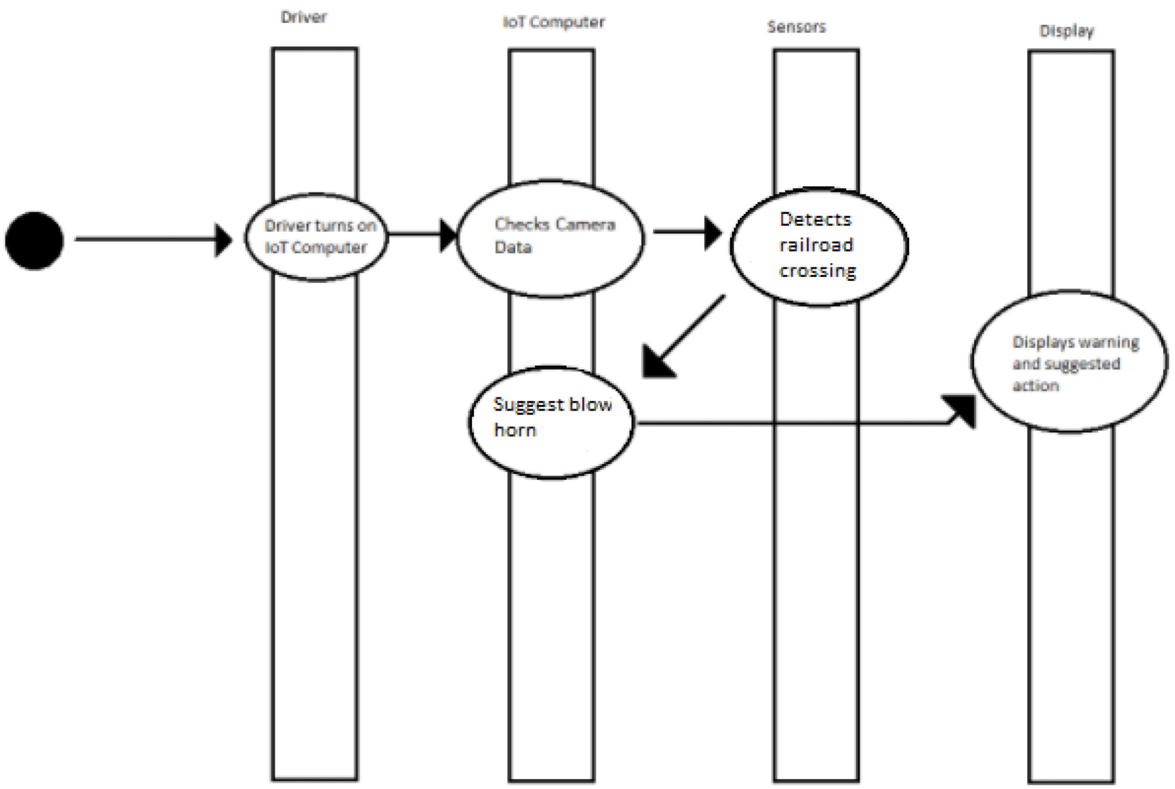
Use Case 4.6.5:


Driver
IoT Computer
Sensors
Display
Driver turns on IoT Computer
Detect wheel RPM and train speed
Send data back to computer
Yes
Is RPM appropriate for speed?
No
Displays warning and suggested action

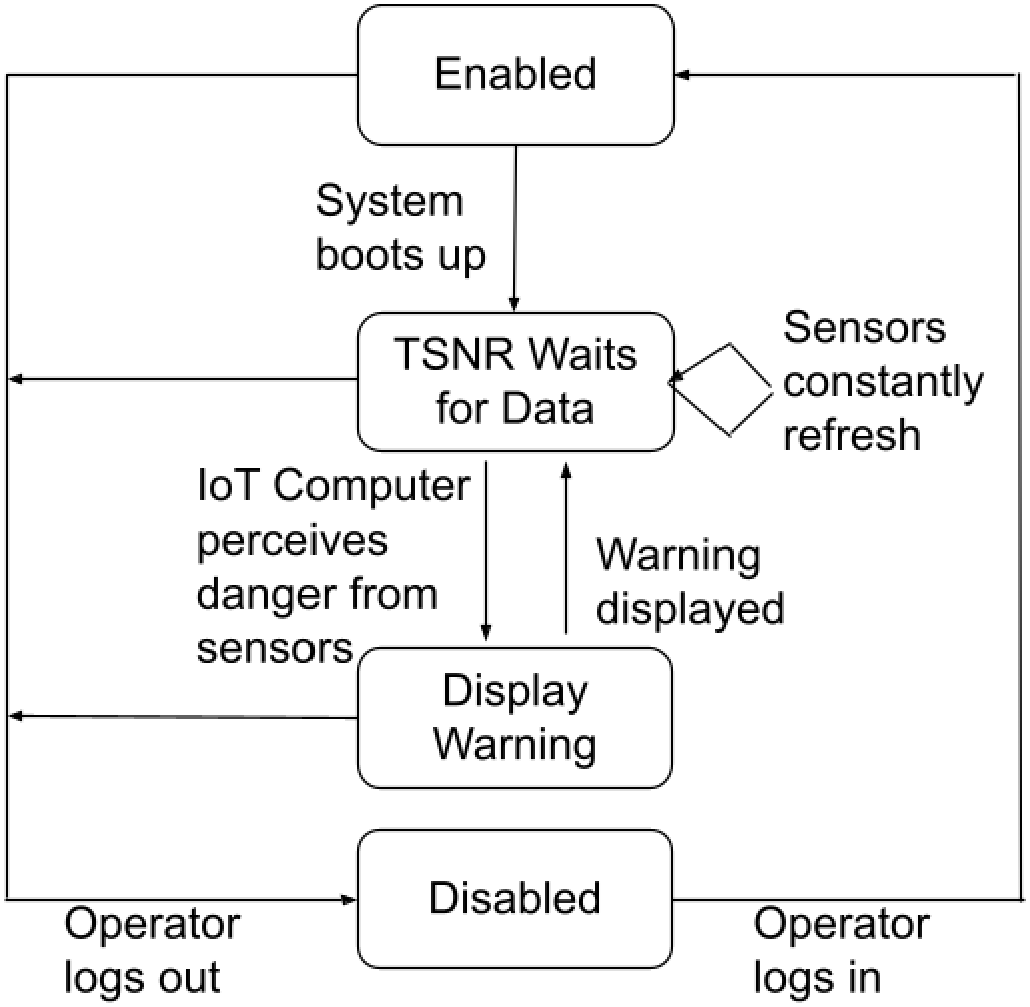
Use Case 4.6.6:


Driver
IoT Computer
Sensors
Display
Driver turns on IoT Computer
Gets pressure data from sensors
Send data back to computer
Displays warning and suggested action
Yes
is pressure on roof within normal bounds?
No

Use Case 4.6.7:


Driver
IoT Computer
Sensors
Display
Driver turns on IoT Computer
Checks Camera Data
Detects railroad crossing
Suggest blow horn
Displays warning and suggested action

4.7 State Diagrams

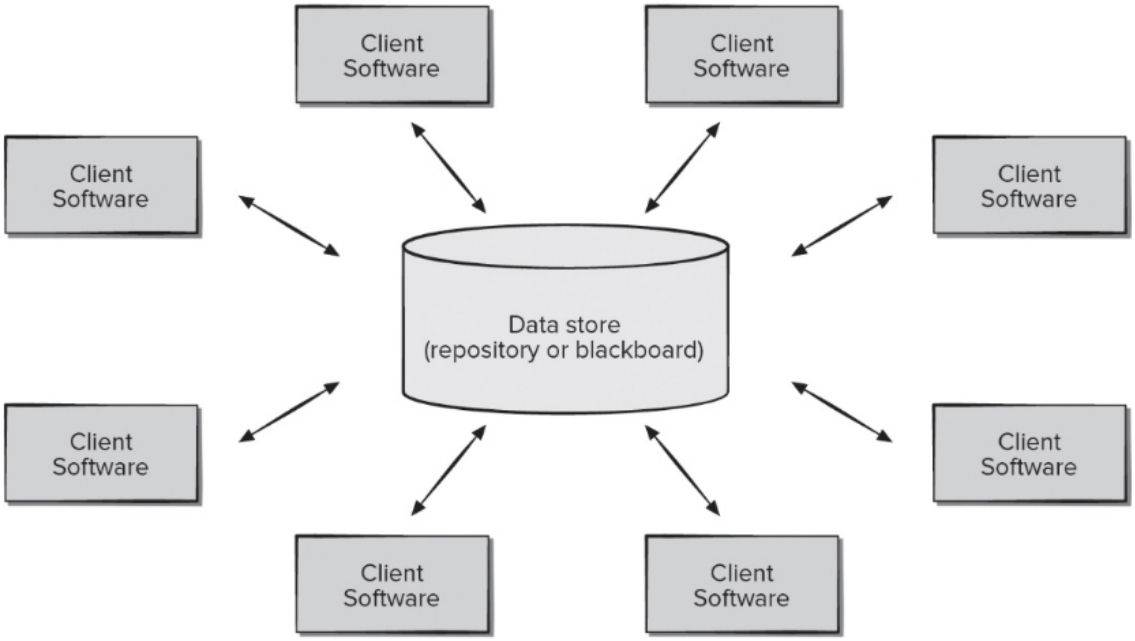


5. Software Architecture Analysis

In this section of the document, several different potential software architectures will be analyzed by a list of pros and cons. We will consider how each architecture fits our requirements modelling for the IoT software and ultimately choose the architecture which best fits our complete system.

5.1 Data Centered Architecture

With Data Centered Architecture, there is a central database and code is separated into client softwares which all interact with the central database to provide the different functionalities of the software ecosystem. An example of this architecture in use is the Stevens web services.

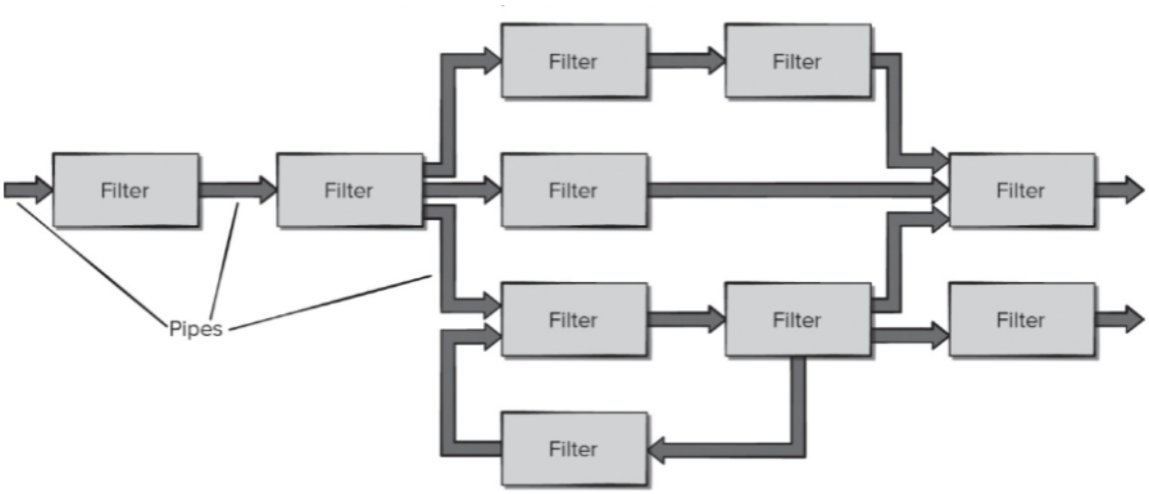


Pros	Cons
- Componentized software can be easier to test	- A database is not very useful when the data stored is small, like sensor data

We decided not to use a central database in our software architecture because its use would be limited. We want our program to constantly check sensor data which does not benefit from the advantages that databases give, like the ability to store large amounts of data.

5.2 Data Flow Architecture

Data Flow Architecture follows a non-deterministic execution pattern to transform data through the input and output of individual modules. Unlike using a traditional program counter, Data Flow Architecture individual modules wait for input to be received from other modules before execution.

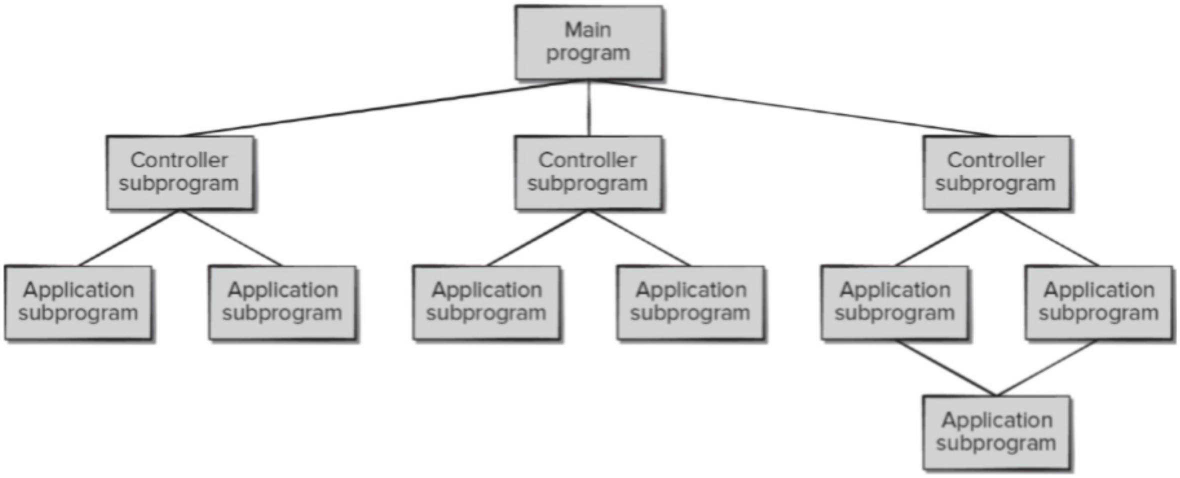


Pros	Cons
- Code can run in different order at runtime based on the flow of data - Good for database engines/parallel computing/network routing	- Filters cannot work cooperatively in pipe and filter architecture - Increased latency

Considering the above factors, we decided not to use Data Flow Architecture because of the high latency associated with pipes, along with the need for our subsystems to cooperate with one another. We also feel that this architecture is not suited to our group because of our relative inexperience with pipe systems as compared to other systems such as object oriented programming.

