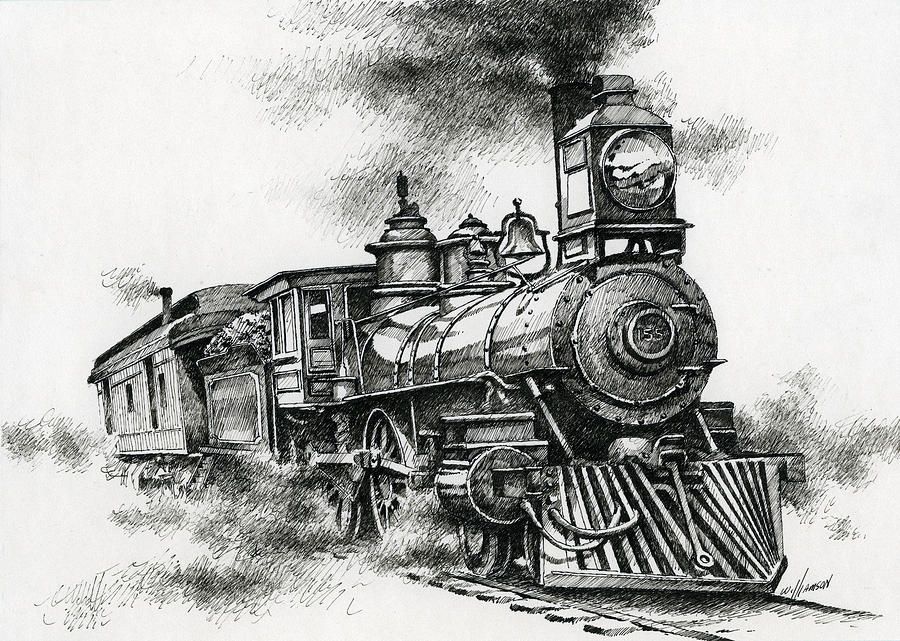
****

**Hugs the Rail Project**

**Team 5: The Big Five**

[**Section 1: Introduction**](#_dz10ki1skk8t) **4**

[**1.1 - Stakeholders and users**](#_995cgosc920) **4**

[1.2 - Expected Delivery](#_ttuozh2f1ex1) 4

[1.3 - Problem Statement](#_gcn2gp33m9hr) 4

[1.4 - Importance and values](#_5r9905fts4kn) 4

[1.5 - Approach](#_k0t0gxd81caz) 5

[1.6 - Timeline:](#_f2yvpg73j3s) 5

[**Section 2: Overview**](#_v0s68gwiyxoj) **6**

[2.1 - What is “Internet of Things”?](#_s21iq01h4tqg) 6

[2.2 - What is the problem?](#_gjsexwo1p16a) 6

[2.3 - How can IoT be used to solve the problem?](#_ewzvp1ma9e9v) 6

[2.4 - Sketching the Components](#_wkys9xfbkfct) 7

[**Section 3: Nonfunctional and Functional Requirements**](#_a7xgrqg9oour) **8**

[3.1 - Nonfunctional Requirements](#_l0zi9zpxl3xk) 8

[3.1.1 - Reliability Requirements](#_1uispdcq477) 8

[3.1.2 - Performance](#_1bzlkd5zv72s) 8

[3.1.3 - Security](#_1hf44siro188) 8

[3.1.4 - Operating System](#_oxiiehwcfntm) 9

[3.2 - Functional Requirements](#_bjvcst3ic6hk) 10

[3.2.1 - Detecting standing objects on the path, suggesting speed changes or brake.](#_rl4poorjkcps) 10

[3.2.2 - Detecting a moving object ahead and behind and their speed, suggesting speed changes or brake.](#_1ciwela6srh8) 10

[3.2.3 - Detecting gate crossings, suggesting speed changes or brake based on GPS data.](#_ms32hs2tjbu0) 10

[3.2.4 - Detecting wheel slippage using GPS data and wheel RPM, suggesting speed changes or brake.](#_92xeuturux2j) 11

[3.2.5 - Upon Detecting a Gate Crossing, the conductor should be informed to blow the horn for 15 seconds while at 1 mile away, and again when it reaches the gate for 5 seconds](#_640s9ntwdftr) 11

[**Section 4: Requirements Modeling**](#_9pxaq98yina3) **12**

[4.1 - Use Cases](#_6m5lt9snj0c5) 12

[4.1.1 - User I.D. and Password](#_j6k833vwboqy) 12

[4.1.2 - Checking Power-to-Weight ratio at start](#_rdtl31n5nf1) 14

[4.1.3 - Detecting Standing Objects on the Path](#_yxj3oxqm3qe2) 15

[4.1.4 - Detecting a Moving Object Ahead](#_9alzeqfbabk6) 16

[4.1.5 - Detecting a Moving Object Behind](#_mz7mrxoyhmnz) 18

[4.1.6 - Detecting Gate Crossings](#_por76z5ozhrr) 20

[4.1.7 - Detecting Wheel Slippage](#_8eo2lk4pt05c) 21

[4.1.8 - Honking the Horn on arrival at Gate Crossing](#_eh8azs8eqhp8) 22

[4.2 - Use Case Diagram](#_bzftn3zihveo) 23

[4.3 - Class Diagram](#_260ajffoi91b) 24

[4.4 - CRC Modeling Cards](#_mm1cgopbc91d) 25

[4.5 - Activity Diagram](#_uf8f9nmzz95t) 27

[4.6 - Sequence Diagram](#_9s8pfghdx3bh) 28

[4.7 - State Diagram](#_wdy5p2x3969z) 32

[**Section 5: Software Architecture**](#_pqss7orw3t6e) **33**

[5.1 - Architectural Styles](#_qpmp84ktqvld) 33

[5.1.1 - Data Centered Architecture](#_vdqb00vrnb72) 33

[5.1.2 - Data Flow Architecture](#_dbhp0iw73btt) 34

[5.1.3 - Call Return Architecture](#_gsz0519ndjhu) 35

[5.1.4 - Object Oriented Architecture](#_6ev82eo3w5oa) 36

[5.1.5 - Layered Architecture](#_lfljml3z32kl) 37

[5.1.6 - Model View Controller Architecture](#_cvoqvh1u3vdk) 39

[5.2 - Representing the System in Context](#_adgbkvmnzwca) 40

[5.3 - Components](#_9huofd3zjwb) 41

[5.4 - Decision: Object Oriented Architecture](#_mhk5eqwxvmt4) 42

[5.4.1 - IoT as an Object Oriented Architecture (Detailed)](#_a63q0y7zmdye) 43

[**Section 6: Project Code**](#_hbw3ho1plmji) **49**

[**Section 7: Testing**](#_5rubnmcnzvj1) **56**

[7.1 - Scenario Based Testing](#_bnzl30d87l5z) 56

[7.1.1 - Testing Use Case 1](#_yjptv97kqffn) 56

[7.1.2 - Testing Use Case 2](#_qwpcybmvju79) 57

[7.1.3 - Testing Use Case 3](#_meuwhn3upzup) 58

[7.1.4 - Testing Use Case 4](#_bu9r6xg3cq5r) 59

[7.1.5 - Testing Use Case 5](#_mwd1qj7h8wkf) 60

[7.1.6 - Testing Use Case 6](#_wv6c7usjt3uh) 61

[7.1.7 - Testing Use Case 7](#_lmh22jt90sea) 62

[7.1.8 - Testing Use Case 8](#_n530d43k16dp) 63

[7.1.9 - Extra Testing 1](#_htbg7hzexq9e) 64

[7.1.10 - Extra Testing 2](#_2oowwmndehct) 65

[7.2 - Validation Testing](#_wxewmtrrsh9) 66

[7.2.1 - Non-functional Requirements Testing](#_7n1gfrkklkd1) 66

[7.2.2 - Functional Requirements Testing](#_shiudsdtm97j) 67

# Section 1: Introduction

# 1.1 - Stakeholders and users

* Reza Peyrovian (Instructor)
* Avaneesh Kolluri (TA)
* Douglas Walsh (Team Member)
* Cynthia Loh (Team Organizer)
* Adham Abdelwahab (Team Member)
* Rafael Sanchez (Team Leader)
* Hugs The Rail Stakeholders

## 1.2 - Expected Delivery

April 6th

## 1.3 - Problem Statement

Our problem is that HTR is not as safe, cost effective, and efficient as we would like it to be. It is unable to make decisions locally without the use of wifi connectivity. HTR cannot capture data from its locomotives and the environment around them. It doesn’t have an analytic (IoT) engine that will download the latest rules for operation from the Cloud. Also, it does not have the ability for the train operator to enter command and receive status. Our team will fix these problems using the Internet of Things.

## 1.4 - Importance and values

Our software aims to directly improve the locomotive control system by better managing the information system the trains will work with, all without compromising safety and security. If we are able to correctly implement our system, we can minimize transportation speeds to the limitations of the trains, saving Hugs the Rail both time and money. We will also make a user friendly interface for the locomotive operator to use comfortably. We will not be sacrificing efficiency for safety, but can ensure that our software will only bring improvements to the transportation process.

## 1.5 - Approach

For the current project, we have decided to follow a Unified Process Model. This model allows us to review and change any requirements, and maintain communication and involvement throughout the whole project as a team. Unlike the Waterfall approach, which is purely sequential, UP allows for overlap between phases, leaving room for improvement and review. UP is also great for combining different stages. Thus, when we are working on Communication, we can also get a headstart on Planning, etc. For these reasons, a Unified Process Model seemed best.

## 1.6 - Timeline:

|  |  |  |
| --- | --- | --- |
| Deadlines | Stage | Unified Process Stages |
| Feb 16th | Introduction Section |  |
| Feb 23rd | Overview of IoT |  |
| March 1st | Communication & Planning | Inception |
| March 8th | Planning & Modeling | Elaboration |
| March 22nd | Construction | Construction |
| March 29th | Construction & Deployment | Transition |
| April 6th | Release | Production |
| April 26th | Release 2: GUI | Production |
| May 3rd | Final Release | Production |

# Section 2: Overview

## 2.1 - What is “Internet of Things”?

Like the word suggests, the Internet of Things refers to all the objects around the globe that can communicate effectively with the internet and the world via sensors. Anything can be turned into part of the IoT. From a small pill to a big airplane. Connecting all objects and adding sensors to them allows for a higher level of intelligence, enabling them to send real time data and make decisions, without the need of a human being. Since our project will involve trains, this would mean somehow embedding a system which adds intelligence to the train. For instance, sensors could be installed in the train, allowing it to make smart decisions regarding local data, given that the wifi is down.

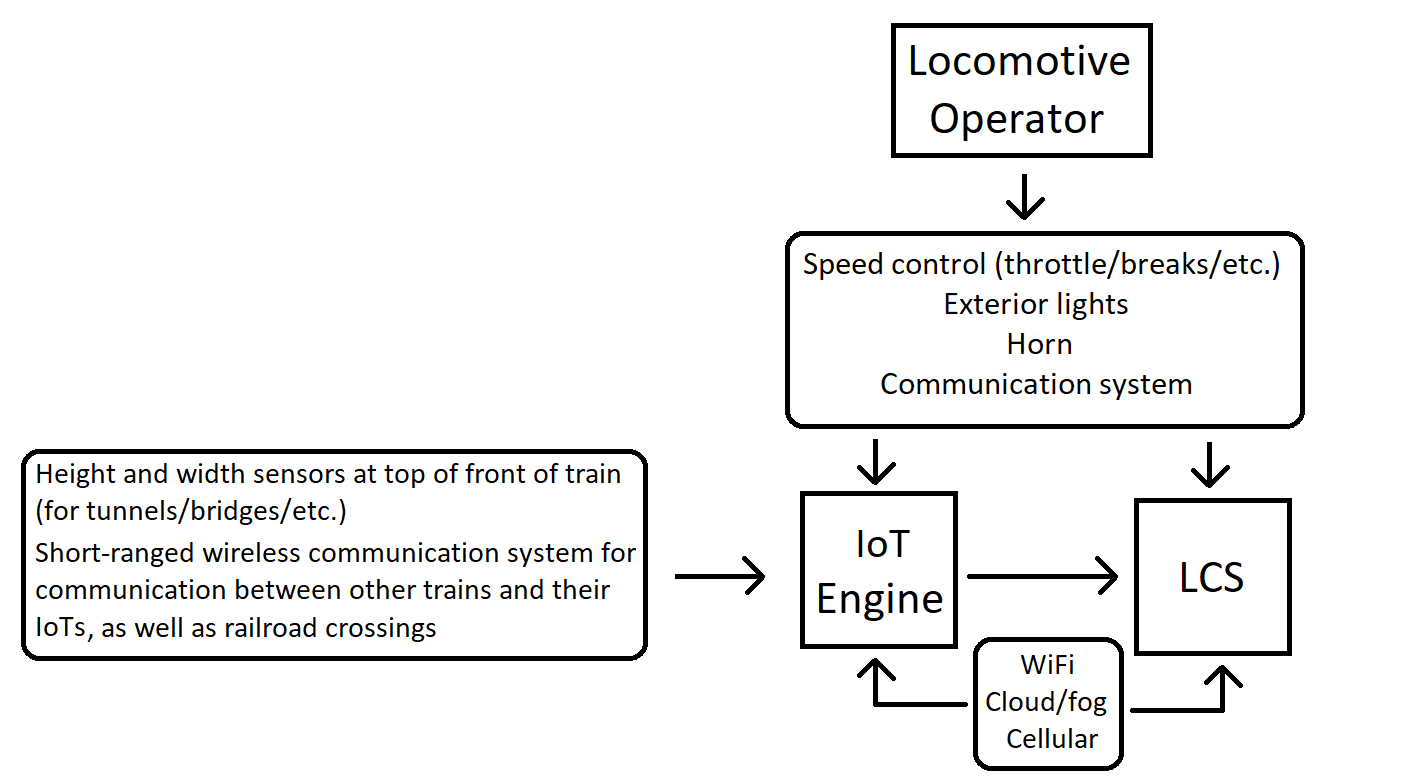
## 2.2 - What is the problem?

Currently, train operations are dependent on wifi or cellular networks to function. Without wifi-cellular connectivity, trains and their operators would not be able to receive necessary information from the Fog/Cloud servers. Some of this information includes travel conditions, traffic, etc. Also, train maintenance is very important for train efficiency and safety, yet some trains can experience hundreds of breakdowns in a year. The purpose of this project is to use IoT so trains can make decisions locally when wifi-cellular networks have failed and to improve the reliability and safety of boarding a train.

## 2.3 - How can IoT be used to solve the problem?

Iot can be used for a variety of trip improving features. One basic one is retrieving paths and routes that it can take for optimal transit from the server. This can also be effective in fetching paths of other trains as well. We can also use IoT to install sensors around wheels or certain train parts to understand speed and status of the trains performance. This can quickly communicate to a server if a train is in danger or not performing properly and may need maintenance. The IoT will allow us to gather data regularly and present it in a format that people can easily recognize and understand.