5.3 Call Return Architecture

The “Main Program” style of Call and Return architecture divides the program into a main process which a number of subprograms report to. Controller subprograms oversee further subprograms in a tree style.

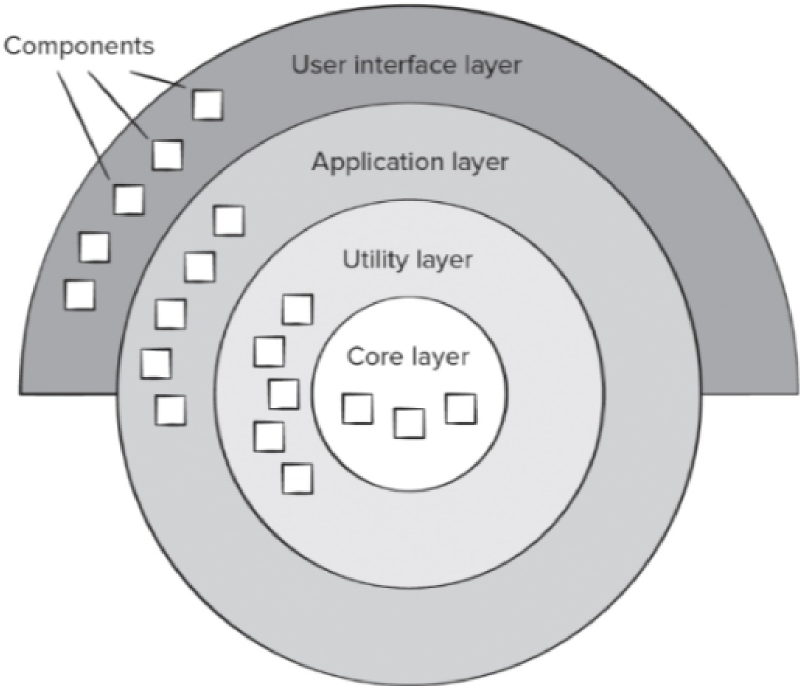


Pros	Cons
- Many subprograms provides clarity as to where the system may fail - Subprograms can be easily modified without affecting other subprograms	- Having multiple subprograms necessitates increased maintenance and testing costs

Because of the pros and cons, we decided not to use Call Return Architecture because even though a Call Return Structure would be well organized and easy to maintain/change, we feel an object oriented architecture can accomplish the same task without the increased testing. Also, an error in a Controller Subprogram would make the program lose functionality in all the Application Subprograms in that controller, since the data feeds to the Controller.

5.4 Layered Architecture

Layered architecture, common in personal computing divides the system into layers of software which do similar things. Outer layers depend on the inner layers to function, allowing software development to be done at many levels of abstraction of the system.

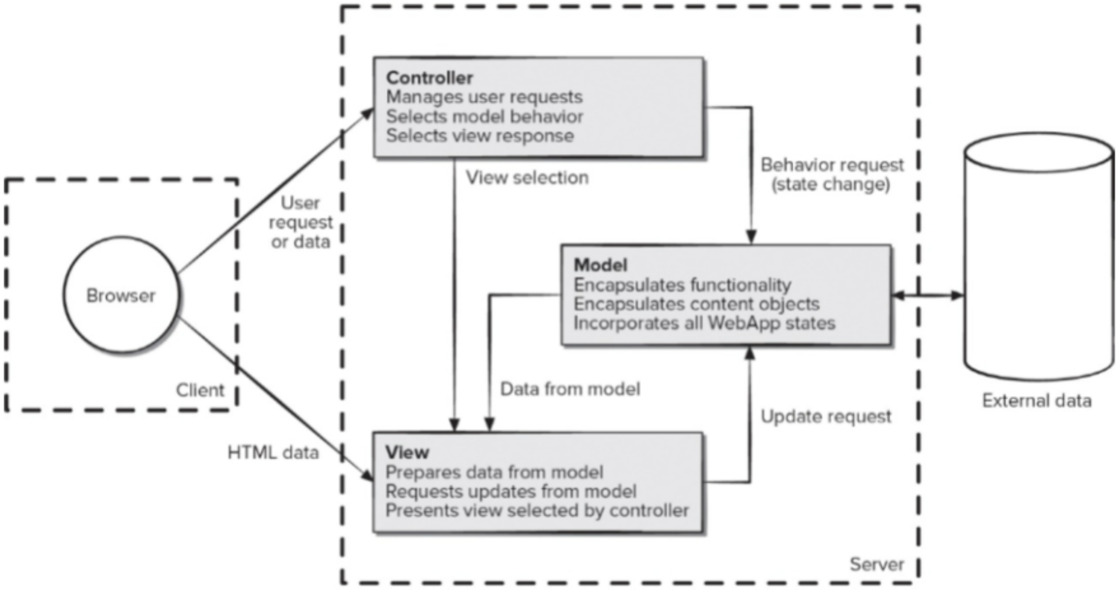


Pros	Cons
- Each layer focuses on a certain scope of operations, problems are more straightforward - Each layer can be tested individually to find errors easier - Changes in one layer will not affect lower layers	- Performance gets slower and slower with each added layer - Better suited for a large system or ecosystem

We decided not to use Layered Centered Architecture because we have more substantial understanding of other architectures and this architecture is more fit for advanced systems. Our system should be lightweight and easily understandable, while this architecture is suitable for systems where the components are very dependent on inner layers, while our system is straightforward.

5.5 Model View Controller Architecture

Used for user interfaces, the Model View Controller model of software architecture. The Model portion of the design is the central component of the software and comparable to the “main program” of the Call Return Architecture. View refers to the visual representations generated by the software. Controller is used to process user input.

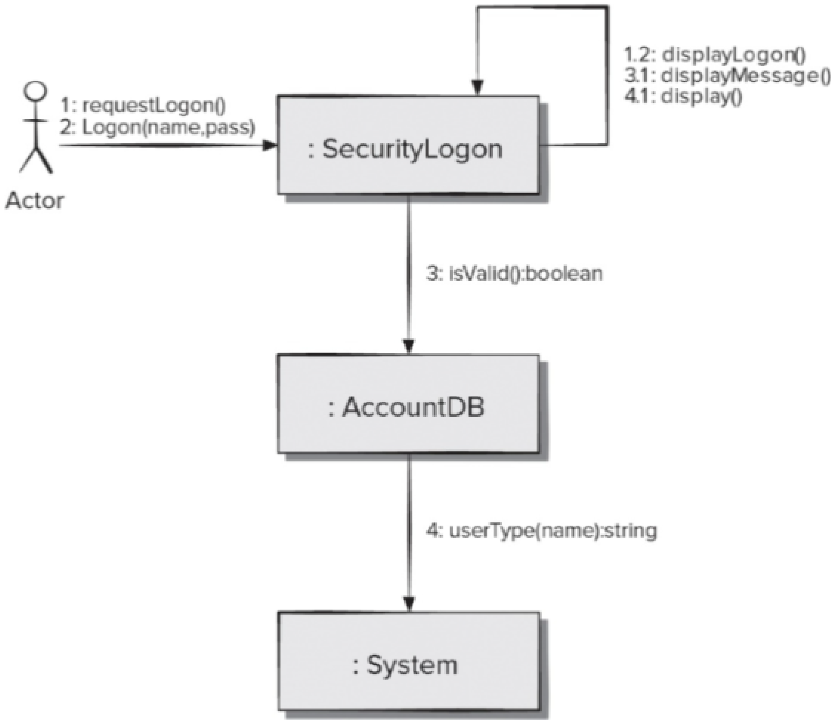


Pros	Cons
- Development is quick - Easy to update - Easy to debug	- Our user interface does not require this complexity - Browser is not the best choice for IoT Display because it can be simpler

The Model View Controller Architecture does not fit our project because, while development using this architecture would be swift, its complexity would increase the total development time by a greater margin than we would gain. Importantly, this architecture is used for user-interface systems while our system will take no input from the operator. In addition, our system is not connected to the internet or external data, and the model component would be where most of our architecture truly lies.

5.6 Object-oriented Architecture

Object-oriented Architecture breaks up the code into reusable and self-sufficient objects which interact with each other and transfer data. Object-oriented Architecture takes advantage of inheritance, class, polymorphism and other design patterns.



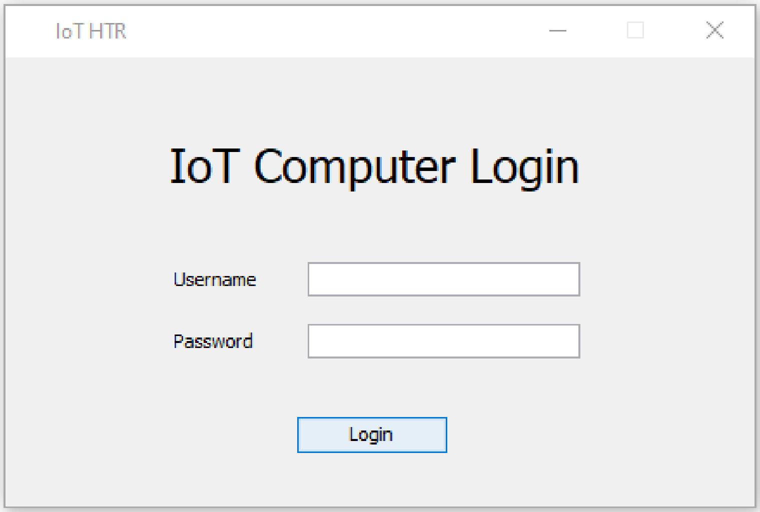
Pros	Cons
- Simple Implementation - Easily organized since each component has its own functions - Can mirror our existing class modelling	- Small components need classes for simple functionality

Ultimately, we decided to use Object-oriented Architecture because of its understandability, code reusability, and our familiarity with object oriented programming. Also, our requirements modelling has already defined objects to be used in code. Due to our familiarity to this architecture, our efficiency in being able to produce a satisfactory program would allow us to make additional adjustments in order to provide a more refined product. Following this architecture, we will implement the objects and methods shown in section 4.3 and 4.4.

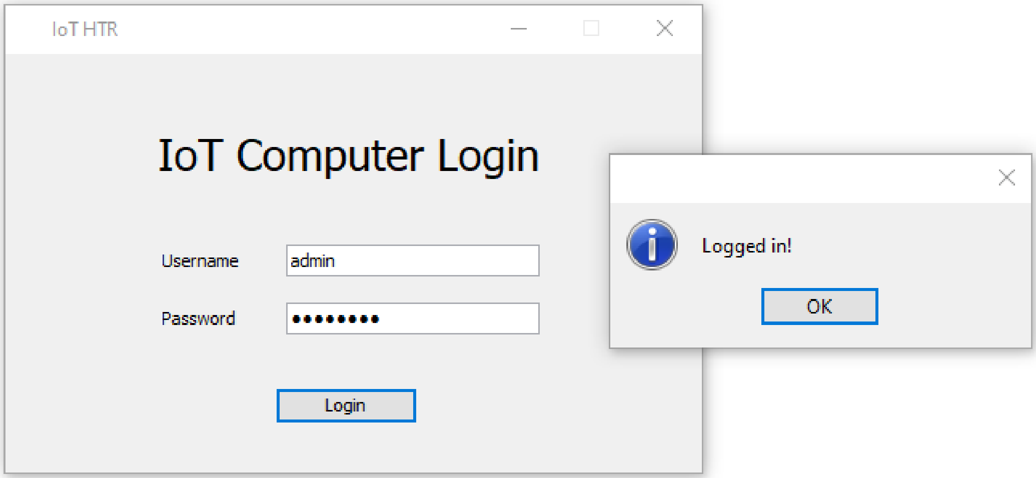
5.6.1 IoT Display

The IoT Display will not take any input from the operator. The display serves only to communicate warnings and suggestions generated by the IoT Computer. When there is a hazard detected, a large recommendation message is displayed, along with a sound being played and the corresponding warning icon lights up.
Since the project requires a GUI, we chose to implement Java’s Swing GUI toolkit because of its ease of use.

Before logged in:



When logged in:

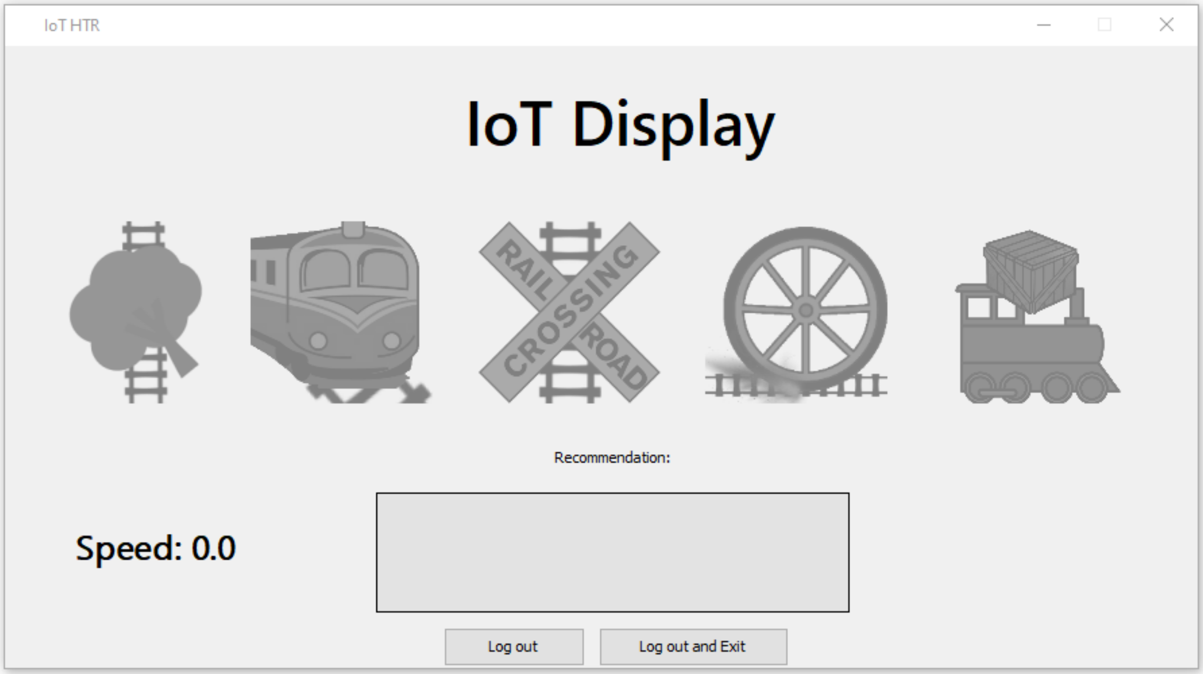


Username admin

Password ••••••••

Logged in!