## 2.4 - Sketching the Components

****

# Section 3: Nonfunctional and Functional Requirements

## 3.1 - Nonfunctional Requirements

### 3.1.1 - Reliability Requirements

**R - 1**  IoT HTR should be operable under extreme weather conditions in temperatures ranging from -10 F to 150 F.

**R - 2** Attaching strong covers to the IoT on all sides will reinforce its components.

**R - 3** if the IoT drops from 5 feet high, it shall still function without any problems.

**R - 4** Monthly tests will be conducted on the trains that involve 10,000 test cases.

### 3.1.2 - Performance

**R - 5** We will not allow devices that respond slower than 0.5 seconds and are willing to purchase modernized equipment to ensure we stay below this number.

**R - 6** IoT system is able to support all necessary sensors to ensure functionality all across the train.

**R - 7** The IoT will let operators know all local information about the train.

### 3.1.3 - Security

**R - 8** Train operators are only able to access the IoT with a User ID and Password.

**R - 9** When using IoT, the interface will prompt the user to input an id and password before allowing any further use of the IoT.

**R - 10** If correct credentials are inputted, the user will be allowed access to the information gathered by the IoT sensors.

**R - 11** If incorrect credentials are inputted, the prompt remains and the user is not allowed access into the IoT network.

### 3.1.4 - Operating System

**R - 12** The HTR shall run on updated and recent versions of Windows 10.

## 3.2 - Functional Requirements

### 3.2.1 - Detecting standing objects on the path, suggesting speed changes or brake.

**R - 1**  Install sensors in the front of the train to detect standing objects on the path.

**R - 2**  Sensors will be able to detect objects up to 1 mile in the direction the train is heading.

**R - 3** The sensors will find the distance between the train and stationary objects.

**R - 4**  If a standing object is detected 1 mile away from the train, the IoT will provide a warning.

### 3.2.2 - Detecting a moving object ahead and behind and their speed, suggesting speed changes or brake.

**R - 5** Install sensors in the front and in the back of the train to detect moving objects on the path.

**R - 6** Sensors will be able to detect objects up to 1 mile ahead and behind the train.

**R - 7** The sensors will find the speed of moving objects detected.

**R - 8** If there is a detected moving object behind the train and the object’s speed is greater than the train’s, the IoT will warn the LCO.

**R - 9** If there is a moving object with a speed slower than the train’s, the IoT will warn the LCO.

### 3.2.3 - Detecting gate crossings, suggesting speed changes or brake based on GPS data.

**R - 10** Through GPS, IoT shall detect if there is a gate crossing within a 1 mile radius.

**R - 11** If the gate crossing is open and is not closing, then the IoT shall warn the train operator.

**R - 12** As the train approaches any railroad crossing, warn the train operator.

### 3.2.4 - Detecting wheel slippage using GPS data and wheel RPM, suggesting speed changes or brake.

**R - 13** Sensors on both sides of the train should detect the Rotations Per Minute of all wheels.

**R - 14** GPS Data shall be checked every 5 seconds by the IoT to detect if the wheels are slipping from the rail at any moment that the train is moving.

**R - 15** If the wheels are slipping, warn the LCO.

**R - 16** Before the train starts, weight detectors should check if the train has an equal power-to-weight ratio.

**R - 17** If the train doesn’t have an equal power-to-weight ratio, warn the locomotive operator.

### 3.2.5 - Upon Detecting a Gate Crossing, the conductor should be informed to blow the horn for 15 seconds while at 1 mile away, and again when it reaches the gate for 5 seconds

**R - 18** IoT should check that a gate crossing is within a mile.

**R - 19** If there is a gate crossing, inform the locomotive operator to honk the horn for 15 seconds.

**R - 20** Once the train reaches the gate, the LCO should be informed to honk the horn again for 5 seconds.

# Section 4: Requirements Modeling

## 4.1 - Use Cases

### 4.1.1 - User I.D. and Password

**Use Case:** Input user credentials to start up the engine.

**Use No:** 1

**Primary Actor:** Locomotive Operator.

**Secondary Actor:** IoT, Sensors.

**Goal:** To check that the user inputted valid credentials.

**Preconditions:** System was programmed beforehand for a user I.D and password.

**Trigger:**  The Locomotive Operator just turned the IoT Engine on.

**Scenario:**

1. LCO (Locomotive Operator) turns on the IoT Engine.

2. IoT asks LCO to enter I.D. and password.

3. A prompt tells the LCO that the locomotive started successfully.

4. Sensory data has been read by the IoT, and they indicate everything is working correctly.

5. LCO sees a prompt indicating the IoT is ready.

**Exceptions:**

1. LCO enters the wrong I.D and/or password: IoT asks the LCO to re-enter I.D. and password.

2. Locomotive didn’t start successfully due to an unknown problem: Warn the LCO the locomotive could not start.

3. Sensory data could not be read by the IoT: Warn the LCO sensory data could not be read.

4. Sensory data was read by IoT but it indicates that there is a standing object ahead, thus the train cannot start: Warn the LCO of this problem.

5. Sensory data was read by IoT but it indicates that there is a moving object ahead or behind the train, thus the train cannot start: Warn the LCO of this problem.

6.Sensory data was read by IoT but it indicates that there is wheel slippage, thus the train cannot start: Warn the LCO of this problem.

### 4.1.2 - Checking Power-to-Weight ratio at start

**Use Case**: IoT checks that the train has roughly equal power-to-weight ratio.

**Use No:** 2

**Primary Actor:** IoT.

**Secondary Actor:** Locomotive Operator, Weight Detectors.

**Goal:** Prevent damage to the wheels and the train by checking a power-to-weight ratio that enables smooth acceleration from a ‘cold start’.

**Preconditions:** Weight detectors are working properly, the IoT engine was turned on, and maximum weight (286 000 lbs) the train can carry has been programmed beforehand.

**Trigger:** IoT detects through weight detectors that there is too much weight on the train. (Current Weight is greater than 286 000 lbs)

**Scenario:**

1. IoT retrieves data from weight detectors regarding current weight of the train.

2. IoT compares the current weight with its max weight.

**Exceptions:**

1. IoT failed to retrieve data from weight detectors: Warn the LCO.

2. Current weight of the train exceeds the expected cargo of the train: Warn the LCO to lighten the cargo on the train before starting.

### 4.1.3 - Detecting Standing Objects on the Path

**Use Case**: IoT checks if there is a standing object ahead of the train.

**Use No:** 3

**Primary Actor:** IoT.

**Secondary Actor:** Sensors, Locomotive Operator.

**Goal:** Prevent a potential collision with a standing object in front of the train.

**Preconditions:** Train started successfully and is moving, and IoT is functioning correctly by reading data from sensors.

**Trigger:** A standing object in front of the train was detected by the sensors within 1 mile.

**Scenario:**

1. IoT reads data from the sensors.

2. IoT reads that a standing object is ahead.

3. IoT warns the LCO that there is a standing object ahead of the train.

4. IoT suggests the LCO to honk the horn and to slow down the train.

**Exceptions:**

1. IoT failed to read data from the sensors: Warn the LCO that sensory data could not be read.

### 4.1.4 - Detecting a Moving Object Ahead

**Use Case**: IoT checks if there is a moving object going at a lower speed ahead of the train.

**Use No:** 4

**Primary Actor:** IoT.

**Secondary Actor:** Locomotive Operator, Sensors, GPS Data.

**Goal:** Prevent a potential collision with a moving object in front of the train.

**Preconditions:** Train started successfully and is moving at a constant speed of 200 mph, IoT is functioning correctly by reading data from sensors, and data for the train’s speed has been retrieved.

**Trigger:** A moving object ahead going at a slower speed than the train’s (less than 200 mph) was detected by the sensors.

**Scenario:**

1. Sensors detect a moving object in front of the train.

2. IoT reads data from the sensors indicating there is a moving object ahead.

3. IoT reads data from the sensors regarding the speed of the moving object ahead.

4. IoT retrieves data from GPS regarding the speed of the train.

5. IoT compares the current speed of the train with the speed of the moving object.

**Exceptions:**

1. IoT could not read data from the sensors: Warn the LCO that sensory data could not be read.

2. IoT could not read the speed of the moving object: Warn the LCO that sensory data regarding the speed of the moving object could not be read.

3. IoT could not retrieve data from GPS: Warn the LCO that GPS data could not be retrieved and is instead using the train’s speedometer data.

4. Train is moving faster than the moving object up ahead: Suggest the LCO to slow down the train and honk the horn.

### 4.1.5 - Detecting a Moving Object Behind

**Use Case**: IoT checks if there is a moving object going at a higher speed behind the train.

**Use No:** 5

**Primary Actor:** IoT.

**Secondary Actor:** Locomotive Operator, Sensors, GPS Data, Speedometer.

**Goal:** Prevent a potential collision with a moving object behind the train.

**Preconditions:** Train started successfully and is moving (at around 200 mph), IoT is functioning correctly by reading data from sensors, and data for the train’s speed has been retrieved.

**Trigger:** A moving object behind the train going at a faster speed than the train’s (more than 200 mph) was detected by the sensors.

**Scenario:**

1. Sensors detect a moving object behind the train.

2. IoT reads data from the sensors indicating there is a moving object behind.

3. IoT reads data from the sensors regarding the speed of the moving object behind.

4. IoT retrieves data from GPS regarding the speed of the train.

5. IoT compares the current speed of the train with the speed of the moving object.

**Exceptions:**

1. IoT could not read data from the sensors: Warn the LCO that sensory data could not be read.

2. IoT could not read the speed of the moving object: Warn the LCO that sensory data regarding the speed of the moving object could not be read.

3. IoT could not retrieve data from GPS: Warn the LCO that GPS data could not be retrieved and is instead using the train’s speedometer data.

4. Train is moving slower than the moving object behind: Suggest the LCO to speed up the train and honk the horn.

### 4.1.6 - Detecting Gate Crossings

**Use Case:** IoT checks if there is a gate crossing ahead.

**Use No:** 6

**Primary Actor:** IoT.

**Secondary Actor:** Locomotive Operator, GPS Data.

**Goal:** Prevent a potential accident due to a malfunctioning gate crossing ahead of the train.

**Preconditions:** Train started successfully and is moving, IoT is functioning, and GPS data for the gate crossing’s location and functioning status has been retrieved.

**Trigger:** IoT retrieved data from GPS regarding the presence of a gate crossing ahead. (within a mile)

**Scenario:**

1. IoT reads from GPS that a gate crossing is ahead.

2. IoT warns the LCO that there is a gate crossing up ahead and suggests honking the horn.

3. IoT retrieves the functioning status of the gate crossing ahead by reading GPS data about the gate crossing’s functioning status.

**Exceptions:**

1. IoT could not read data from GPS: Warn the LCO that GPS data could not be read.

2. Gate crossing is not functioning: Warn the LCO that gate crossing is not functioning (it’s not closing as it ought to be) and suggest honking the horn and braking.

### 4.1.7 - Detecting Wheel Slippage

**Use Case:** IoT checks if the power to weight ratio is at least 1.

**Use No:** 7

**Primary Actor:** IoT.

**Secondary Actor:** Sensors, GPS Data, Locomotive Operator.

**Goal:** Prevent a potential derailing by checking if there is wheel slippage based on RPM (rotations per minute) of the wheels and GPS Data.

**Preconditions:** Train started successfully and is moving, IoT is functioning, and GPS data about the current speed of the train has been successfully retrieved.

**Trigger:** Wheel slippage is detected if the RPM data on the train wheels does not align with the GPS data for how fast the train should be moving.

**Scenario:**

1. IoT detects wheel slippage as the train GPS data shows it is not moving as fast as it should.

2. IoT displays a warning to the Locomotive Operator that wheel slippage may be occuring.

3. IoT suggests the Locomotive Operator to slow down the train.

4. IoT checks if wheel slippage continues.

5. IoT suggests the Locomotive Operator to stop the train.

**Exceptions:**

1. Data from GPS regarding speed cannot be retrieved: Warn the operator and compare the wheel’s RPM to previous RPM data.

### 4.1.8 - Honking the Horn on arrival at Gate Crossing

**Use Case:** IoT checks if the train arrived at gate crossing.

**Use No:** 8

**Primary Actor:** IoT.

**Secondary Actor:** Locomotive Operator, GPS Data.

**Goal:** Warn LCO again to honk the horn for 5 seconds upon arrival at the gate crossing.

**Preconditions:** Train started successfully and is moving, IoT is functioning, and GPS data for the gate crossing’s location and functioning status has been retrieved.

**Trigger:** IoT is now aware that there is a gate crossing ahead, and that the train has arrived at a gate crossing.

**Scenario:**

1. IoT reads that there is a gate crossing and that the train has arrived at one.

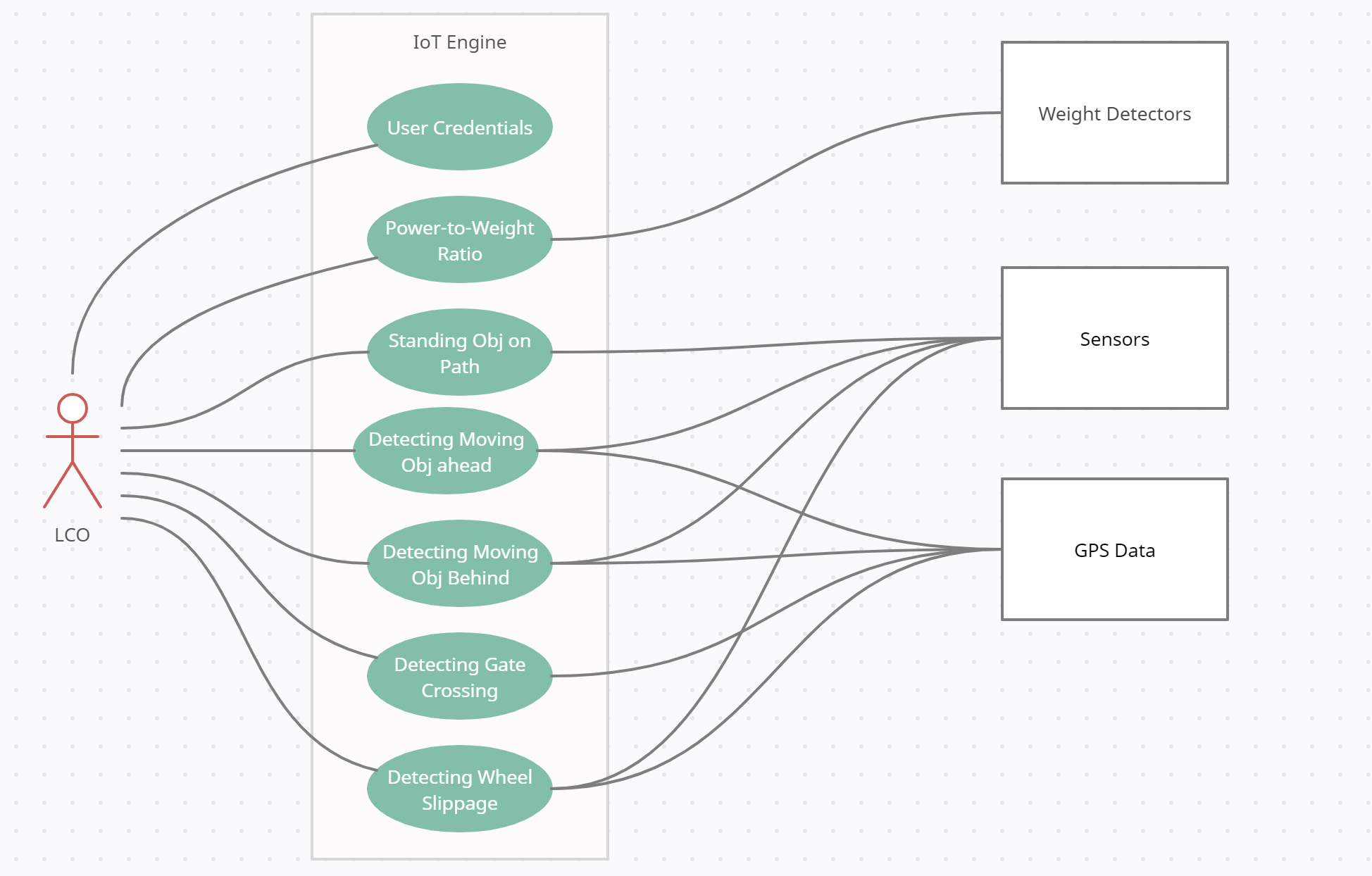
2. IoT suggests to honk the horn for 5 seconds.

**Exceptions:**

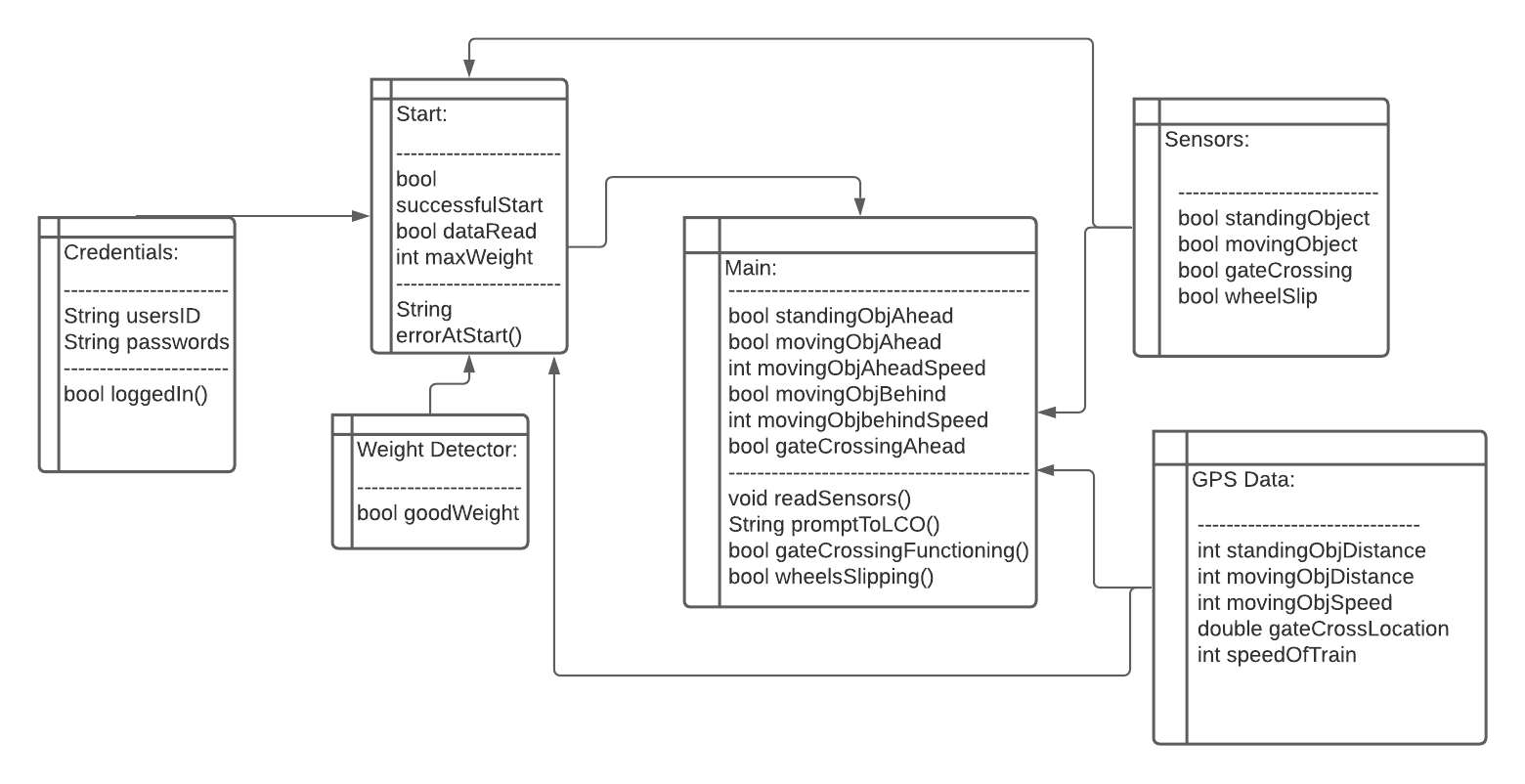
1. IoT could not read data from GPS: Warn the LCO that GPS data could not be read.

### 

## 4.2 - Use Case Diagram



## 4.3 - Class Diagram



## 4.4 - CRC Modeling Cards

|  |  |
| --- | --- |
| **Start** | |
| **Responsibility** | **Collaborators** |
| Contains user ID | Start, Credentials, Weight Detector, Sensors, GPS Data |
| Contains password |
| Makes sure LCO inputted correct info |
| Checks that data was read successfully |
| Establishes GPS Communication |

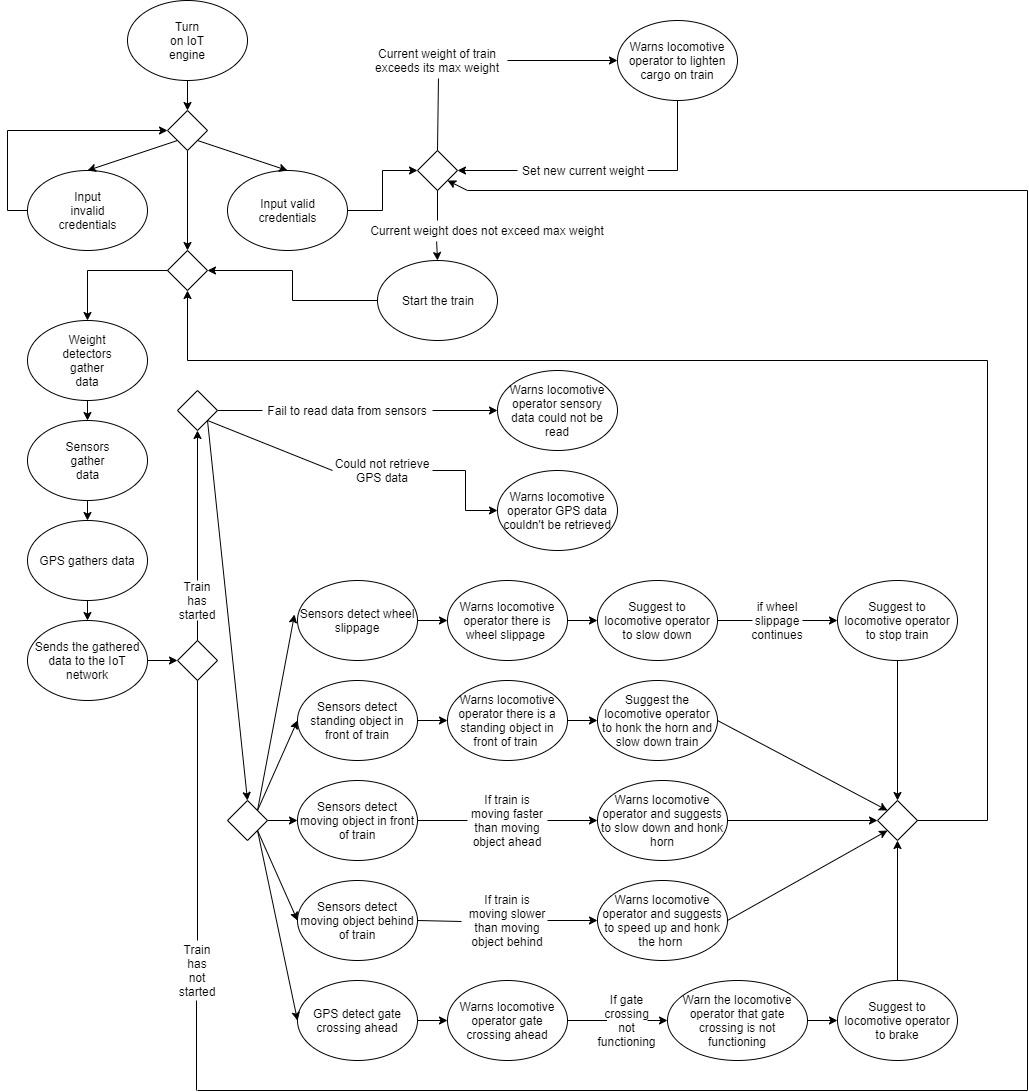
|  |  |
| --- | --- |
| **Weight Detector** | |
| **Responsibility** | **Collaborators** |
| Checks weight of train | Weight Detector, Start |
| Sends data to Start regarding weight of cargo |

|  |  |
| --- | --- |
| **Sensors** | |
| **Responsibility** | **Collaborators** |
| Contains data regarding a standing object is ahead | Sensors, Main |
| Sends data about the standing object |
| Contains data regarding speed of moving object ahead |
| Contains data regarding speed of moving object behind |
| Sends required data about moving object |
| Sends data about RPM of wheels |

|  |  |
| --- | --- |
| **GPS** | |
| **Responsibility** | **Collaborators** |
| Contains data regarding Gate Crossing’s location | GPS, Main |
| Contains data regarding Gate Crossing’s functioning status |
| Sends data about Gate Crossings |

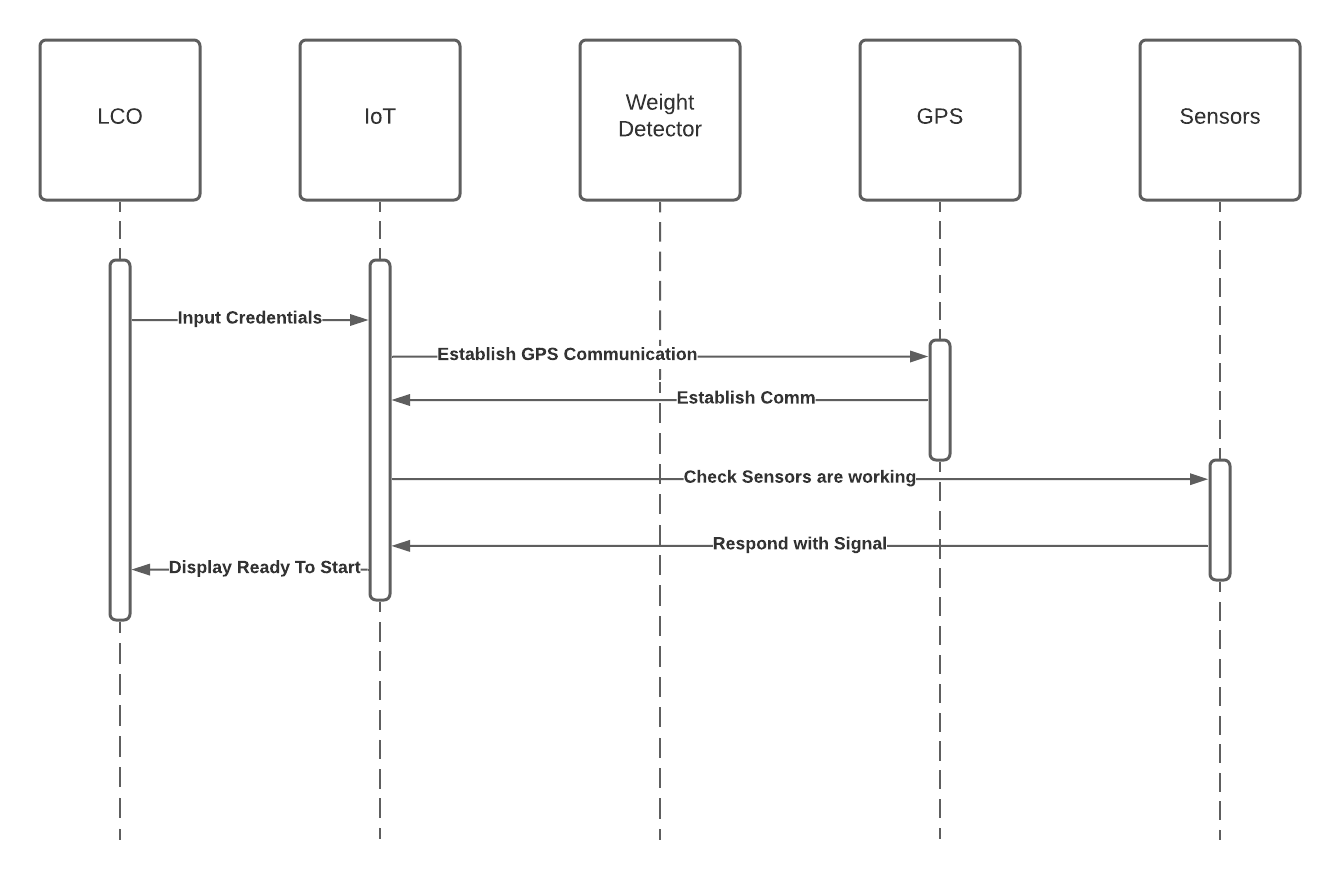
|  |  |
| --- | --- |
| **Main** | |
| **Responsibility** | **Collaborators** |
| Handles input from the sensors | Sensors, GPS |
| Checks that the inputs are correct |
| Checks for exceptions or warnings |
| Displays those warnings to LCO |
| Constantly communicating with GPS and Sensors |