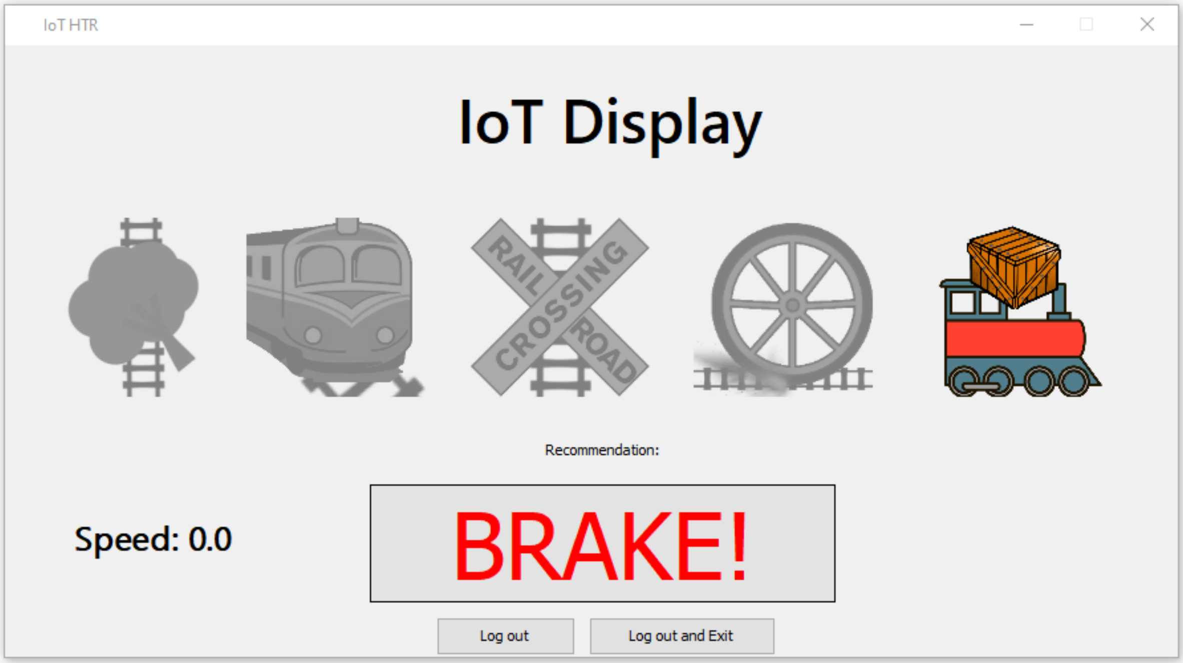
When there are no hazards detected:



Recommendation:

Speed: 0.0

When hazards are detected (obstacle on top of train, for example):



Recommendation:

BRAKE!

Speed: 0.0

6. Code

iot.IoTComputer

```
package iot;

import java.awt.Color;
import java.awt.Toolkit;

public class IoTComputer extends Sensors {

        static boolean loggedIn = false;

        // These variables are used for timing the "Horn!" warnings for railroad
        // crossings.
        static int timer = 16 * Sensors.time / 100;
        static int timer2 = 6 * Sensors.time / 100;

        static boolean login(String usr, String pwd) {
                return loggedIn = usr.equals("admin") && pwd.equals("password");
        }

        static void setSpeed() {
                Display.txtArea.setText("Speed: " +
Double.toString(Sensors.detectSpeedGPS()));
        }

        static void getSuggestion() {
                if (Sensors.interrupted)
                        return;

                setSpeed();

                Display.txtArea.append(null);
                String suggestion = "";

                Double obstacle_distance = Sensors.detectObstacleFront();
                double speed_from_rpm = Sensors.detectRPM() / (10 * (24 * 3.1415 * 60 /
6336));

                if (Sensors.detectSpeedGPS() - speed_from_rpm > .5) {
                        Display.lblWheelSlip.setEnabled(true);
                        suggestion = "Slow!";
                } else {
                        Display.lblWheelSlip.setEnabled(false);
                }
                boolean crossing = Sensors.detectRailroadCrossing();
                if (obstacle_distance > 0 && !crossing) {
                        // Check if the obstacle is moving
                        double obstaclespeed = Sensors.detectObstacleSpeed();
```

```
    if (obstaclespeed == 0) {
        Display.lblObstacle.setEnabled(true);
        Display.lblMovingObstacle.setEnabled(false);
        // Obstacle on track!
        suggestion = "Slow!";
        if (obstacle_distance < 500) {
            suggestion = "BRAKE!";
        }
    } else if (obstaclespeed > Sensors.detectSpeedGPS()) {
        // The obstacle is moving safely away from the train
    } else {
        Display.lblObstacle.setEnabled(false);
        Display.lblMovingObstacle.setEnabled(true);
        // Slow Train ahead!
        if (obstacle_distance > 800) {
            suggestion = "Slow!";
        } else {
            suggestion = "BRAKE!";
        }
    }
} else {
    Display.lblObstacle.setEnabled(false);
    Display.lblMovingObstacle.setEnabled(false);
}

if (crossing) {
    // Railroad crossing!
    Display.lblRailroadCrossing.setEnabled(true);
    if (timer >= 0) {
        timer--;
        suggestion = "Horn!";
    }
    if (obstacle_distance < 1000) {
        if(timer2 >=0) {
            timer2--;
            suggestion = "Horn!";
        }
    }
} else {
    timer = 15 * Sensors.time / 100;
    timer2 = 5 * Sensors.time / 100;
    Display.lblRailroadCrossing.setEnabled(false);
}

if (Sensors.detectObjectTop()) {
    // Object on top of train!
    suggestion = "BRAKE!";
    Display.lblObjectTop.setEnabled(true);
} else {
    Display.lblObjectTop.setEnabled(false);
```

```
        }

        if (!Display.lblRecommendation.getText().toString().equals(suggestion)
                && !suggestion.equals("")) {
            Toolkit.getDefaultToolkit().beep();
        }

        if (suggestion.equals("BRAKE!")) {
            Display.lblRecommendation.setForeground(Color.RED);
        } else {
            Display.lblRecommendation.setForeground(Color.BLACK);
        }

        Display.lblRecommendation.setText(suggestion);
    }
}
```

iot.Sensors

```
package iot;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;

public class Sensors {

    static double speedGPS = 0.0;
    static boolean objectTop = false;
    static double obstacleFront = 0.0;
    static double obstacleSpeed = 0.0;
    static boolean railroadCrossing = false;
    static double rpm = 0.0;

    //This is used for inter-thread communication when the SwingWorker is interrupted
    static boolean interrupted = false;

    //This variable controls the speed of the simulation
    static final int time = 300;

    public static void tsnrRead() throws Exception {
        File file = new File("./readings.txt");
    BufferedReader br = new BufferedReader(new FileReader(file));
    String str;
    while (((str = br.readLine()) != null) && Sensors.interrupted == false) {
        //* All sensor values have been read, so the
        //* TSNR updates values synchronously.
        String[] readings = str.split("[,]", 0);
        speedGPS = Double.parseDouble(readings[0]);
        objectTop = Boolean.parseBoolean(readings[1]);
        obstacleFront = Double.parseDouble(readings[2]);
        obstacleSpeed = Double.parseDouble(readings[3]);
        railroadCrossing = Boolean.parseBoolean(readings[4]);
        rpm = Double.parseDouble(readings[5]);

        //* TSNR passes the job onto IoT Computer to get suggestions
        //* for the conductor.
        IoTComputer.getSuggestion();
        Thread.sleep(time);

    }
    Display.lblRecommendation.setText("End!");
    br.close();
    }

    static double detectSpeedGPS() {
        return speedGPS;
    }

    static boolean detectObjectTop() {
        return objectTop;
    }

    static double detectObstacleFront() {
        return obstacleFront;
    }

    static double detectObstacleSpeed() {
        return obstacleSpeed;
    }

    static boolean detectRailroadCrossing() {
        return railroadCrossing;
    }

    static double detectRPM() {
        return rpm;
    }
}
```

iot.Display

```
package iot;

import java.awt.Color;
import java.awt.EventQueue;

import javax.swing.*;
import javax.swing.border.Border;
import java.awt.Font;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.awt.SystemColor;
import java.awt.Toolkit;

public class Display {

    private JFrame frmIotDisplay;

    public static JLabel lblRecommendation;
    public static JTextArea txtArea;
    public static JLabel lblObstacle;
    public static JLabel lblMovingObstacle;
    public static JLabel lblRailroadCrossing;
    public static JLabel lblWheelSlip;
    public static JLabel lblObjectTop;

    /**
     * Launch the application.
     */
    public static void main2(String[] args) {
        Sensors.interrupted=false;
        try {

UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        } catch (ClassNotFoundException | InstantiationException |
IllegalAccessException
                | UnsupportedLookAndFeelException e1) {
            //
            e1.printStackTrace();
        }
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    Display window = new Display();

window.frmIotDisplay.setIconImage(Toolkit.getDefaultToolkit().getImage(getClass().getResour
ce("/files/icon.png")));
                    window.frmIotDisplay.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }

    /**
     * Create the application.
     */
    public Display() {
        initialize();
    }

    /**
     * Initialize the contents of the frame.
     */
    private void initialize() {
        frmIotDisplay = new JFrame();
        frmIotDisplay.setTitle("IoT HTR\r\n");
        frmIotDisplay.setResizable(false);
        frmIotDisplay.setBounds(100, 100, 900, 500);
        frmIotDisplay.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frmIotDisplay.getContentPane().setLayout(null);
        frmIotDisplay.setLocationRelativeTo(null);

        lblRecommendation = new JLabel("");
        lblRecommendation.setForeground(SystemColor.textText);
        lblRecommendation.setBackground(SystemColor.controlHighlight);
        lblRecommendation.setHorizontalAlignment(SwingConstants.CENTER);
        lblRecommendation.setFont(new Font("Tahoma", Font.PLAIN, 70));
        lblRecommendation.setBounds(275, 331, 351, 89);
        frmIotDisplay.getContentPane().add(lblRecommendation);
        lblRecommendation.setOpaque(true);
        Border blackline = BorderFactory.createLineBorder(Color.black);
        lblRecommendation.setBorder(blackline);

        JLabel lblTitle = new JLabel("IoT Display");
        lblTitle.setFont(new Font("Segoe UI Semibold", Font.PLAIN, 46));
        lblTitle.setHorizontalAlignment(SwingConstants.CENTER);
        lblTitle.setBounds(268, 11, 376, 89);
        frmIotDisplay.getContentPane().add(lblTitle);

        JButton btnLogout = new JButton("Log out");
        btnLogout.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                Sensors.interrupted = true;
                //This boolean stops the SwingWorker when the window closes.
                Login.main(null);
                frmIotDisplay.setVisible(false);
                frmIotDisplay.dispose();
            }
        });
        btnLogout.setBounds(325, 431, 105, 29);
        frmIotDisplay.getContentPane().add(btnLogout);

        JButton btnLogoutExit = new JButton("Log out and Exit");
        btnLogoutExit.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
```

```
                        Sensors.interrupted = true;
                        frmIotDisplay.setVisible(false);
                        frmIotDisplay.dispose();
                    }
                });
                btnLogoutExit.setBounds(440, 431, 141, 29);
                frmIotDisplay.getContentPane().add(btnLogoutExit);

                JLabel lblSubtitle = new JLabel("Recommendation:");
                lblSubtitle.setHorizontalAlignment(SwingConstants.CENTER);
                lblSubtitle.setBounds(391, 291, 121, 29);
                frmIotDisplay.getContentPane().add(lblSubtitle);

                txtArea = new JTextArea();
                txtArea.setBackground(SystemColor.control);
                txtArea.setBounds(50, 352, 163, 38);
                frmIotDisplay.getContentPane().add(txtArea);
                txtArea.setText("Speed: 0.0");
                txtArea.setFont(new Font("Segoe UI Symbol", Font.BOLD, 25));
                txtArea.setEditable(false);
                txtArea.setDisabledTextColor(Color.BLACK);
                txtArea.setEnabled(false);

                lblObstacle = new JLabel("");
                lblObstacle.setEnabled(false);
                lblObstacle.setIcon(new
ImageIcon(Display.class.getResource("/files/obstacle1.png")));
                lblObstacle.setHorizontalAlignment(SwingConstants.CENTER);
                lblObstacle.setBounds(35, 130, 135, 135);
                frmIotDisplay.getContentPane().add(lblObstacle);

                lblMovingObstacle = new JLabel("");
                lblMovingObstacle.setIcon(new
ImageIcon(Display.class.getResource("/files/moving_obstacle.png")));
                lblMovingObstacle.setEnabled(false);
                lblMovingObstacle.setHorizontalAlignment(SwingConstants.CENTER);
                lblMovingObstacle.setBounds(182, 130, 135, 135);
                frmIotDisplay.getContentPane().add(lblMovingObstacle);

                lblRailroadCrossing = new JLabel("");
                lblRailroadCrossing.setEnabled(false);
                lblRailroadCrossing.setIcon(new
ImageIcon(Display.class.getResource("/files/RailroadCrossing.png")));
                lblRailroadCrossing.setHorizontalAlignment(SwingConstants.CENTER);
                lblRailroadCrossing.setBounds(351, 130, 135, 135);
                frmIotDisplay.getContentPane().add(lblRailroadCrossing);

                lblWheelSlip = new JLabel("");
                lblWheelSlip.setEnabled(false);
                lblWheelSlip.setIcon(new
ImageIcon(Display.class.getResource("/files/WheelSlip.png")));
        lblWheelSlip.setHorizontalAlignment(SwingConstants.CENTER);
        lblWheelSlip.setBounds(519, 130, 135, 135);
        frmIotDisplay.getContentPane().add(lblWheelSlip);

        lblObjectTop = new JLabel("");
        lblObjectTop.setEnabled(false);
        lblObjectTop.setIcon(new
ImageIcon(Display.class.getResource("/files/objectTop.png")));
        lblObjectTop.setHorizontalAlignment(SwingConstants.CENTER);
        lblObjectTop.setBounds(696, 130, 135, 135);
        frmIotDisplay.getContentPane().add(lblObjectTop);


        SwingWorker<Void, String> worker = new SwingWorker<Void, String>() {
    @Override
    protected Void doInBackground() throws Exception {
        Sensors.tsnrRead();
     return null;
    }
   };
   worker.execute();

    }
}
```

iot.Login

```
package iot;

import java.awt.EventQueue;

import javax.swing.*;
import java.awt.Font;
import java.awt.Toolkit;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

public class Login {

    private JFrame frmIotHtr;
    private JTextField username;
    private JTextField password;

    /**
     * Launch the application.
     */
    public static void main(String[] args) {
        try {
```

```
UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        } catch (ClassNotFoundException | InstantiationException |
IllegalAccessException
                | UnsupportedLookAndFeelException e1) {
            //
            e1.printStackTrace();
        }
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    Login window = new Login();


window.frmIotHtr.setIconImage(Toolkit.getDefaultToolkit().getImage(getClass().getResource("/
files/icon.png")));
                    window.frmIotHtr.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }

    /**
     * Create the application.
     */
    public Login() {
        initialize();
    }

    /**
     * Initialize the contents of the frame.
     */
    private void initialize() {
        frmIotHtr = new JFrame();
        frmIotHtr.setTitle("IoT HTR");
        frmIotHtr.setBounds(100, 100, 450, 300);
        frmIotHtr.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frmIotHtr.getContentPane().setLayout(null);
        frmIotHtr.setResizable(false);
        frmIotHtr.setLocationRelativeTo(null);

        JLabel lblUsername = new JLabel("Username");
        lblUsername.setBounds(98, 117, 67, 25);
        frmIotHtr.getContentPane().add(lblUsername);

        JLabel lblPassword = new JLabel("Password");
        lblPassword.setBounds(98, 153, 67, 25);
        frmIotHtr.getContentPane().add(lblPassword);
        username = new JTextField();
        username.setBounds(175, 119, 158, 20);
        frmIotHtr.getContentPane().add(username);
        username.setColumns(10);

        password = new JPasswordField();
        password.setBounds(175, 155, 158, 20);
        frmIotHtr.getContentPane().add(password);
        password.setColumns(10);

        JLabel lblTitle = new JLabel("IoT Computer Login");
        lblTitle.setFont(new Font("Tahoma", Font.PLAIN, 27));
        lblTitle.setHorizontalAlignment(SwingConstants.CENTER);
        lblTitle.setBounds(86, 30, 257, 65);
        frmIotHtr.getContentPane().add(lblTitle);

        JButton btnLogin = new JButton("Login");
        btnLogin.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {

                String usr = username.getText();
                String pwd = password.getText();

                if(IoTComputer.login(usr, pwd)) {
                    JOptionPane.showMessageDialog(null, "Logged in!", "",
JOptionPane.INFORMATION_MESSAGE);
                    Display.main2(null);
                    frmIotHtr.setVisible(false);
                    frmIotHtr.dispose();
                } else {
                    JOptionPane.showMessageDialog(frmIotHtr, "Incorrect
login details");
                }
            }
        });
        btnLogin.setBounds(168, 208, 89, 23);
        frmIotHtr.getContentPane().add(btnLogin);
        frmIotHtr.getRootPane().setDefaultButton(btnLogin);
    }
}
```

7. Testing

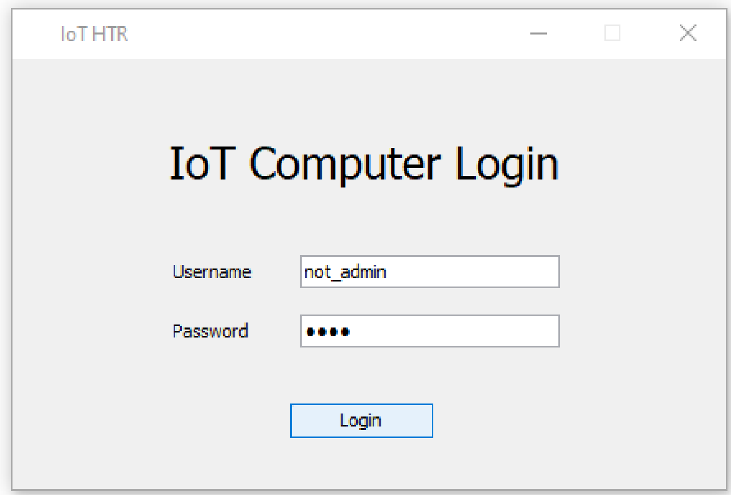
7.1 Operator logs in

Test Description: The operator starts the program, and types in a username and password.

Methodology: The operator tries many incorrect details before inputting “admin” and “password”, for which the system unlocks and operation begins.

Expected Outcome: A pop-up will display “Incorrect login details” if incorrect details are entered, and will display “Logged in!” if admin/password is inputted.

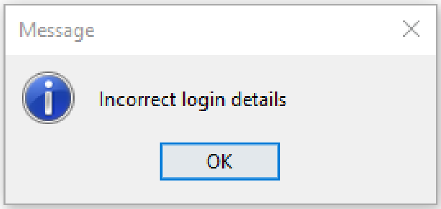
Output:



IoT Computer Login

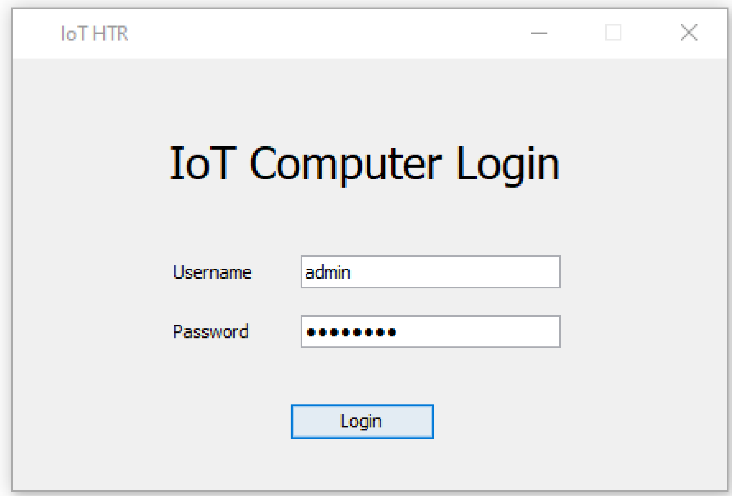
Username: not_admin

Password: ••••



Incorrect login details

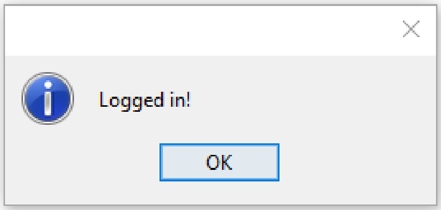
Without admin and password, the system is unavailable. With correct details, one may log in.



IoT Computer Login

Username: admin

Password: ••••••••



Logged in!

7.2 Obstacle on the track

Test Description: The train approaches a non-moving object on the track.

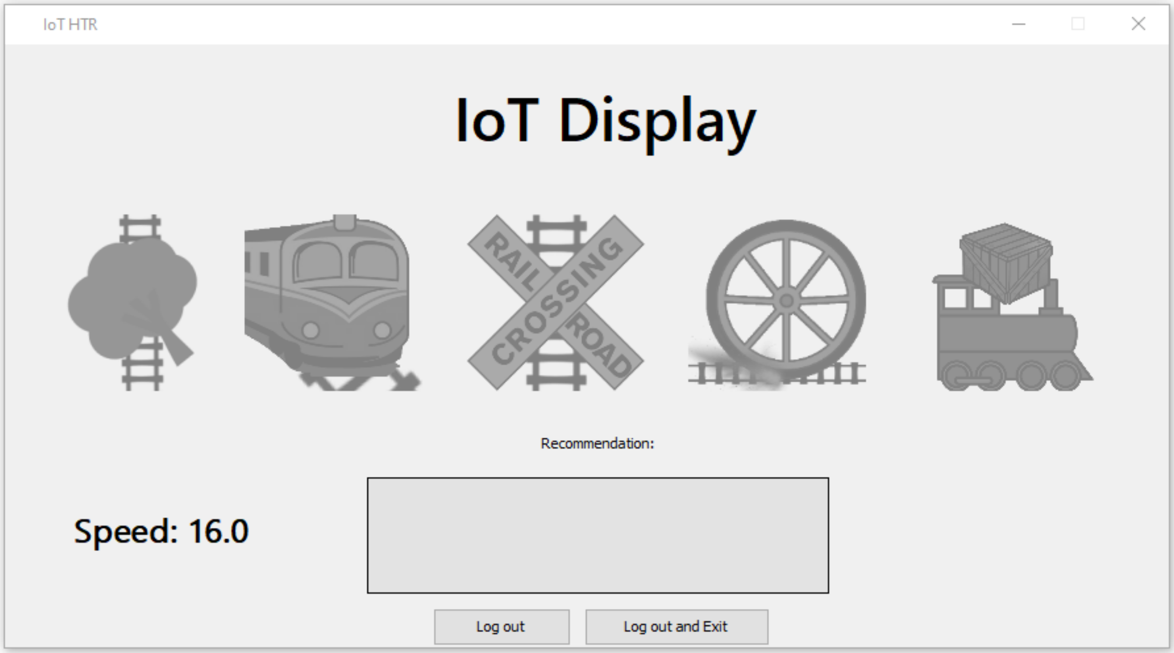
Methodology:

1. Sensor readings are generated for a train ride where an object is detected at a range of 1,000 ft.
2. As the train approaches the object, the distance sensor measurement decreases, simulating the approach.

Expected Outcome: After logging in, the train will speed up and detect no hazards. Once the object sensor reports a non-zero distance, the object warning icon is displayed, and the recommendation is set. For more than 500ft distance from the object, the recommendation is “Slow down”, otherwise “BRAKE!” is recommended.

Output:

User logs in successfully.

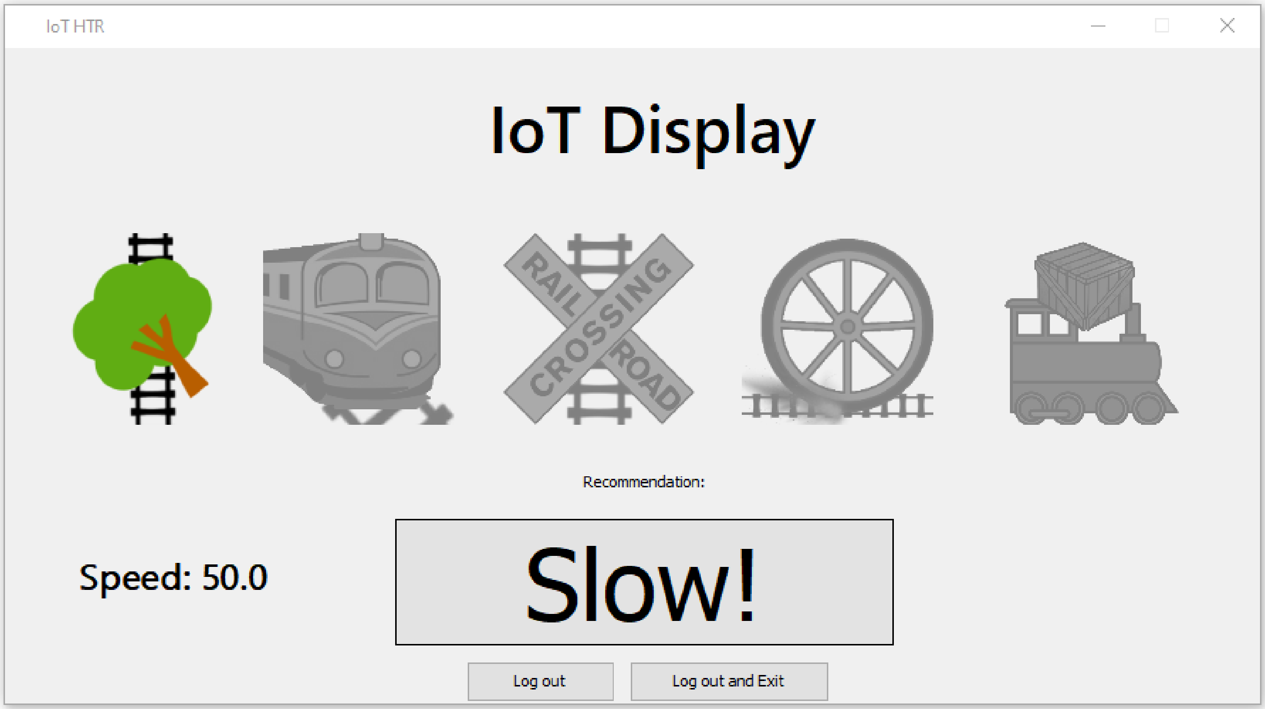

IoT HTR

IoT Display

Recommendation:

Speed: 16.0

The sensor log file is read automatically as the train speeds up with no warnings displayed.