## 4.5 - Activity Diagram

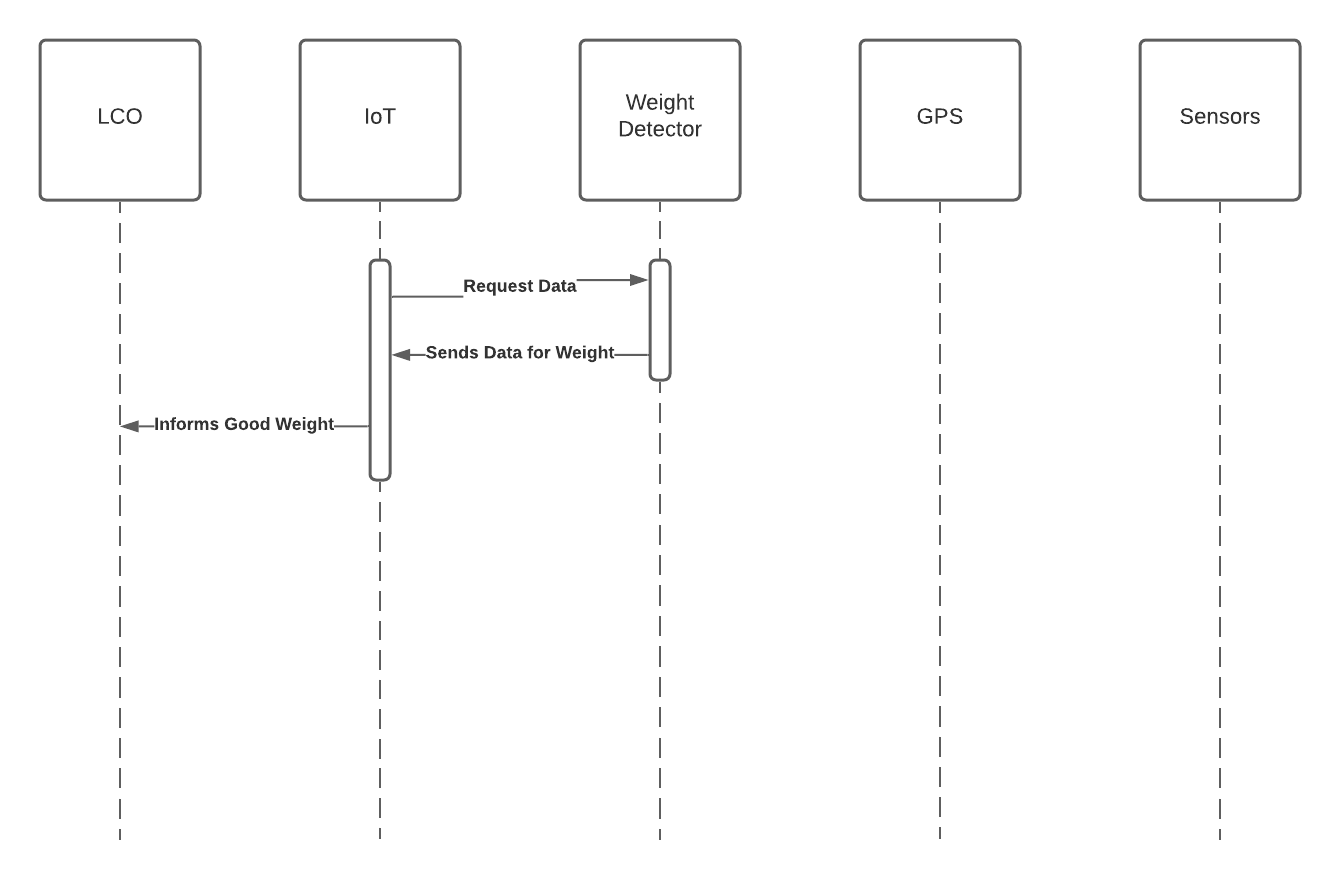
****

## 4.6 - Sequence Diagram

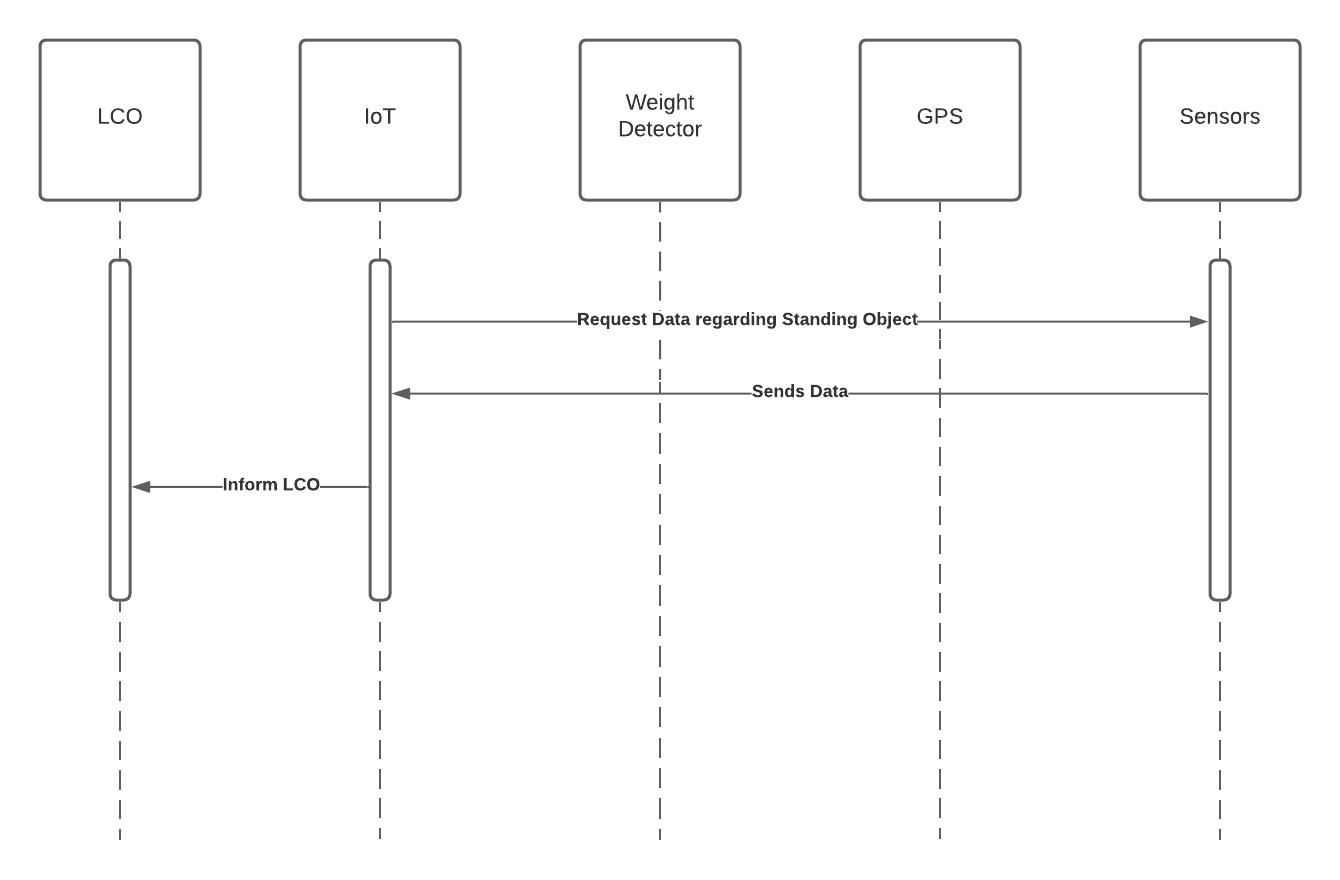
**Use Case 1**

****

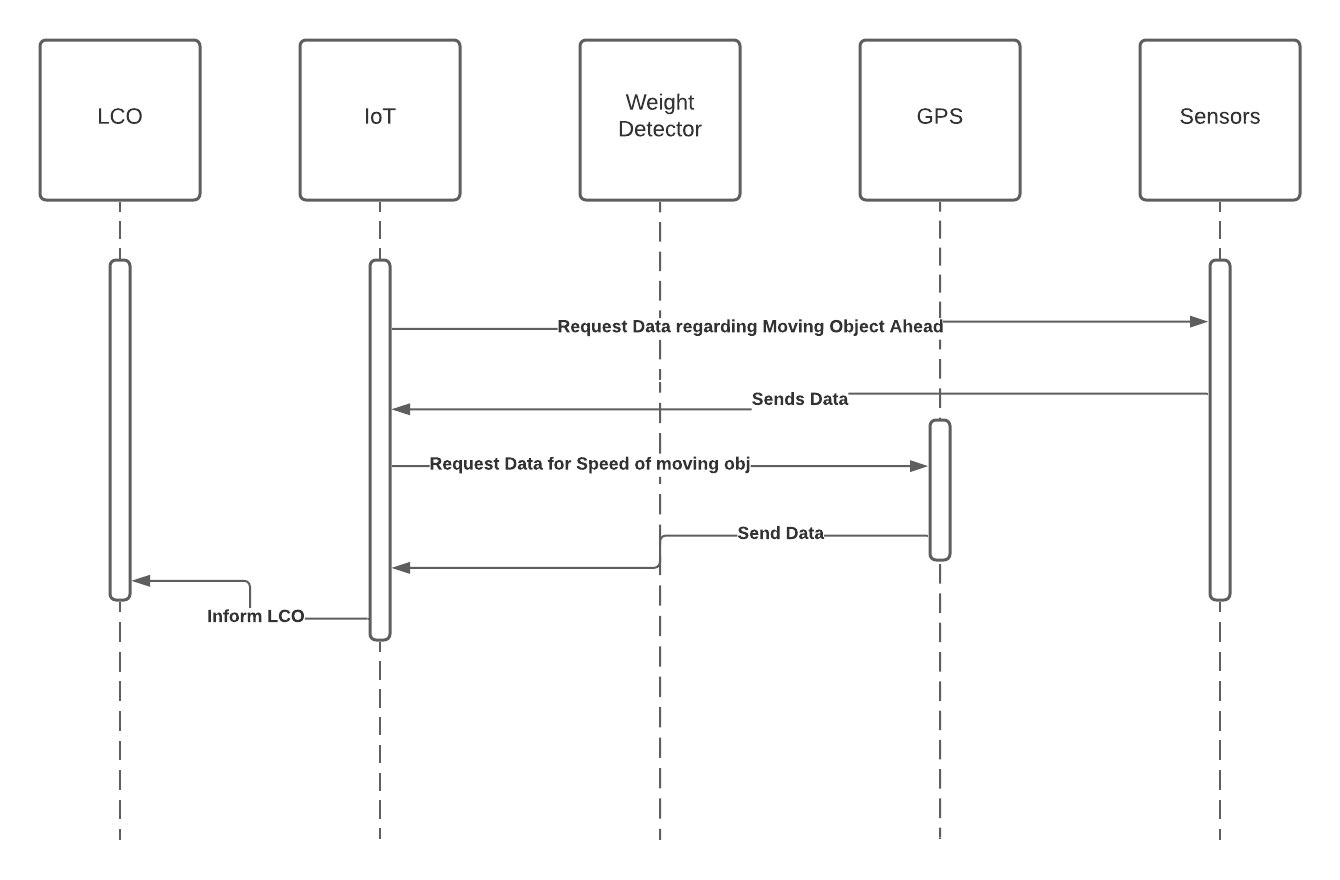
**Use Case 2**

****

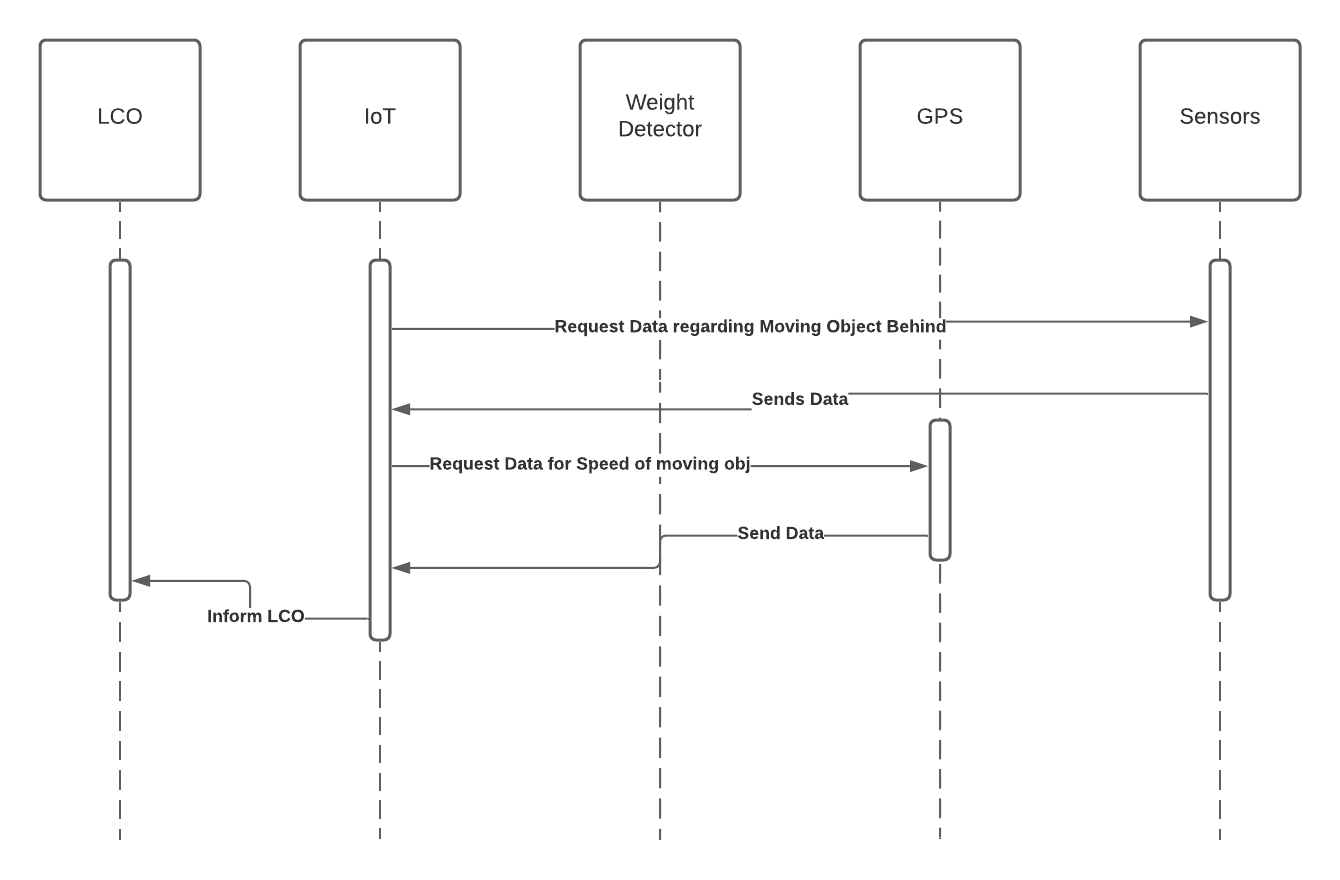
**Use Case 3**

****

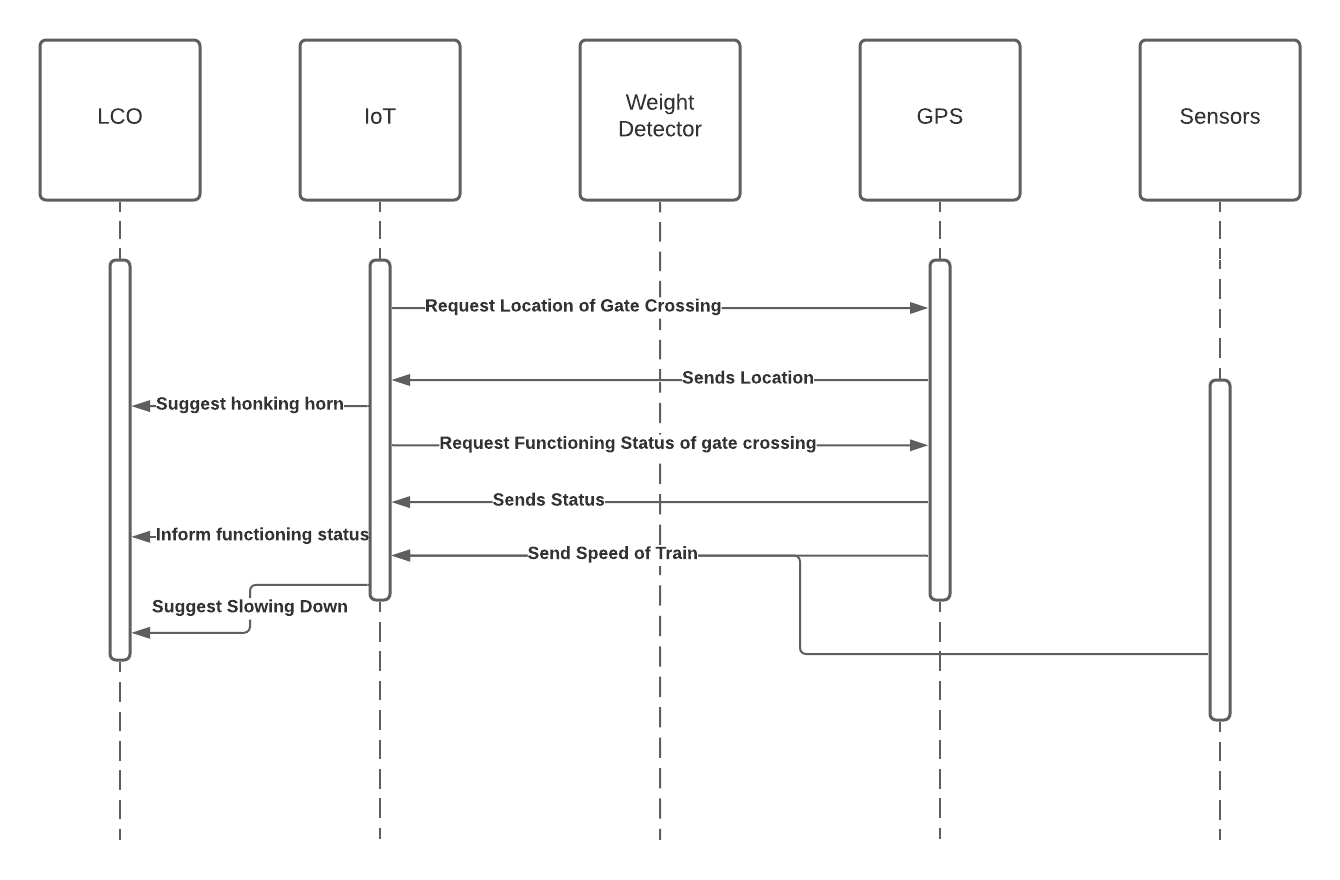
**Use Case 4**

****

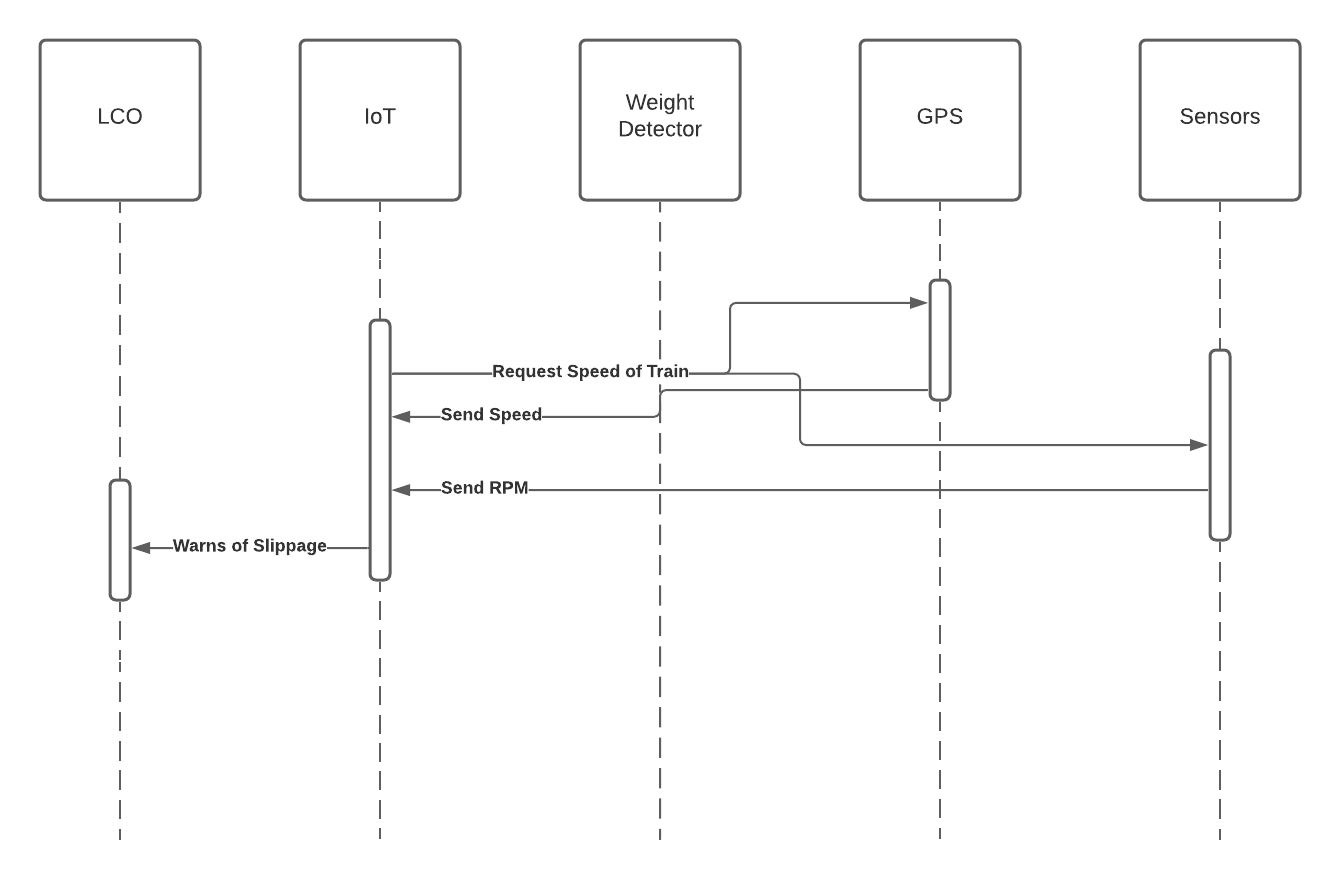
**Use Case 5**

****

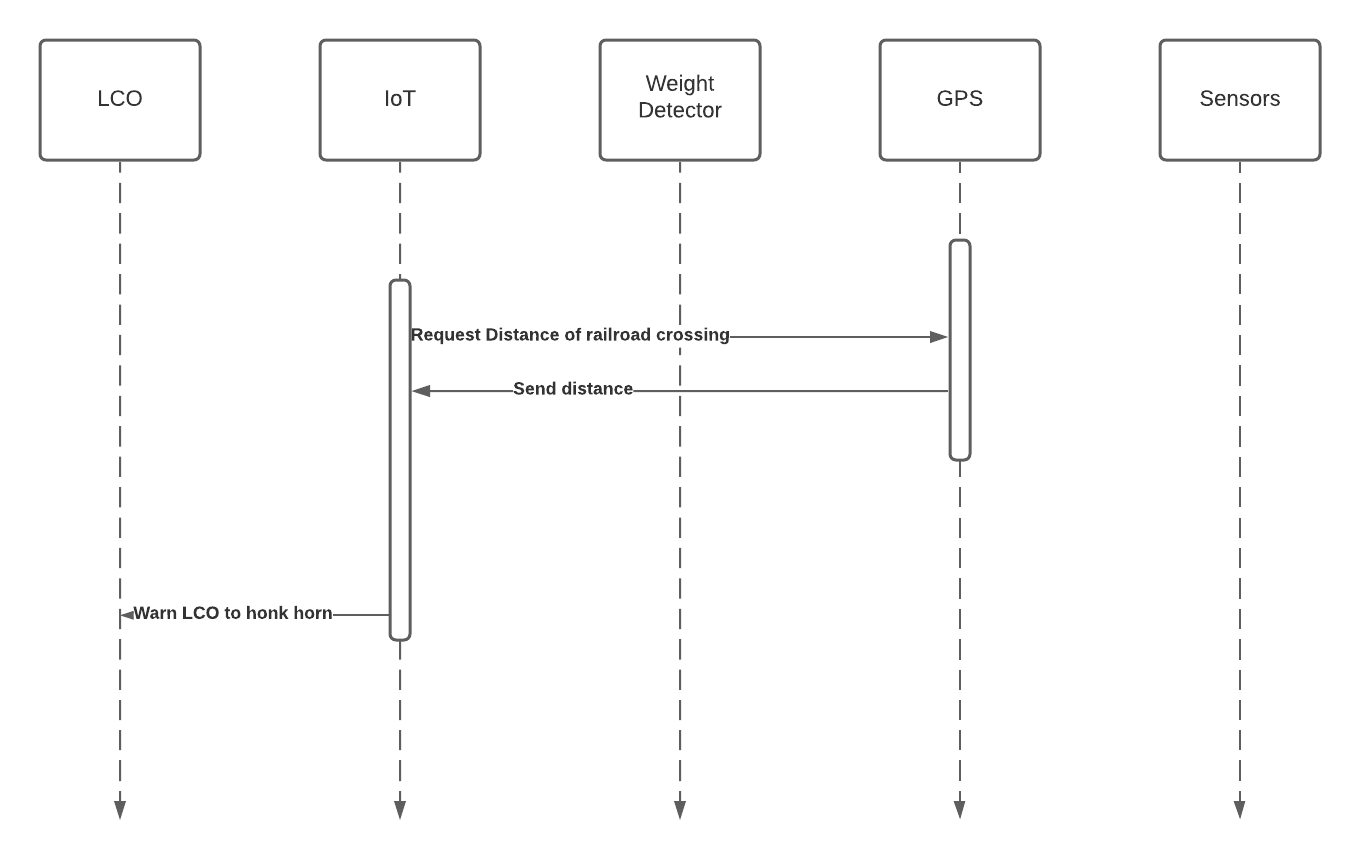
**Use Case 6**

****

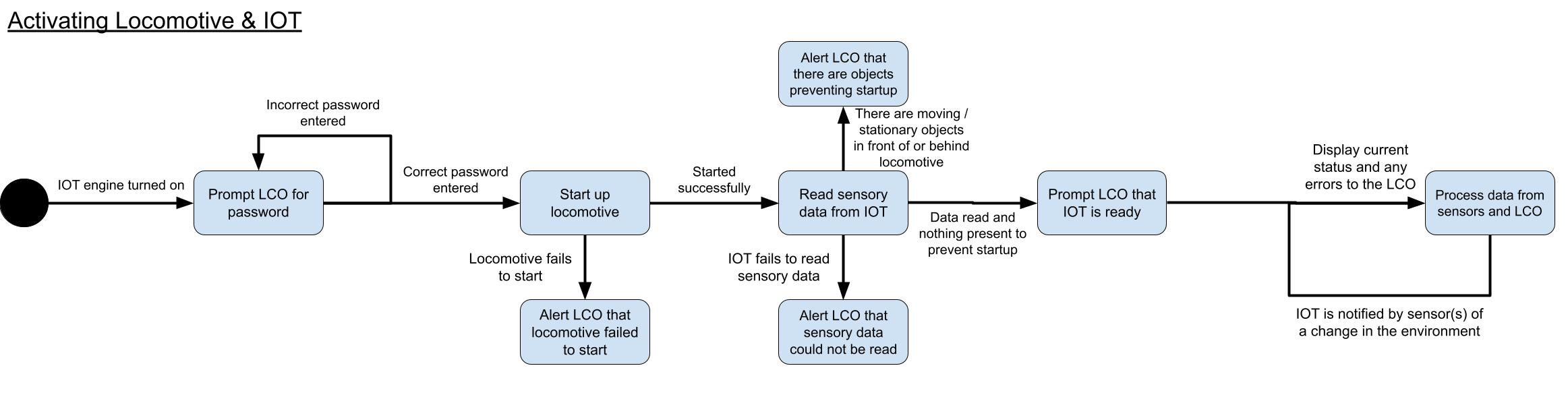
**Use Case 7**

****

**Use Case 8**



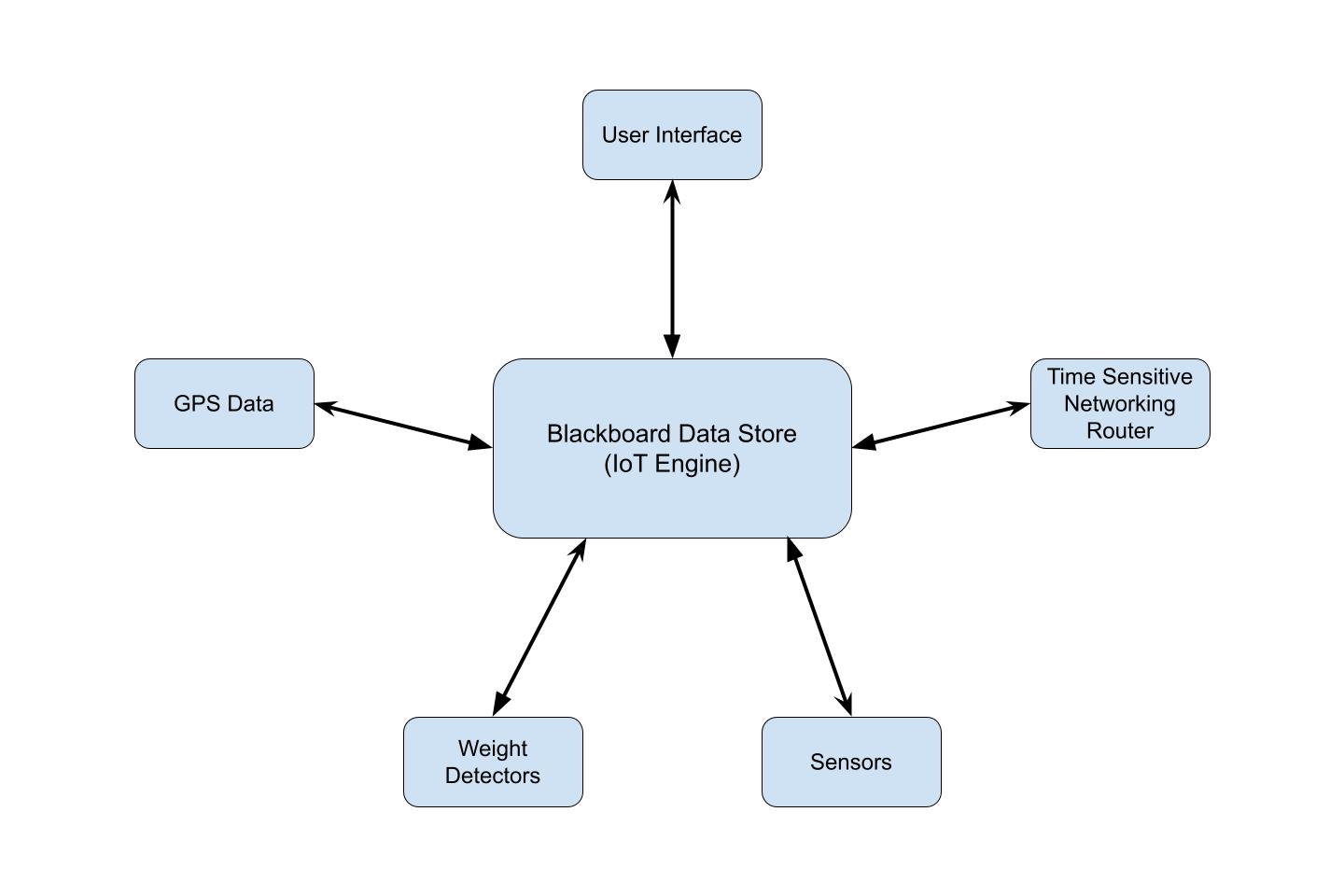
## 4.7 - State Diagram



# Section 5: Software Architecture

## 5.1 - Architectural Styles

### 5.1.1 - Data Centered Architecture

****

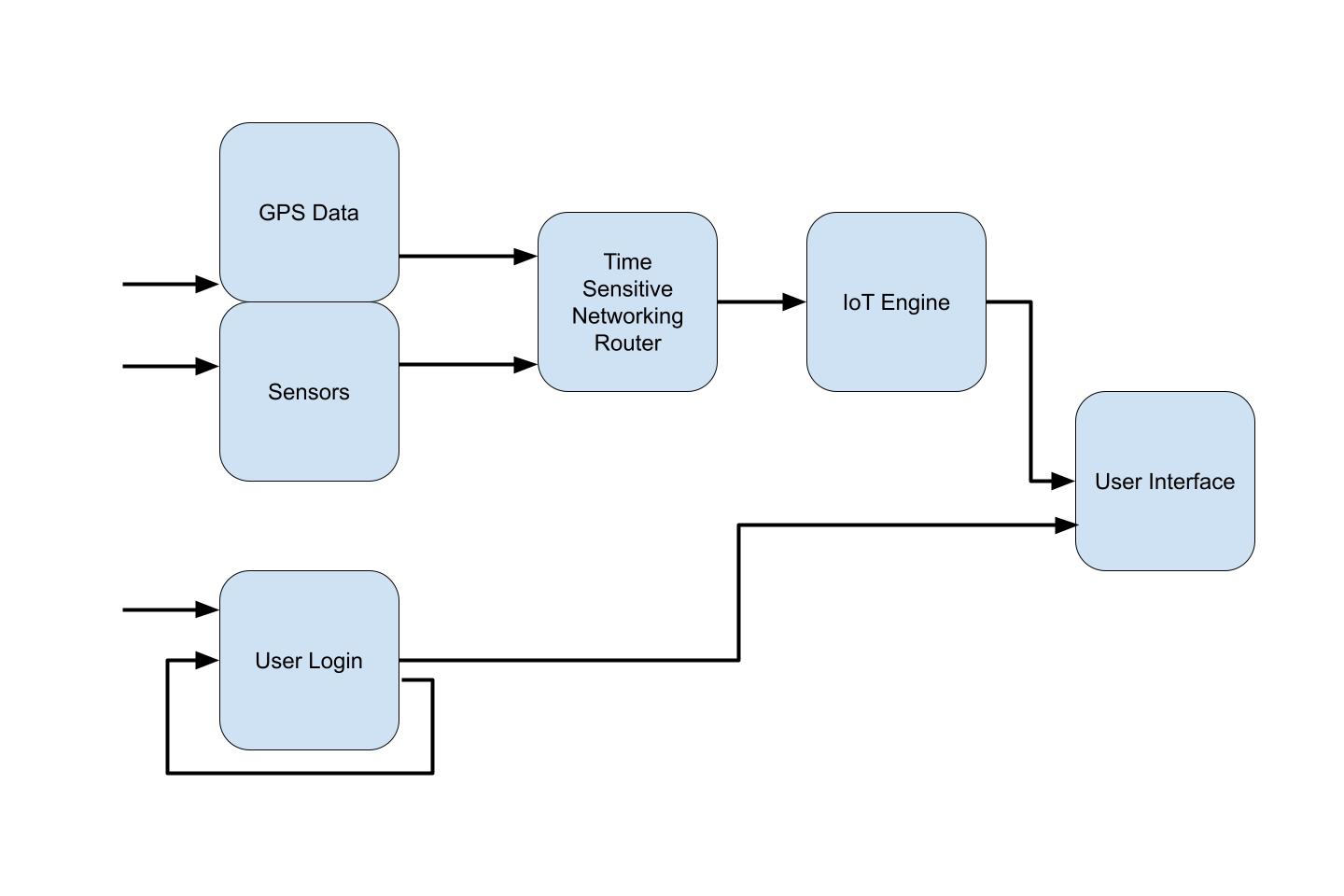
**Advantages**

* Easy to add new components and change existing ones.