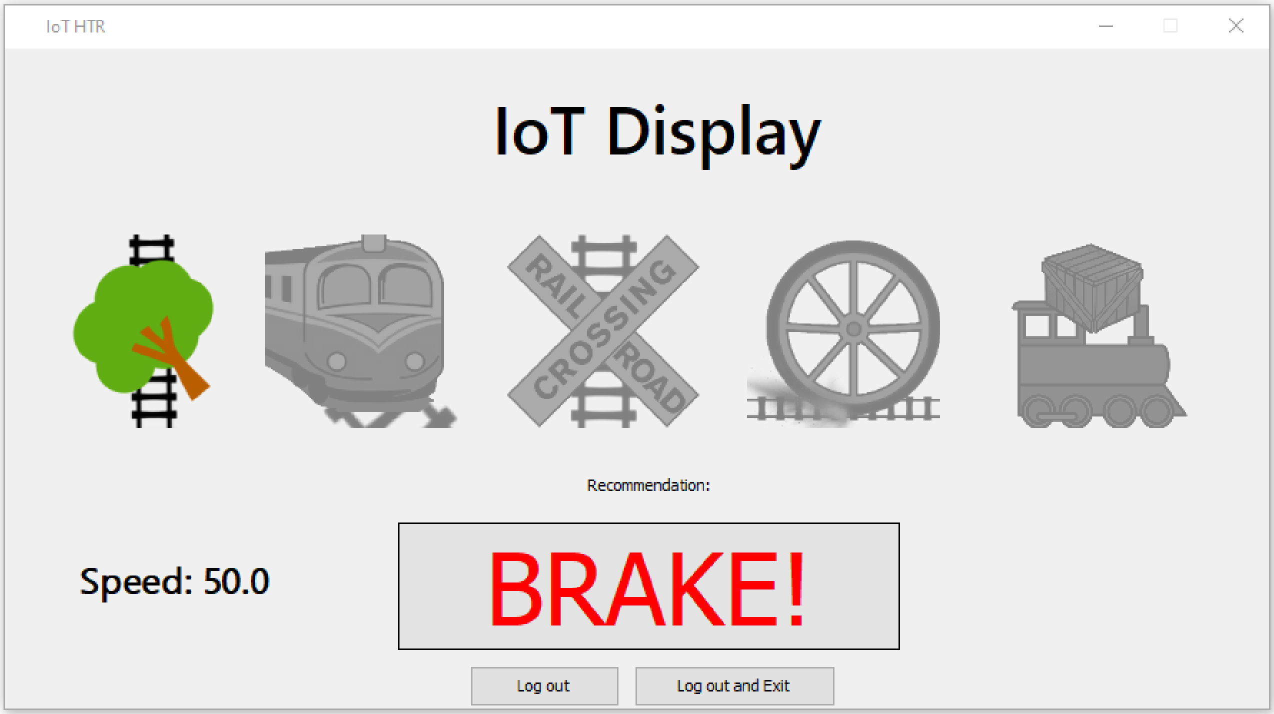
IoT Display

Recommendation:

Speed: 50.0

Slow!

The obstacle sensor detects a non-moving obstacle starting at a range of 1,000 ft. A sound is played and the recommendation is set to “Slow!”



IoT Display

Recommendation:

Speed: 50.0

BRAKE!

The obstacle enters a range of 500 ft, so a sound is played and the warning is set to “BRAKE!”

7.3.1 Slow-moving obstacle on the track

Test Description: The train, at 40mph, approaches a slow moving (10mph) obstacle on the track.

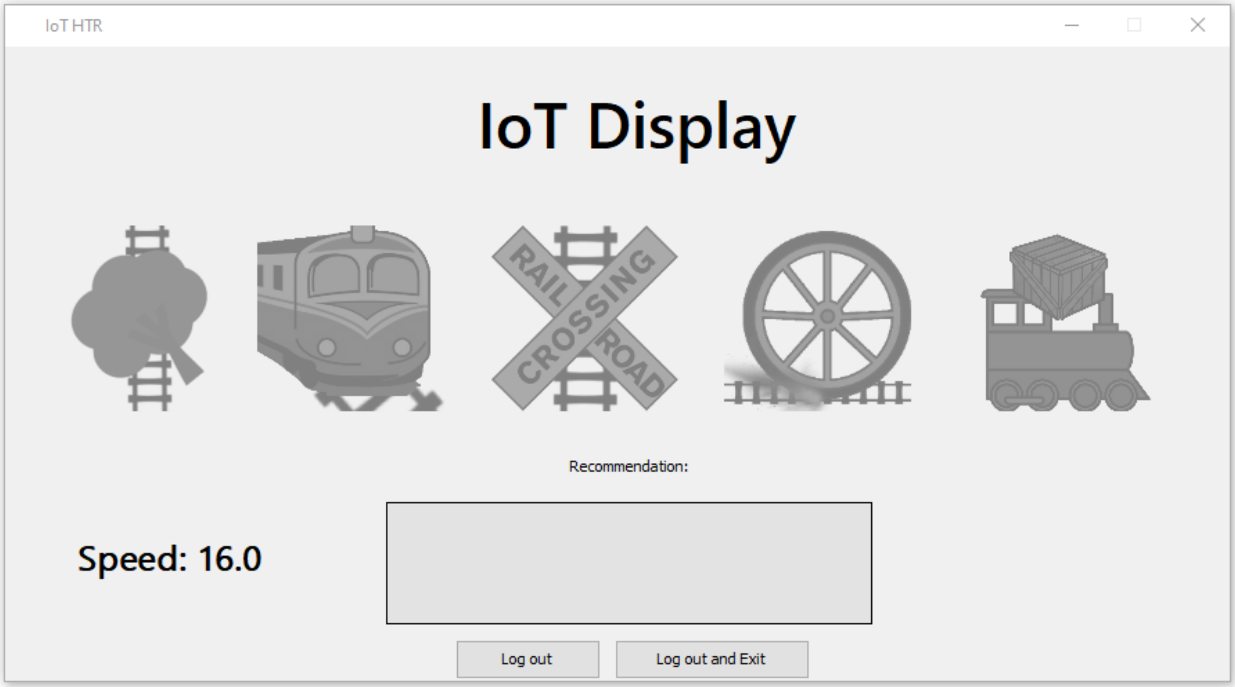
Methodology:

1. Sensor readings are generated for a train ride where a moving object (10mph) is detected at a range of 1,000 ft.
2. Since the train will be going faster, the object distance decreases, simulating an approach.

Expected Outcome: The moving obstacle will be detected at a range of 1,000ft. The object’s speed, measured by the object speed sensor will be compared to the train’s GPS speed. Since the obstacle is moving slower than the train, a suggestion will be generated. For more than 500ft distance from the obstacle, the recommendation is “Slow down”, otherwise “BRAKE!” is recommended.

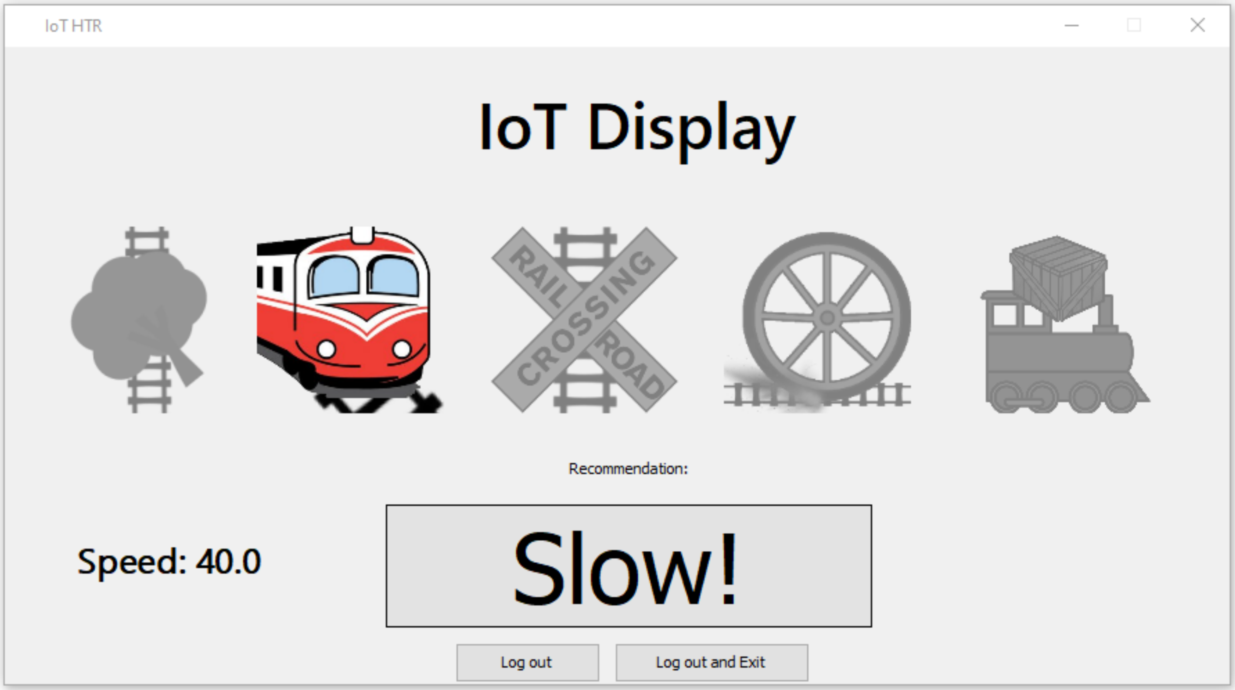
Output:

User logs in successfully.



Speed: 16.0

The sensor log file is read automatically as the train speeds up with no warnings displayed.



IoT HTR

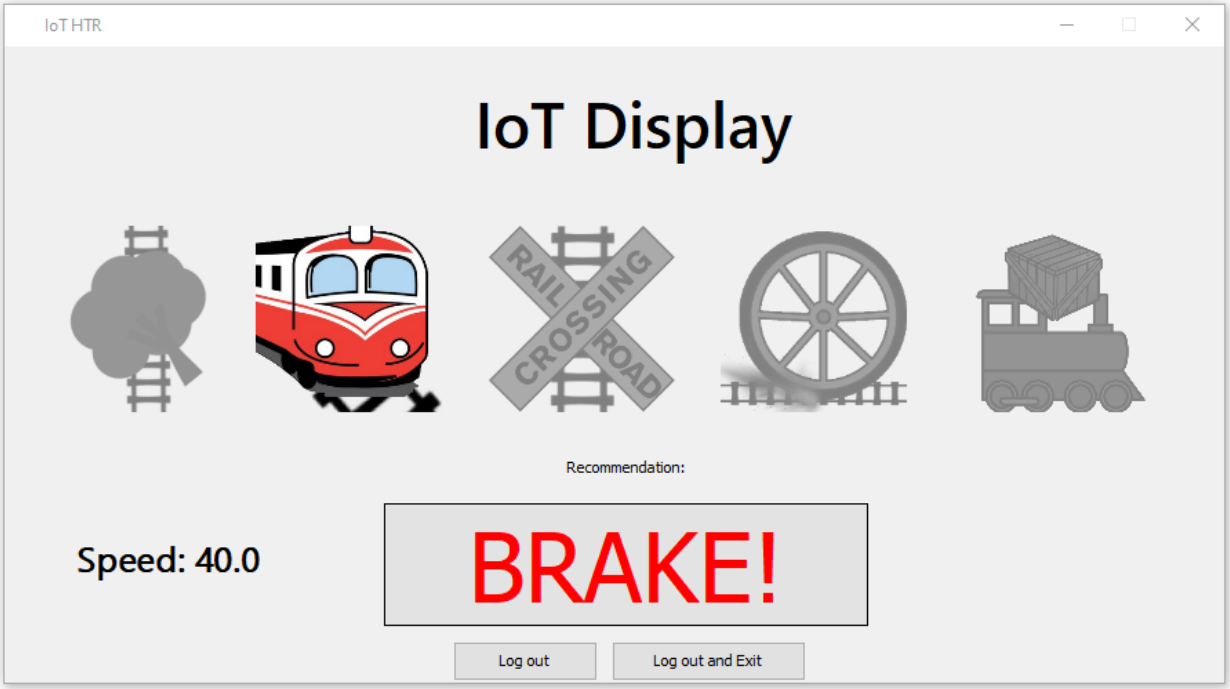
IoT Display

Recommendation:

Speed: 40.0

Slow!

The obstacle sensor detects a moving obstacle starting at a range of 1,000 ft. The speed of the object is found to be slower than the speed of the GPS. A sound is played and the recommendation is set to “Slow!”



IoT HTR

IoT Display

Recommendation:

Speed: 40.0

BRAKE!

The obstacle enters a range of 500 ft, so a sound is played and the warning is set to “BRAKE!”

7.3.2 Fast-moving obstacle on the track

Test Description: The train, at 10mph, approaches a fast moving (50mph) obstacle on the track.

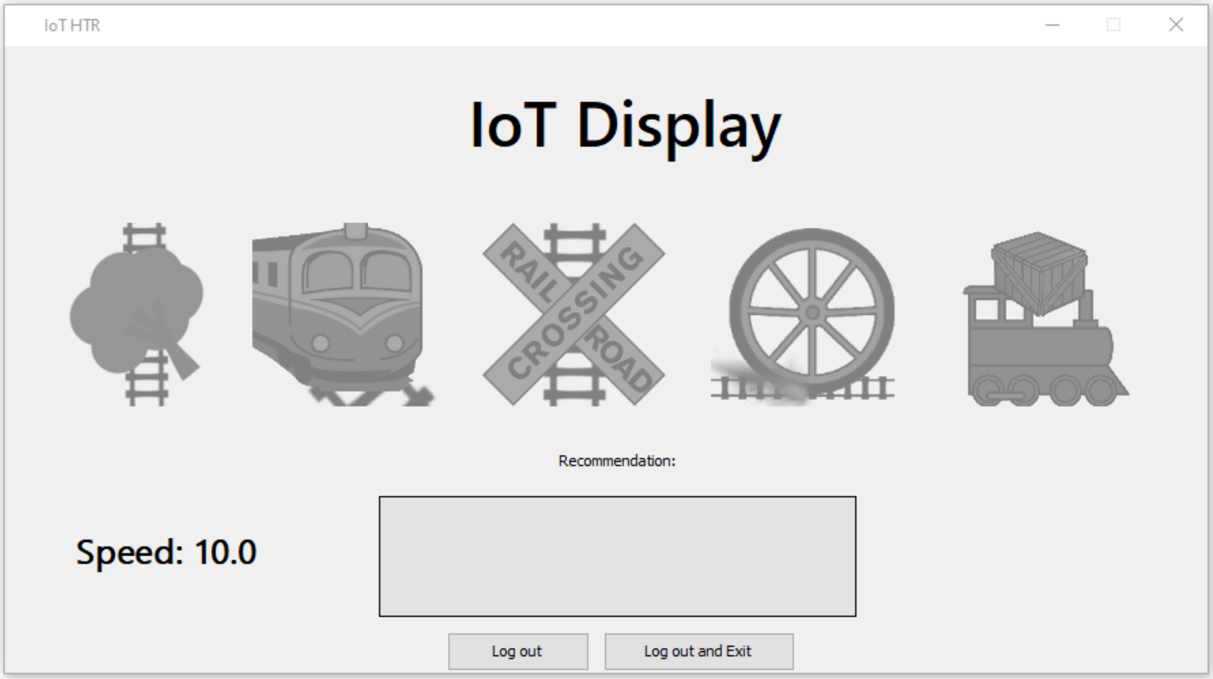
Methodology:

1. Sensor readings are generated for a train ride where a moving object (50mph) is detected at a range of 200 ft.
2. Since the train will be going slower, the object distance increases, simulating the train exiting the 1,000ft range of our sensors.

Expected Outcome: The moving obstacle will be detected at a range of 200 ft. The object's speed, measured by the object speed sensor will be compared to the train's GPS speed. Since the obstacle is moving faster than the train, the obstacle is not a hazard, and no warnings will be generated.

Output:

The user successfully logs in.



Speed: 10.0

No warnings are displayed because the object is safely moving away at a greater speed (detected by the speed sensor) than the train's GPS speed.

7.4 Railroad crossing detection

Test Description: The train, at 100mph, approaches a railroad crossing from a distance of 1 mile.

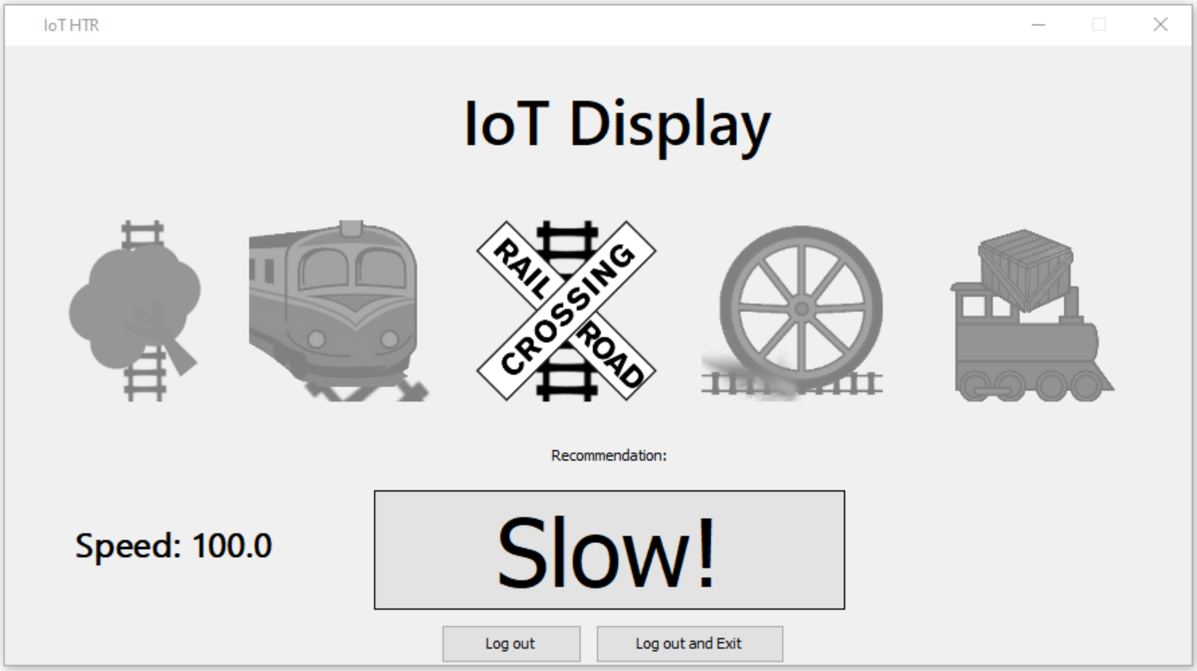
Methodology:

1. Sensor readings are generated for a train ride where a railroad crossing is detected at a range of 1 mile.
2. The distance will decrease to 0 as the train passes through.

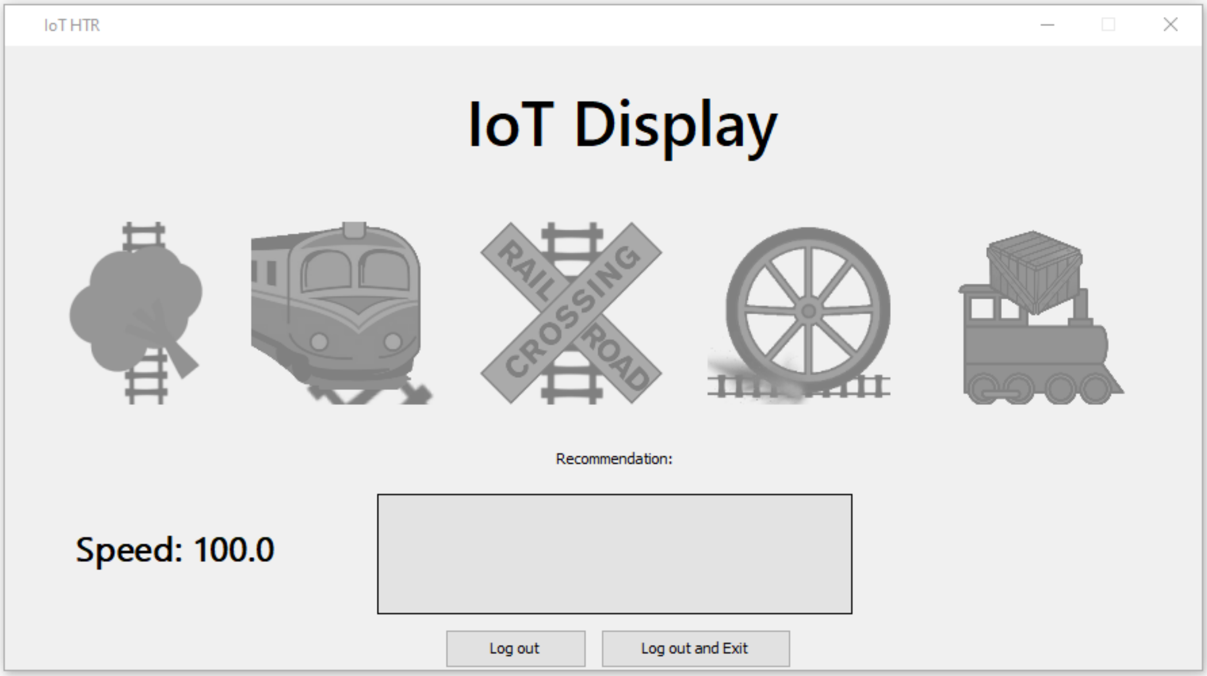
Expected Outcome: The railroad crossing warning image will light up when the crossing enters the 1 mile range. During the approach, the conductor is warned to “Slow!” for the pass.

Output:

The user successfully logs in.
Recommendations are displayed for honking the horn, as tested in 7.3.7



When the railroad crossing enters a range of 1 mile, and the horn recommendation is over the suggestion is set to “Slow!” and a sound is played.



Once the train passes the crossing, the icon grays and the recommendation is reset.

7.5 Wheels are slipping

Test Description: GPS speed data does not match the wheel RPM, indicating wheel slippage.

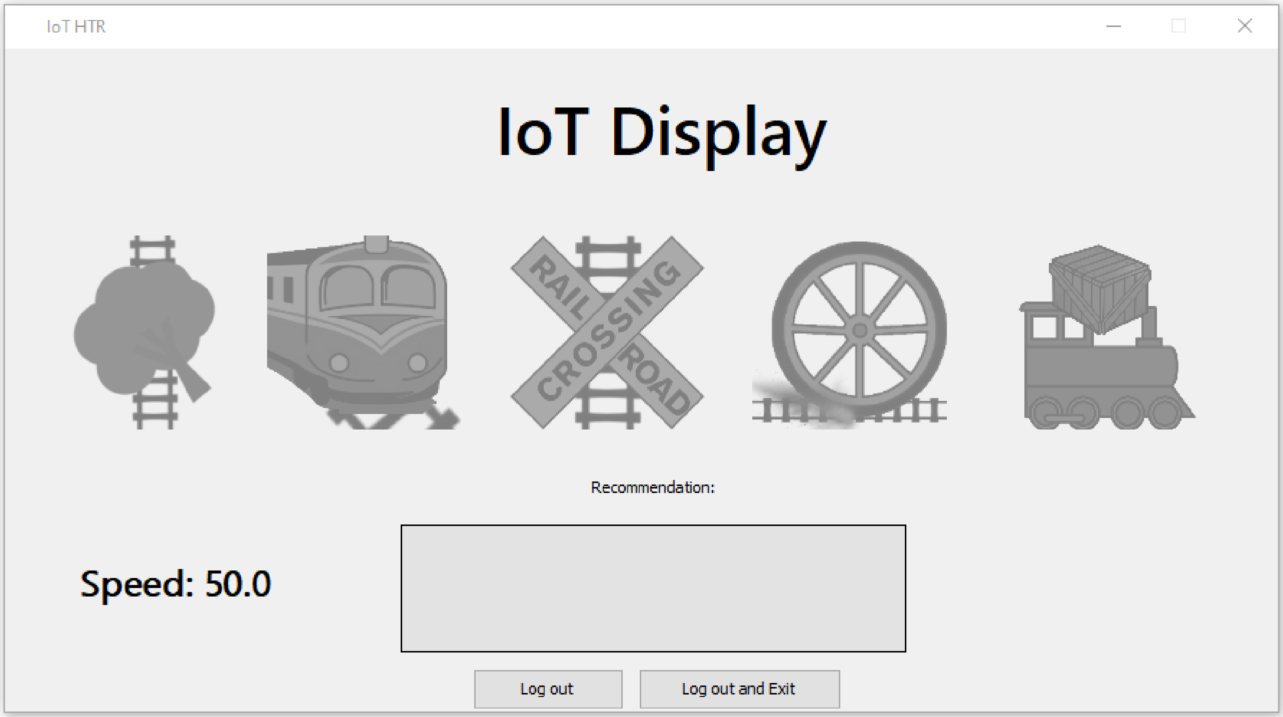
Methodology:

1. Sensor readings are generated for a train ride where wheel RPM data matches speed from GPS for some time, until RPM data is reduced, simulating wheel slippage.

Expected Outcome:

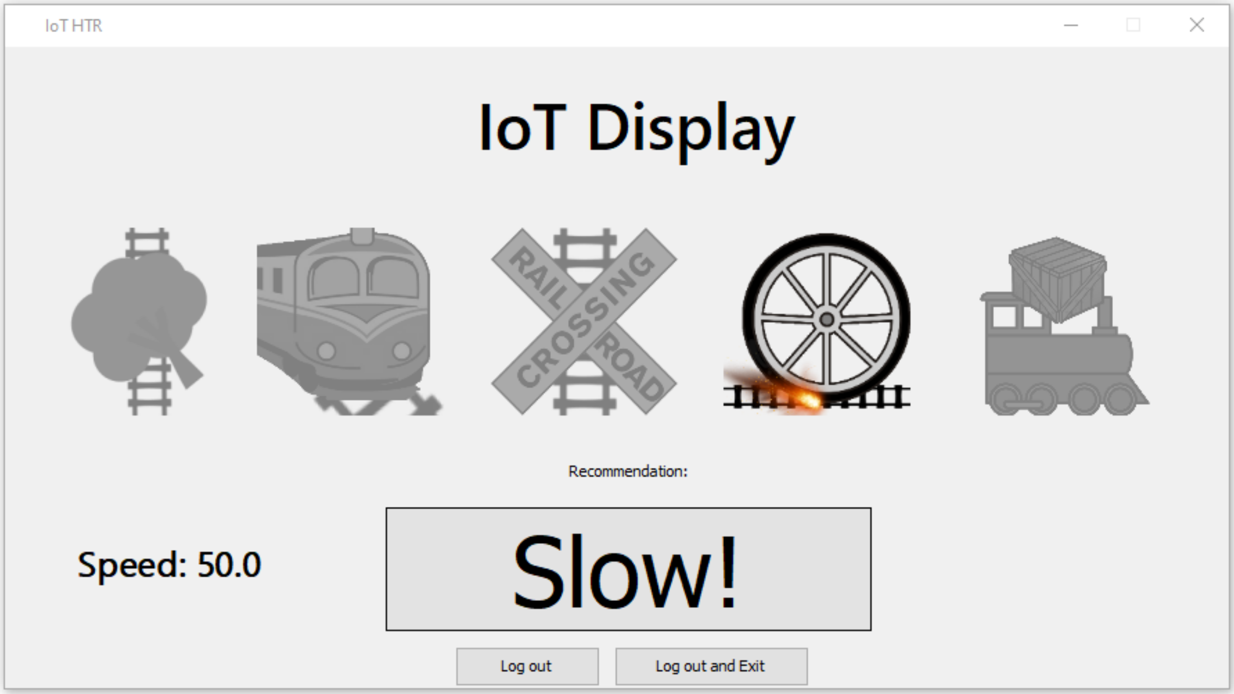
While RPM matches GPS speed, no warnings/suggestions are displayed. Once wheels begin to slip, a warning message is displayed and “Slow down” is suggested.