* Can coordinate the transfer of data between components if a blackboard variation is used.
* Components are able to operate independently.

**Disadvantages**

* Cost and resource intensive.
* High dependency between data structure of data store and its agents.
* Changes in data structure highly affect the clients.

### 5.1.2 - Data Flow Architecture



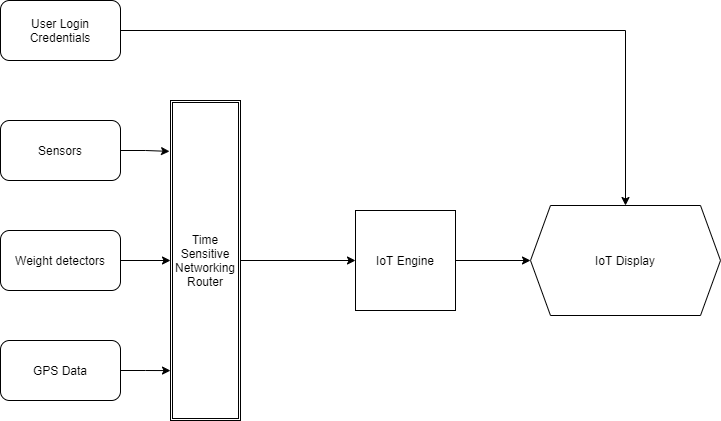
**Advantages**

* Pipe-filter provides concurrency and high throughput for excessive data processing.
* It simplifies the system maintenance and provides reusability.
* It has low coupling between filters and flexibility by supporting both sequential and parallel execution.

**Disadvantages**

* Pipe and filter are not suitable for dynamic interactions.
* Difficult to configure Pipe and Filter architecture dynamically.
* Takes a long time to create.

### 5.1.3 - Call Return Architecture

****

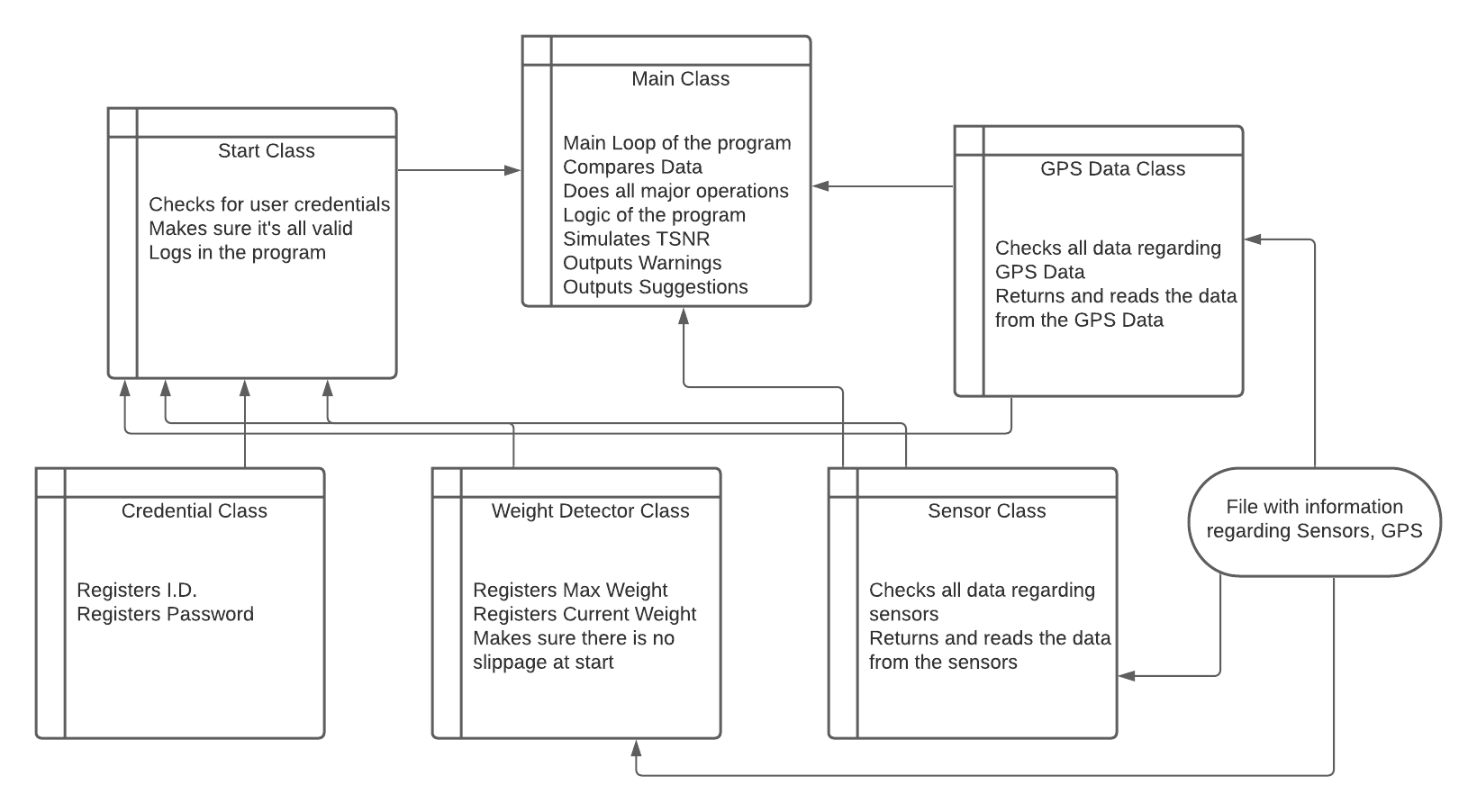