Output:



Speed: 50.0

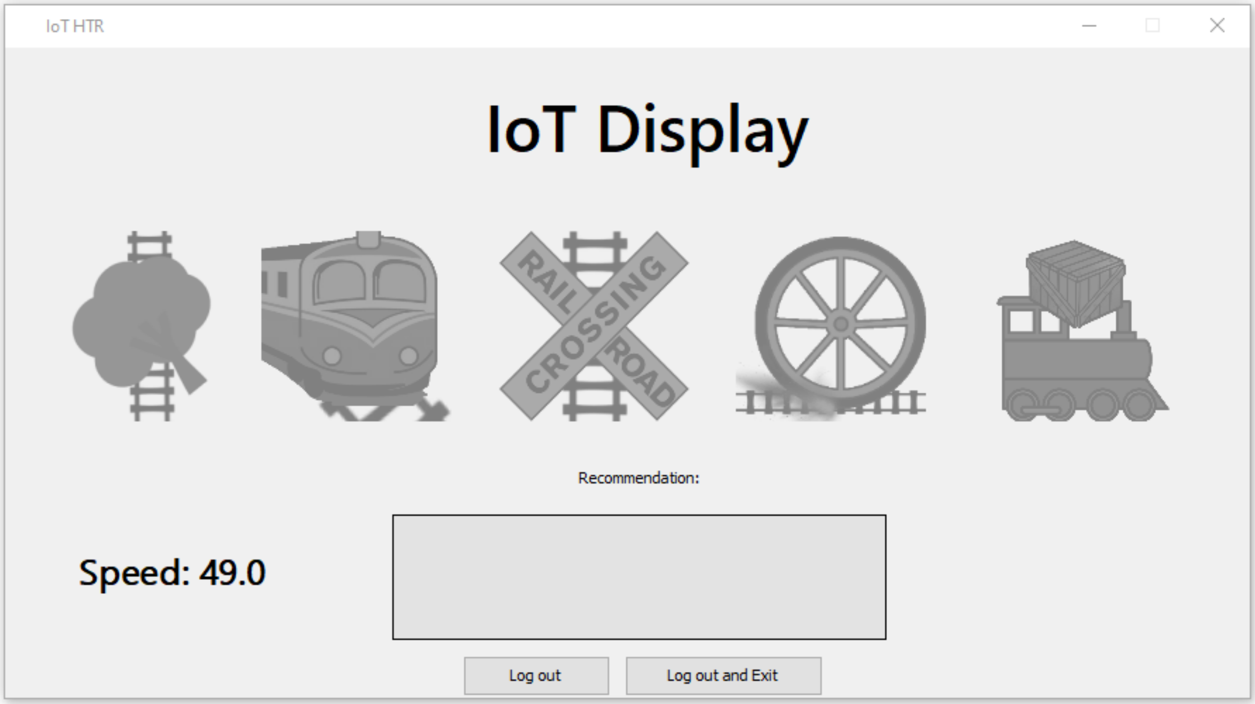
The RPM is multiplied by (10 * (24 * 3.1415 * 60 / 6336)) to obtain speed from wheels. While this matches the speed reported by the GPS (within .5 mph), no warnings are displayed.



Speed: 50.0

Slow!

RPM drops but speed stays the same, so a sound is played and “Slow!” is recommended.



Speed: 49.0

The operator slows the speed of the train to 49 and the wheel RPM matches the speed of the train, so no recommendation is displayed.

7.6 Object is on top of train

Test Description: An object is detected on top of the non-moving train, before it is removed. The train then can move safely.

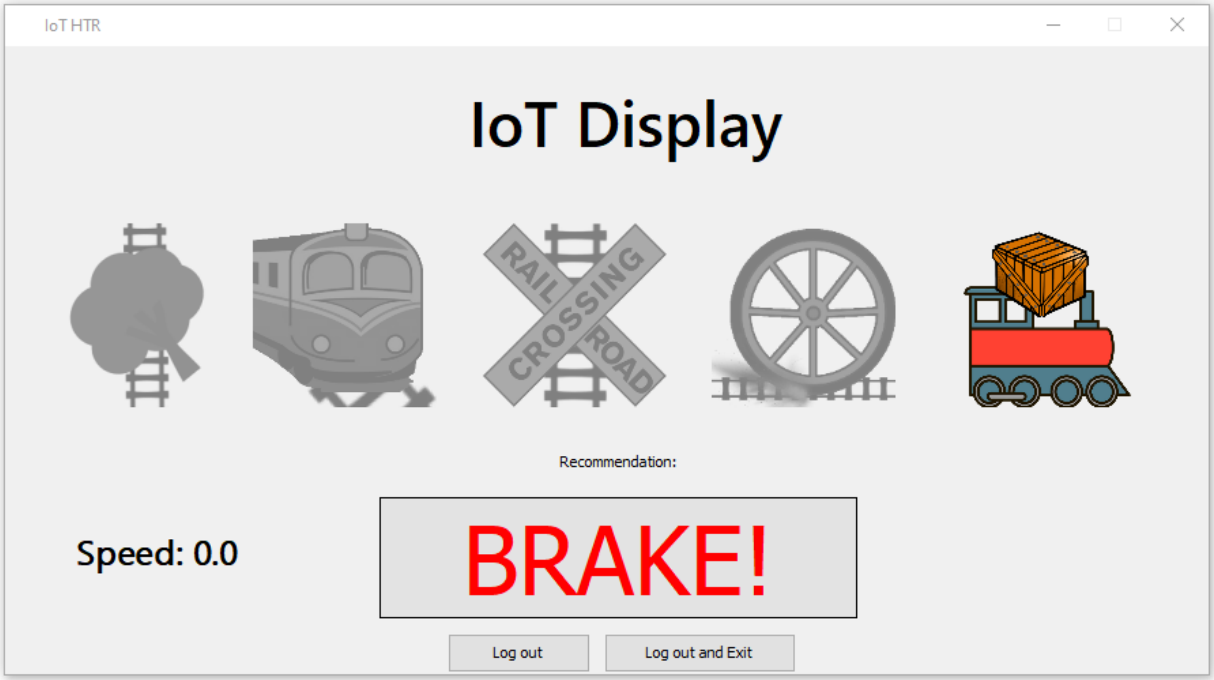
Methodology:

1. Sensor readings are generated for a train ride where something is detected on top of the train for some time, then falls off.
2. Once the object sensor detects the top of the train is clear, the train begins to move.

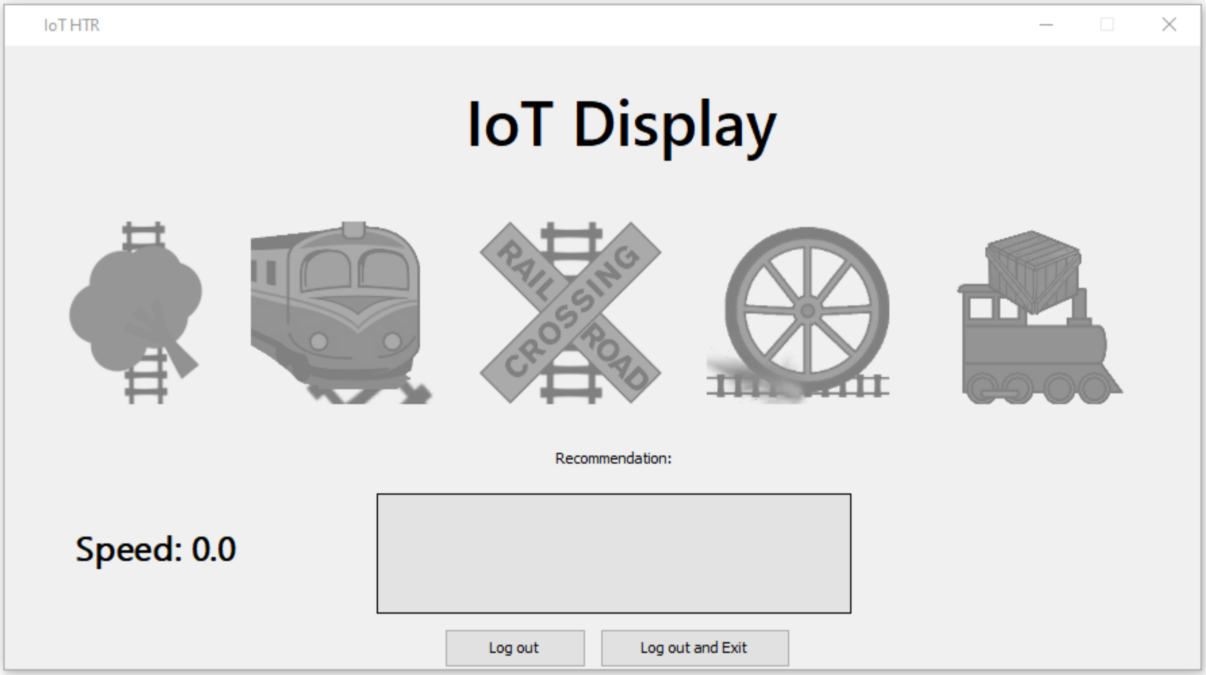
Expected Outcome:

When there is no object, no warnings/suggestions are displayed. Once the object is detected, a warning is displayed and “BRAKE!” is suggested. Once the object is no longer detected, there are no warnings/suggestions and the train begins to move safely.

Output:

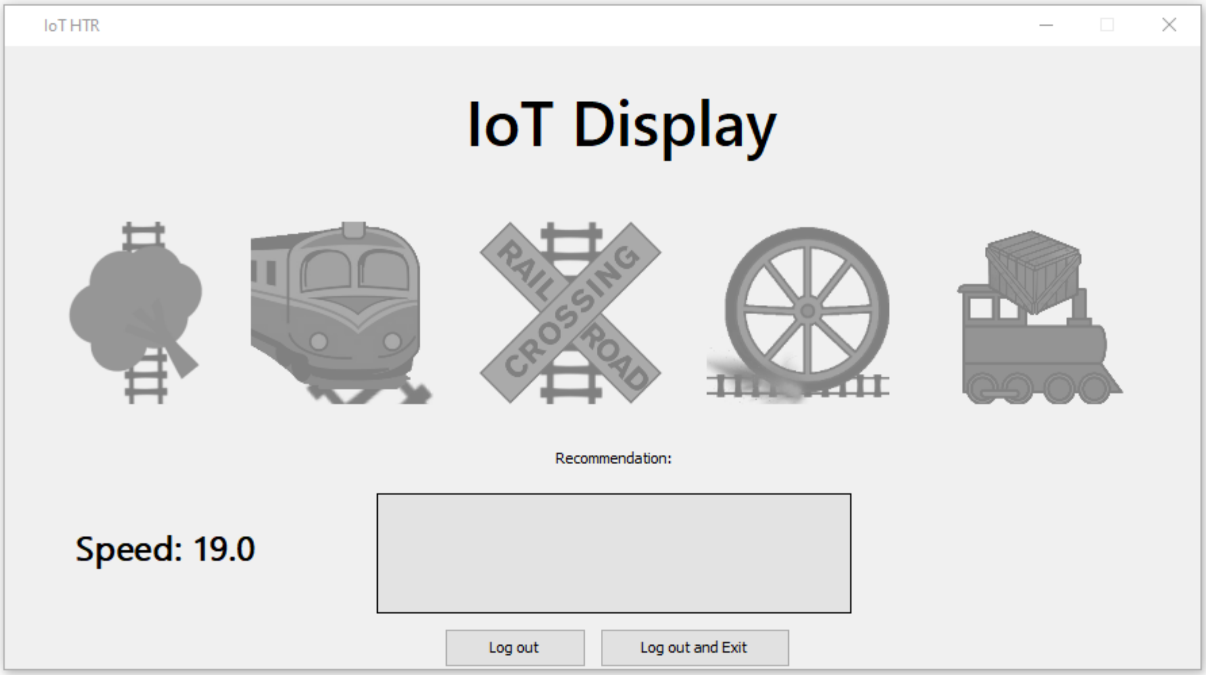


Before the train begins to move, the top object sensor detects something on top of the train. A sound is played and “BRAKE!” is recommended.



Speed: 0.0

Once the top of the train is clear, the warning and suggestion go away.



Speed: 19.0

The train then gains speed safely.