**Advantages**

* Easy to scale and modify.
* Easy to read and understand.
* Shows the control flow between components.

**Disadvantages**

* Correctness of one component depends on the correctness of the components whose service it relies on.
* Parallel Processing may be difficult to implement.
* Exceptions to normal operations may be difficult to handle.

### 5.1.4 - Object Oriented Architecture

****

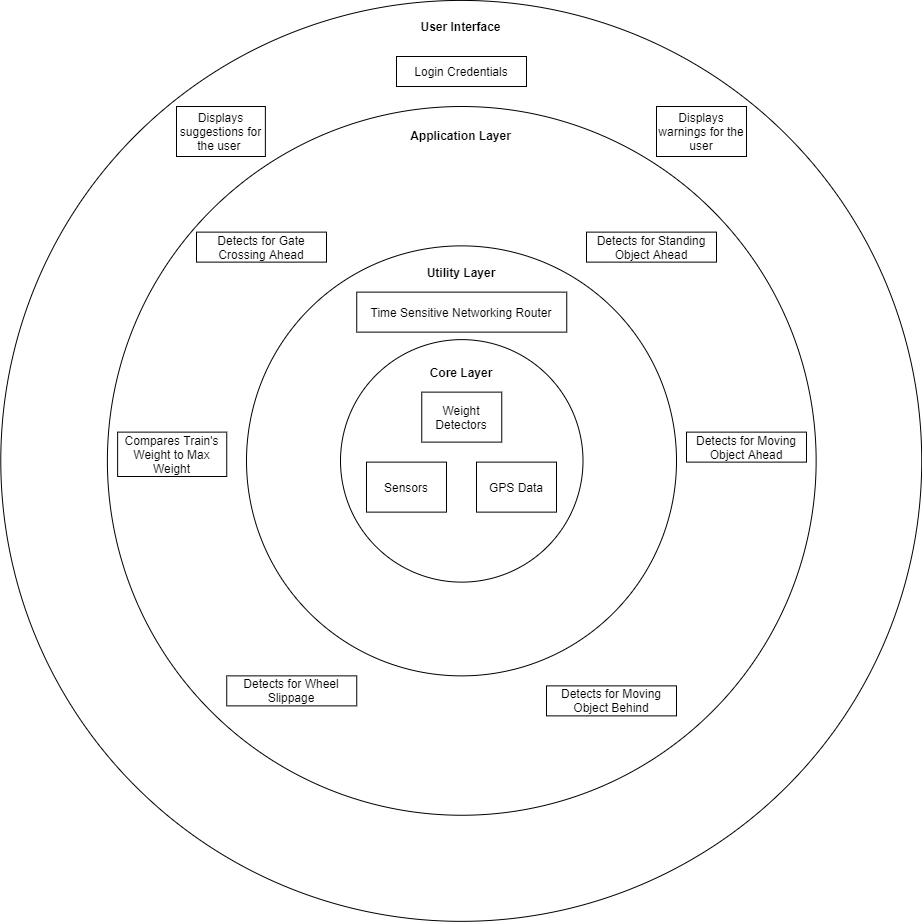
**Advantages**

* Highly scalable.
* Reliable and flexible architecture model.
* Maps to real world objects making it easy to understand.
* Often reusable due to polymorphism and abstraction.

**Disadvantages**

* Difficult to determine all necessary classes and objects.
* Difficult to enforce a set time and budget as the components are interdependent.
* Object Oriented Programs tend to be larger in size.

### 5.1.5 - Layered Architecture

****

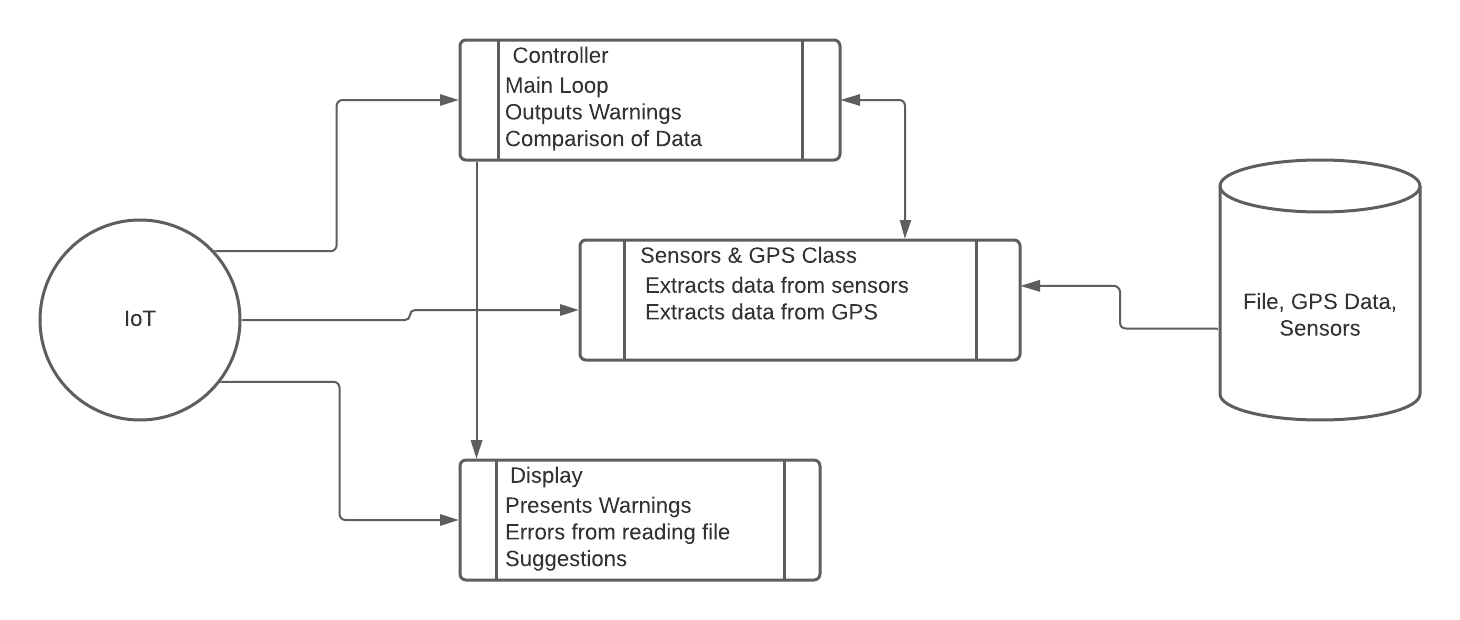
**Advantages**

* Allows for separation of concern because each layer is a distinct section of the project with a smaller scope which makes implementation much more straightforward.
* More testable because of the separate layers.
* Well-defined levels of abstraction.
* This architecture has high changeability because the changes in one layer will not affect the inner layers.
* Can replace inner layers as long as interfaces remain the same.
* There is the potential for adding layers.

**Disadvantages**

* Performance gets slower as more and more layers are added.
* Management cost goes up if there are too many layers.
* Leaky abstraction can disrupt the layered intent.

### 5.1.6 - Model View Controller Architecture

****

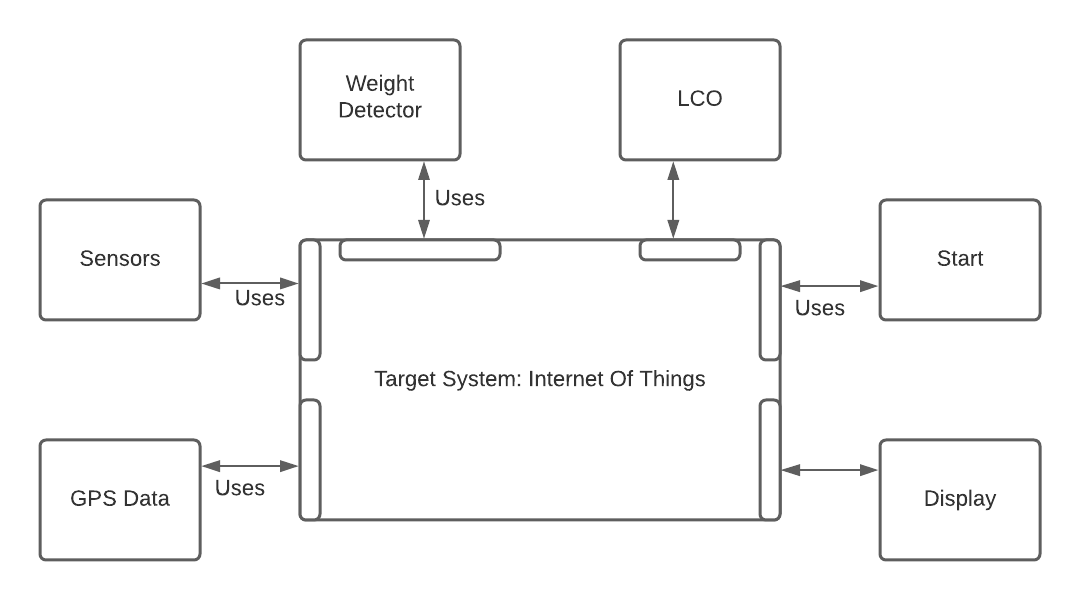
**Advantages**

* Development of the application becomes fast.
* Easy for multiple developers to collaborate and work together.
* Easier to Update the application.
* Easier to Debug as we have multiple levels properly written in the application.

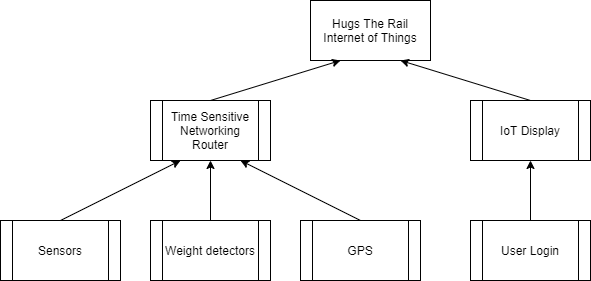
**Disadvantages**

* It is hard to understand the MVC architecture.
* Must have strict rules on methods.
* It is not suitable for small applications which has adverse effects on the application's performance and design.

## 5.2 - Representing the System in Context

****

## 5.3 - Components

****

## 5.4 - Decision: Object Oriented Architecture

Based on the pros and cons stated for each of these architectures, we decided to follow the structure of the Object Oriented Architecture. Not only is it familiar due to our knowledge of object oriented programming languages like Java, but the concepts apply to IoT since we can have multiple classes with multiple variables and functions that represent objects that have their own unique functions. For example, the Start Class could check that credentials are inputted correctly, and once they are inputted correctly the program will enter the Main Class. Its organization and familiarity will help in debugging the project as well. For these reasons, Object Oriented Architecture seemed best for our purposes.

### 5.4.1 - IoT as an Object Oriented Architecture (Detailed)

**Start Class:**

**Description:** The start class will “start” the program. It will check that everything has been read successfully. Works together with Credential Class and Weight Detector Class, as well as the Sensor Class and GPS Data Class.

**Variables:**

*Bool successful\_start*: Starts as ‘false’. This will be changed to true if credentials\_match returns true.

*Bool data\_sensors\_read*: Starts as ‘false’. This will be changed to true if the data from the sensors could be read. Depends on what Sensor Class methods return.

*Bool data\_GPS\_read*: Starts as ‘false’. This will be changed to true if the data from the GPS could be read. Depends on what GPS Data Class methods return.

**Methods:**

*Bool credentials\_match()*: Returns true of the input from LCO and the preprogrammed I.D. and password match.

*Bool check\_data()*: Returns true if sensor data could be found and read successfully. Checks that data\_sensors\_read was set to ‘true’.

*Bool check\_GPSdata()*: Returns true if GPS data could be found and read successfully. Checks that data\_GPS\_read was set to ‘true’.

*Bool check\_weight()*: Returns ‘true’ if Weight Detector class (the method ‘is\_cargo\_good()’) returns ‘true’, meaning that the weight is good enough for a successful start of the train.

*Void error\_at\_start()*: If any of the previous variables or methods return false, this will make sure to print to the console a Warning saying what went wrong. The way to know what went wrong is by checking for the return value of each method individually and outputting the result to the console.

**Credential Class:**

**Description:** Contains the credentials from the user.

**Variables:**

*String user\_I.D.*: The preprogrammed user I.D. for the LCO.

*String password*: The preprogrammed password for the LCO.

**Methods:**

*String get\_id()*: Returns the I.D.

*String get\_passwd()*: Returns the password.

**Weight Detector Class:**

**Description:** Checks that the current cargo of the train does not exceed its carrying capacity.

**Variables:**

*Long max\_weight*: This preassigned variable will be set beforehand according to how much weight the train can carry.

*Long current\_weight*: This variable contains how much weight the train currently is carrying.

**Methods:**

*Bool is\_cargo\_good()*: This will return ‘true’ if this condition is met: ‘current\_weight’ < ‘max\_weight’.

*Void read\_weight\_from\_file()*: This will update ‘current\_weight’ and set it to the current weight specified by the weight detector (the data read from the file at start)

**Sensor Class:**

**Description:** Returns true to update the variables in Start Class, allowing it to know whether data from file regarding sensor information could be read, and it also needs to send information to the Main Class about sensors.

**Variables:**

*Int standing\_objdist*: Distance regarding a standing object ahead of the train. It is set by set\_standingobjdist() function and if there are no objects it just contains the value of zero.

*Int movobj\_speed\_ahead*: Speed of moving object ahead of the train.

*Int movobj\_speed\_behind*: Speed of moving object behind the train.

**Methods:**

*Bool data\_read()*: If a file with data from sensors could be opened and read return ‘true’.

*Bool set\_standingobjdist()*: Returns true if we could successfully set the distance of the standing object ahead read from the file. Note: having no object in front also returns true. In fact, this function always returns true.

*Bool get\_standingobjdist()*: Returns ‘standing\_objdist’.

*Bool set\_speedobj\_ahead()*: Returns ‘true’ if we could set ‘movobj\_speed\_ahead’ equal to the speed provided by the file.

*Bool set\_speedobj\_behind()*: Returns ‘true’ if we could set ‘movobj\_speed\_behind’ equal to the speed provided by the file.

*Int get\_speed\_ahead()*: Returns ‘movobj\_speed\_ahead’.

*Int get\_speed\_behind()*: Returns ‘movobj\_speed\_behind’.

*Int get\_RPM()*: Returns the rotations per min of the wheels as read from the file.

**GPS Data Class:**

**Description:** Needs to update the variables in Start Class, allowing it to know whether data from the file regarding GPS information could be read, and it also needs to send information to the Main Class about GPS data.

**Variables:**

*Int speed\_of\_train*: Contains the current speed of the train.

*Bool railroad\_function*: Contains the functioning status of the railroad crossing. If it’s closing, it’s set to ‘true’ by set\_railroad\_status().

**Methods:**

*Bool GPS\_read()*: If a file with data from GPS could be opened and read return ‘true’.

*Bool set\_speed\_train()*: Returns ‘true’ if we could set ‘speed\_of\_train’ to be equal to the speed of the train provided in the file read.

*Int get\_train\_speed()*: Returns ‘speed\_of\_train’.

*Void set\_railroad\_status()*: Changes railroad\_function to ‘true’ if data from the file regarding the status of the railroad was successfully retrieved.

*Bool get\_gatecross\_status()*: Returns ‘railroad\_function’.

*Int dist\_railroad()*: Returns -1 if there is no railroad crossing within a mile. But if there is a railroad crossing within a mile, it returns the distance.

**Main Class:**

**Description:** A loop that will continuously read from the files the information regarding sensors, and GPS. It works together with GPS Data Class and Sensor Class.

**Body:**

While true:

Int there\_are\_problems = 0

If (get\_standingobjdist() < 1 mile) {

print(Warning regarding standing object)

there\_are\_problems++

}

If (get\_speed\_ahead() < get\_train\_speed()) {

print(Warning regarding moving object ahead)

there\_are\_problems++

}

If (get\_speed\_behind() > get\_train\_speed()) {

print(Warning regarding moving object behind)

there\_are\_problems++

}

If (dist\_railroad() > 0) {

print(Warning: there is a railroad within a mile)

If (get\_gatecross\_status() == false) {

print(Warning: Railroad is not closing properly)

}

there\_are\_problems++

}

If (train arrived at railroad) {

print(Warning: You should honk the horn again for 5 seconds)

there\_are\_problems++

}

If (get\_train\_speed() != get\_RPM()) {

print(Warning: Wheels are slipping)

there\_are\_problems++

}

If (there\_are\_problems == 0) {

print(No problems at the moment)

}

sleep(5 seconds)

# Section 6: Project Code

|  |
| --- |
| public class Start\_Class {   static boolean data\_read = false;   // Check that credentials match  public static boolean credentials\_match(String input\_ID, String input\_passwd) {  return (input\_ID.equals(Credential\_Class.get\_ID())) &&   (input\_passwd.equals(Credential\_Class.get\_passwd()));  }   // Check that sensor file exists  public static boolean check\_data(String path) {  data\_read = Sensor\_Class.data\_read(path);  return data\_read;  }   // If there is an object standing ahead of the train within 0.3 miles, then return TRUE  public static boolean standing\_obj\_at\_start(double distance) {  return (distance < 0.5);  }   // If there is a moving object ahead, just return whether it's going above 125 mph, because that means  // It's above our normal speed for when we start.  public static boolean moving\_obj\_at\_start(double speed) {  return (speed < 125.0);  }     } |

|  |
| --- |
| public class Credential\_Class {    // Preprogrammed Password and ID for the user  private static String user\_ID = "HTR USER";  private static String password = "HTR PASSWORD";    public static String get\_ID() {  return user\_ID;  }    public static String get\_passwd() {  return password;  }   } |

|  |
| --- |
| public class Weight\_Detector\_Class {    // Max cargo weight train can carry in lbs.  private static double MAX\_WEIGHT = 286000.0;    public static boolean is\_cargo\_good(double cargo\_weight) {  return (cargo\_weight < MAX\_WEIGHT);  }   } |

|  |
| --- |
| import java.io.File;  public class Sensor\_Class {   public static double standing\_objdist;  public static double movobj\_speed\_ahead;  public static double movobj\_speed\_behind;  public static int rpm;   // Checks that a file is given and we can read from it  public static boolean data\_read(String path) {    File INPUT = new File(path);  return (INPUT.exists());  }   // Convert rpm to miles per hour  public static double rpmToMph(int rotationsPerMin) {  double circumference = 0.003;  return (circumference \* rotationsPerMin \* 60);  }   } |

|  |
| --- |
| public class GPS\_Data\_Class {  public static double railroad\_dist;  public static int railroad\_func\_status;  public static double train\_speed;  } |