7.3.6 Recommend to Honk Horn at Railroad crossing

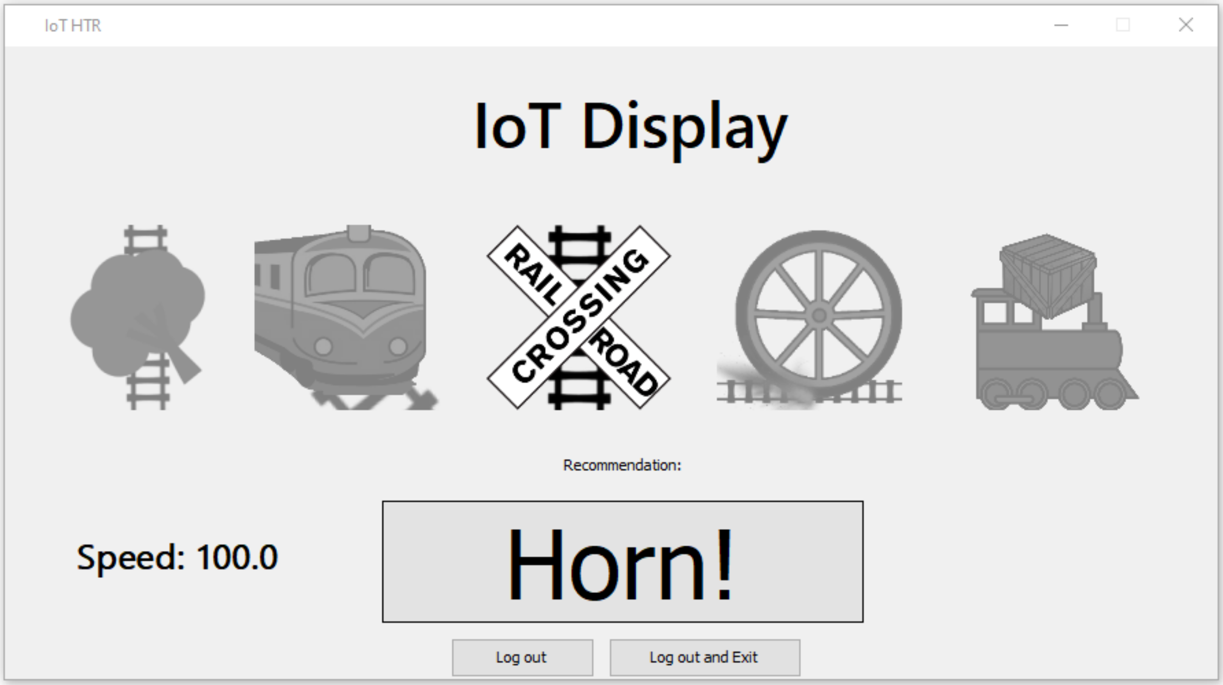
Test Description: The train approaches a railroad crossing on the track.

Methodology:

1. Sensor readings are generated for a train ride where a railroad crossing is detected at a range of 1 mile.
2. At a range of 1,000 ft, the object sensor will detect the crossing, and the distance will decrease to 0 as the train passes through.

Expected Outcome: At range of 1 mile, the recommendation is set to “Horn!” for 15 seconds to instruct the driver to honk the horn. Once the crossing enters the 1,000 ft range of the object sensor, “Horn!” will be displayed for 5 seconds.

Output:

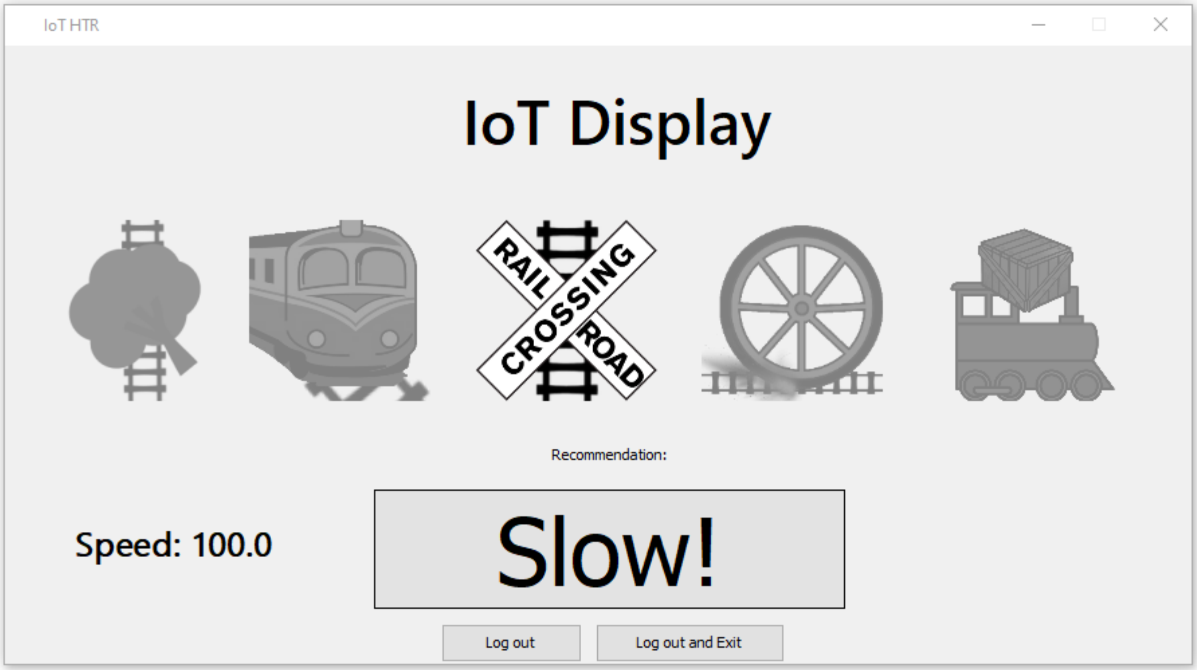


Recommendation:

Horn!

Speed: 100.0

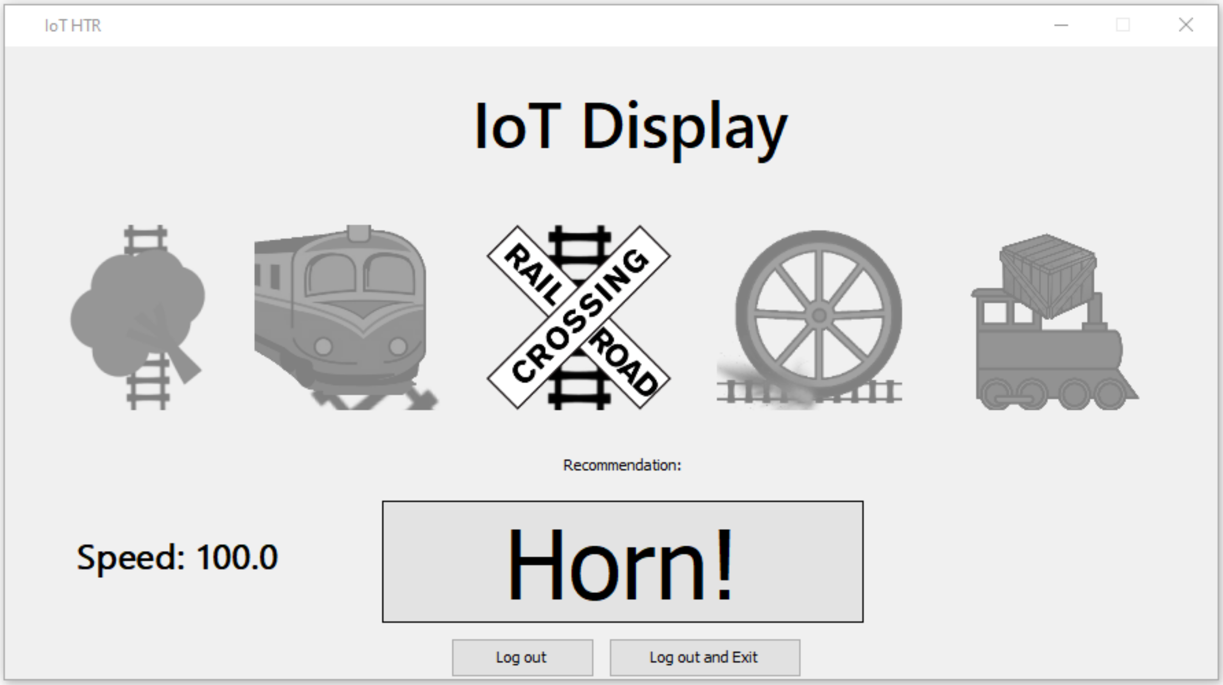
At the distance of 1 mile, a sound is made and “Horn!” is recommended for 15 seconds (based on a timer).



Speed: 100.0

Slow!

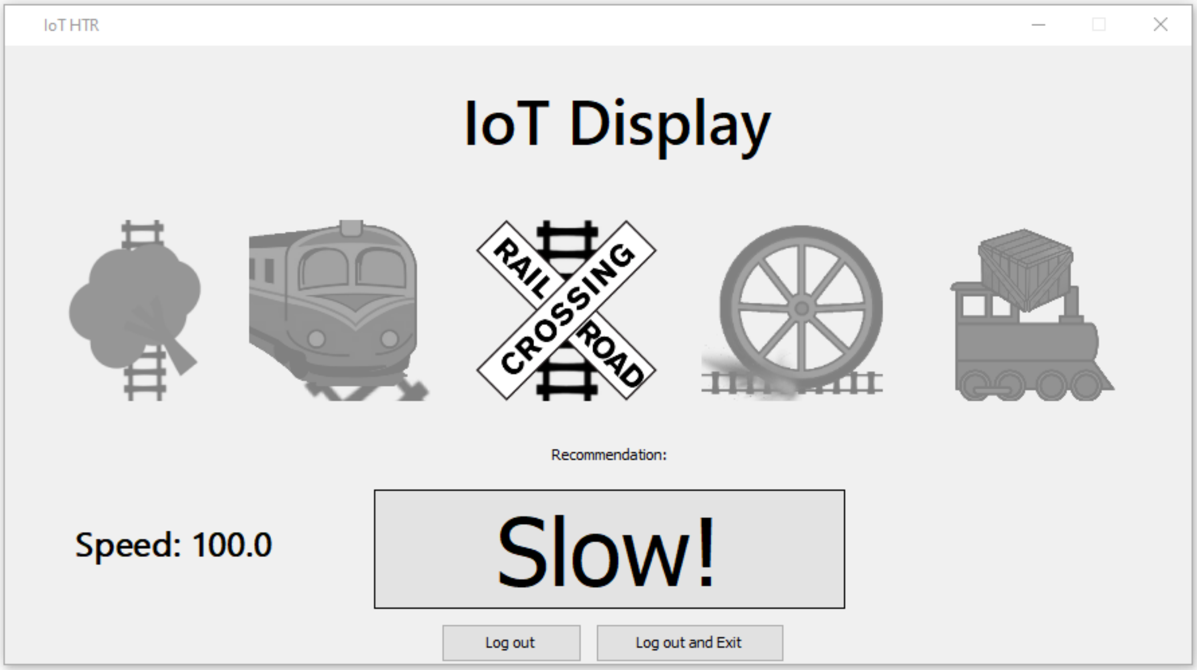
After those 15 seconds, and during the approach to the crossing, the conductor is recommended to slow the train for the pass.



Speed: 100.0

Horn!

At a range of 1,000 ft, "Horn!" is displayed for 5 seconds (based on a timer).

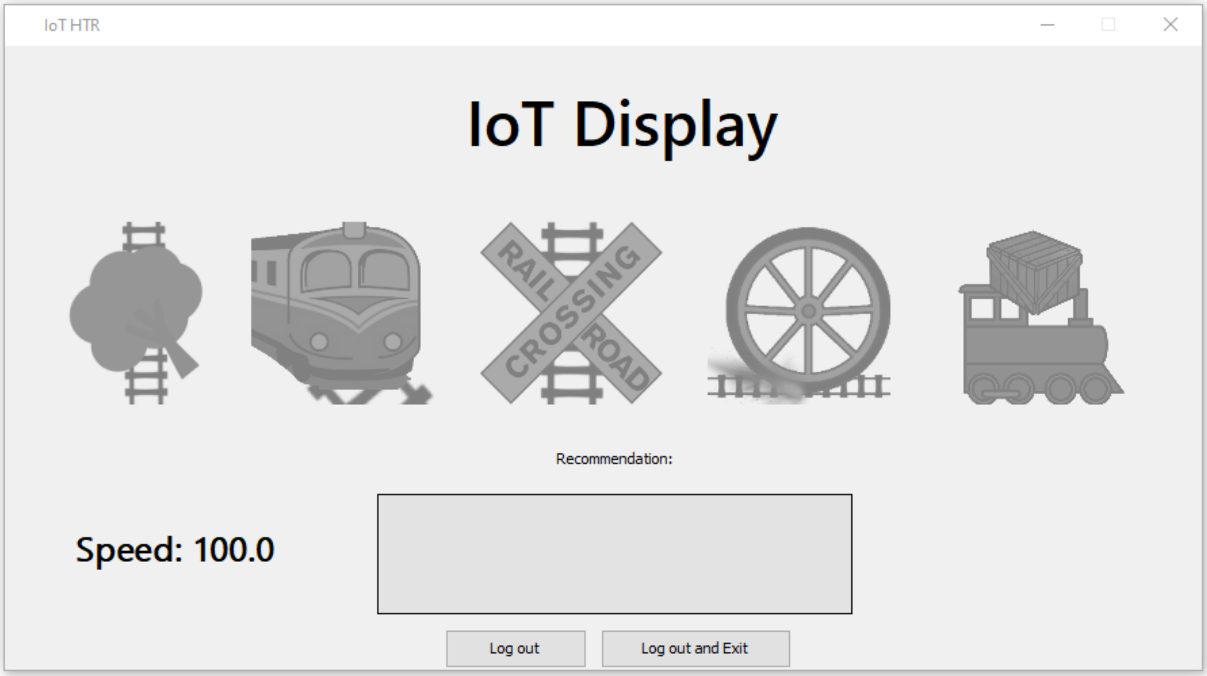


Recommendation:

Speed: 100.0

Slow!

After those 5 seconds, and during the final approach to the crossing, the conductor is again recommended to slow the train for the pass.



Recommendation:

Speed: 100.0

Once the train passes the crossing, the icon grays and the recommendation is reset.