|  |
| --- |
| import java.awt.Color; import javax.swing.\*; import java.util.Scanner; import java.io.\*; import java.awt.event.\*; import javax.sound.sampled.\*;   public class Main implements ActionListener {    // Here we have the path of error image. Change accordingly to run on a different system.  private static String warning\_icon = "C:\\Users\\rafas\\eclipse-workspace\\HTR\\error.png";  private static String check\_icon = "C:\\Users\\rafas\\eclipse-workspace\\HTR\\green\_check.png";  private static String standing\_icon = "C:\\Users\\rafas\\eclipse-workspace\\HTR\\standing\_obj.png";  private static String movobjFront\_icon = "C:\\Users\\rafas\\eclipse-workspace\\HTR\\objFront.png";  private static String movobjBack\_icon = "C:\\Users\\rafas\\eclipse-workspace\\HTR\\objBack.png";  private static String rail\_icon = "C:\\Users\\rafas\\eclipse-workspace\\HTR\\rail\_ahead.png";  private static String rail\_open\_icon = "C:\\Users\\rafas\\eclipse-workspace\\HTR\\rail.png";  private static String sec5\_icon = "C:\\Users\\rafas\\eclipse-workspace\\HTR\\5sec.png";  private static String wheels\_icon = "C:\\Users\\rafas\\eclipse-workspace\\HTR\\wheels.png";  private static String music\_path = "C:\\Users\\rafas\\eclipse-workspace\\HTR\\IoTTheme.wav";  private static String data\_path;  private static JPanel panel;  private static JLabel userLabel;   private static JTextField userText;  private static JLabel pwdLabel;  private static JPasswordField pwdText;  private static JLabel error\_on\_login;  private static JFrame frame;  private static JButton loginbutton;  private static JButton button1;  private static JButton buttonLoop;  private static Scanner reader;  private static int timeElapsed = 0;    public static void main\_body\_loop() {    //Enter the main body  if (reader.hasNext()) {  panel.removeAll();      boolean no\_problems = true;   Sensor\_Class.standing\_objdist = Double.parseDouble(reader.next());  Sensor\_Class.movobj\_speed\_ahead = Double.parseDouble(reader.next());  Sensor\_Class.movobj\_speed\_behind = Double.parseDouble(reader.next());  GPS\_Data\_Class.railroad\_dist = Double.parseDouble(reader.next());  GPS\_Data\_Class.railroad\_func\_status = Integer.parseInt(reader.next());  GPS\_Data\_Class.train\_speed = Double.parseDouble(reader.next());  Sensor\_Class.rpm = Integer.parseInt(reader.next());    timeElapsed++;  String time = "Time elapsed: " + String.valueOf(timeElapsed);  JLabel timeLabel = new JLabel(time);  timeLabel.setBounds(1280/2 - 60 - 10, 30,450,100);  panel.add(timeLabel);    // Next we are going to check each of them  if (Sensor\_Class.standing\_objdist < 1.0) {  JLabel standing\_objLabel = new JLabel("Warning: There is a standing object ahead of the train.");  standing\_objLabel.setBounds(20+10, 120, 500, 25);  panel.add(standing\_objLabel);    JLabel labelStand=new JLabel();  labelStand.setIcon(new ImageIcon(standing\_icon));  labelStand.setBounds(20+10+20+35, 120, 200, 200);  panel.add(labelStand);    no\_problems = false;  }   if (Sensor\_Class.movobj\_speed\_ahead < GPS\_Data\_Class.train\_speed) {  JLabel moving\_objFront = new JLabel("Warning: There is a moving object ahead of the train.");  moving\_objFront.setBounds(1280/2 - 200 +10, 120, 500, 25);  panel.add(moving\_objFront);    JLabel labelMov=new JLabel();  labelMov.setIcon(new ImageIcon(movobjFront\_icon));  labelMov.setBounds(1280/2 - 200 +10 + 20+10+20+35, 125, 200, 200);  panel.add(labelMov);    no\_problems = false;  }   if (Sensor\_Class.movobj\_speed\_behind > GPS\_Data\_Class.train\_speed) {  JLabel moving\_objBehind = new JLabel("Warning: There is a moving object behind the train.");  moving\_objBehind.setBounds(1280/2 + 200 + 10, 120, 500, 25);  panel.add(moving\_objBehind);    JLabel labelMov2=new JLabel();  labelMov2.setIcon(new ImageIcon(movobjBack\_icon));  labelMov2.setBounds(1280/2 + 200 + 10 + 20+10+20+35, 125, 200, 200);  panel.add(labelMov2);    no\_problems = false;  }   if (GPS\_Data\_Class.railroad\_dist < 1.0 && GPS\_Data\_Class.railroad\_dist > 0.0) {  JLabel railroad\_obj = new JLabel("Warning: There is a railroad crossing ahead. Honk horn for:");  railroad\_obj.setBounds(20+10, 120 + 200, 500, 25);  panel.add(railroad\_obj);    JLabel labelRail=new JLabel();  labelRail.setIcon(new ImageIcon(rail\_icon));  labelRail.setBounds(10+20+35+20, 120 + 200 + 30, 200, 200);  panel.add(labelRail);    no\_problems = false;  }    if (GPS\_Data\_Class.railroad\_dist <= 0.0) {  JLabel honk = new JLabel("Warning: You arrived at a railroad crossing. Honk horn for:");  honk.setBounds(20+10, 120 + 200, 500, 25);  panel.add(honk);    JLabel label5=new JLabel();  label5.setIcon(new ImageIcon(sec5\_icon));  label5.setBounds(10+20+35+20, 120 + 200 + 30, 200, 200);  panel.add(label5);    no\_problems = false;  }   if (GPS\_Data\_Class.railroad\_func\_status == 0) {  JLabel rail\_not\_working = new JLabel("Warning: The railroad crossing ahead is not closing.");  rail\_not\_working.setBounds(1280/2 - 200 +10, 120 + 200, 500, 25);  panel.add(rail\_not\_working);    JLabel labelRailOpen=new JLabel();  labelRailOpen.setIcon(new ImageIcon(rail\_open\_icon));  labelRailOpen.setBounds(1280/2 - 200 +10+ 10+20+35, 120 + 200 + 30, 200, 200);  panel.add(labelRailOpen);    no\_problems = false;  }    if (Math.abs(GPS\_Data\_Class.train\_speed - Sensor\_Class.rpmToMph(Sensor\_Class.rpm)) > 0.1) {  JLabel wheel\_slip = new JLabel("Warning: Wheels are slipping.");  wheel\_slip.setBounds(1280/2 + 200 + 10, 120 + 200, 500, 25);  panel.add(wheel\_slip);    JLabel labelWheel=new JLabel();  labelWheel.setIcon(new ImageIcon(wheels\_icon));  labelWheel.setBounds(1280/2 + 200 + 10+ 10+20+35, 120 + 200 + 30, 200, 200);  panel.add(labelWheel);    no\_problems = false;  }   if (no\_problems) {  JLabel labelcheck=new JLabel();  labelcheck.setIcon(new ImageIcon(check\_icon));  labelcheck.setBounds(1280/2 - 200 -20, 150,450,400);  panel.add(labelcheck);  no\_problems = false;  }    buttonLoop = new JButton("Continue");  buttonLoop.setBounds(1280/2 - 110, 600, 150, 25);  buttonLoop.addActionListener(new Main());  panel.add(buttonLoop);    frame.repaint();  // Simulate the Time Sensitive Networking Router  //TimeUnit.SECONDS.sleep(10);  } else {  panel.removeAll();  JLabel thank\_you = new JLabel("IoT finished reading data. Thank you for choosing HTR IoT!");  thank\_you.setBounds(1280/2 - 150 -65, 200, 80\*2\*2\*2, 25\*2);  panel.add(thank\_you);  frame.repaint();  }    }    public static void check\_at\_start() {  if (!Start\_Class.check\_data(data\_path)) {  JLabel error\_reading = new JLabel("Error: Data from sensors and/or GPS could not be read correctly.");  error\_reading.setBounds(1280/2 - 150 -65, 200, 80\*2\*2\*2, 25\*2);  panel.add(error\_reading);      JLabel label5=new JLabel();  label5.setIcon(new ImageIcon(warning\_icon));  label5.setBounds(1280/2 - 90, 250,450,100);  panel.add(label5);    frame.repaint();  } else {    boolean no\_warnings = true;  // And now check for the distance of the standing object at start  Sensor\_Class.standing\_objdist = Double.parseDouble(reader.next());  if (Start\_Class.standing\_obj\_at\_start(Sensor\_Class.standing\_objdist)) {  JLabel standing\_start = new JLabel("Warning at start: There is a standing object ahead of the train.");  standing\_start.setBounds(1280/2 - 160, 180, 500, 25);  panel.add(standing\_start);  no\_warnings = false;  }   Sensor\_Class.movobj\_speed\_ahead = Double.parseDouble(reader.next());  if (Start\_Class.moving\_obj\_at\_start(Sensor\_Class.movobj\_speed\_ahead)) {  JLabel moving\_start = new JLabel("Warning at start: There is a moving object ahead of the train.");  moving\_start.setBounds(1280/2 - 160, 250, 500, 25);  panel.add(moving\_start);  no\_warnings = false;  }   double current\_cargo = Double.parseDouble(reader.next());  if (!Weight\_Detector\_Class.is\_cargo\_good(current\_cargo)) {  JLabel cargo\_start = new JLabel("Warning at start: There is too much cargo on the train.");  cargo\_start.setBounds(1280/2 - 160, 320, 500, 25);  panel.add(cargo\_start);  no\_warnings = false;  }    if (no\_warnings) {  JLabel good = new JLabel("No problems starting IoT.");  good.setBounds(1280/2 - 100, 200, 80\*2, 25\*2);  panel.add(good);  } else {  JLabel labelEr=new JLabel();  labelEr.setIcon(new ImageIcon(warning\_icon));  labelEr.setBounds(1280/2 - 60, 50,450,100);  panel.add(labelEr);  }    button1 = new JButton("Continue");  button1.setBounds(1280/2 - 100, 400, 80\*2, 25\*2);  button1.addActionListener(new Main());  panel.add(button1);    frame.repaint();    }  }    // The program will run through here  public static void main(String args[]) {   if (args.length != 1) {  System.out.println("Usage: java Main.java <FILE PATH>");  return;  }  data\_path = args[0];  Sensor\_Class.data\_read(data\_path);    File file = new File(music\_path);  try {  AudioInputStream audioStream = AudioSystem.getAudioInputStream(file);  Clip clip = AudioSystem.getClip();  clip.open(audioStream);  clip.start();  } catch (Exception t) {  System.out.println("Exception: Couldn't play audio.");  }    // Set up the GUI  panel = new JPanel();  frame = new JFrame();  frame.setSize(1280, 720);  frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);  frame.setTitle("HTR IoT!");  frame.add(panel);  frame.setVisible(true);    panel.setLayout(null);  panel.setBackground(new Color(230, 255, 255));    userLabel = new JLabel("User");  userLabel.setBounds(1280/2 - 150 -100, 200, 80\*2, 25\*2);  panel.add(userLabel);    userText = new JTextField();  userText.setBounds(1280/2 - 50 -100, 200, 165\*2, 25\*2);  panel.add(userText);    pwdLabel = new JLabel("Password");  pwdLabel.setBounds(1280/2 - 150 -100, 300, 80\*2, 25\*2);  panel.add(pwdLabel);    pwdText = new JPasswordField();  pwdText.setBounds(1280/2 - 50 -100, 300, 165\*2, 25\*2);  panel.add(pwdText);    error\_on\_login = new JLabel("");  error\_on\_login.setBounds(1280/2 - 150 -100, 500, 165\*2, 25\*2);  panel.add(error\_on\_login);    loginbutton = new JButton("Login");  loginbutton.setBounds(1280/2 - 150 -100, 400, 165\*2 + 100, 25\*2);  loginbutton.addActionListener(new Main());  panel.add(loginbutton);    frame.repaint();  }    // This code runs if you click the button  @Override  public void actionPerformed(ActionEvent e) {  if (e.getSource() == loginbutton) {  String user = userText.getText();  String password = pwdText.getText();  try {  reader = new Scanner(new File(data\_path));  } catch (Exception E) {  System.out.println("Exception");  }  if (Start\_Class.credentials\_match(user, password)) {  panel.removeAll();  check\_at\_start();  } else {  error\_on\_login.setText("Error: Invalid User Id or Password");  }  } else if (e.getSource() == button1) {  main\_body\_loop();  } else if (e.getSource() == buttonLoop) {  main\_body\_loop();  }    } } |

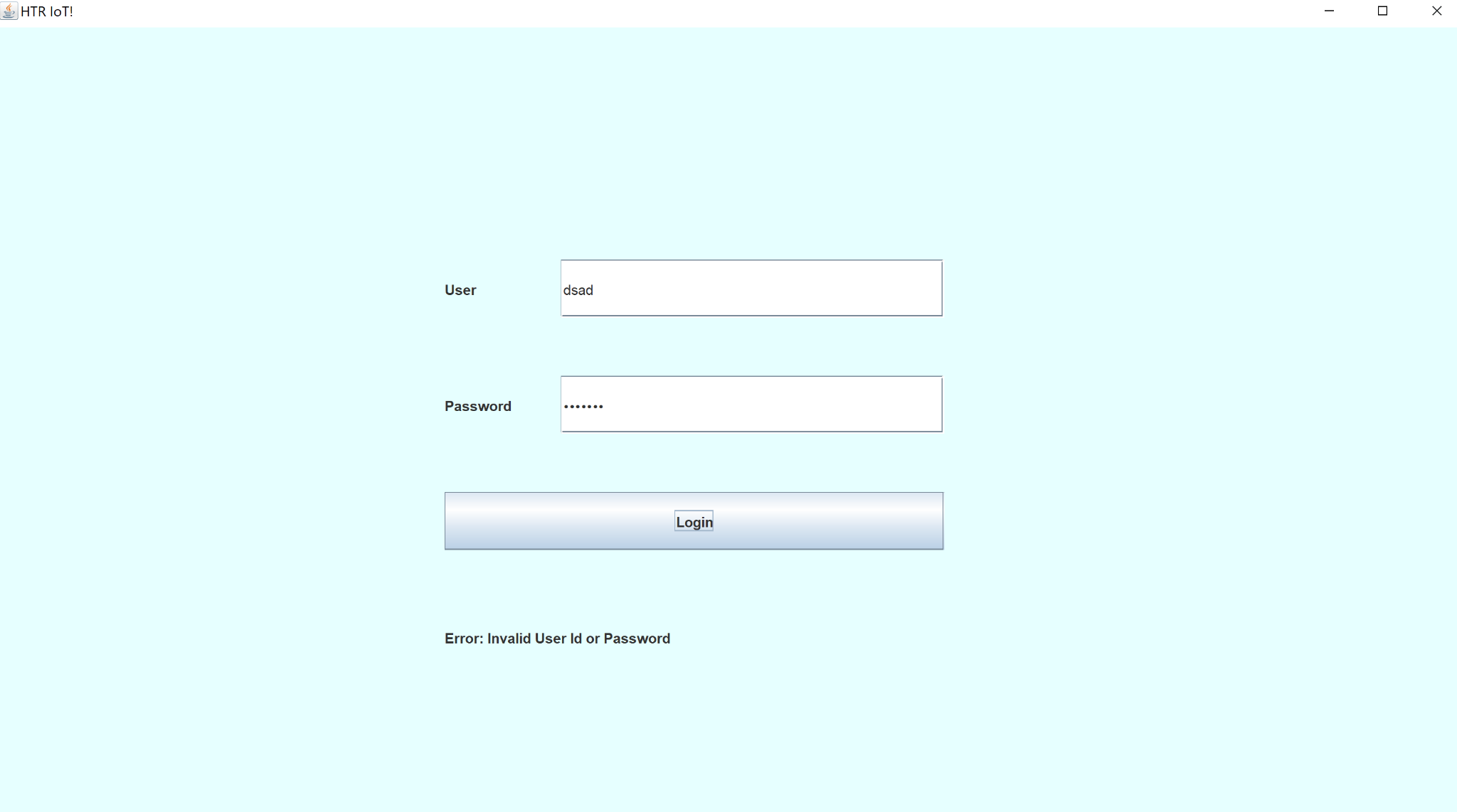
# 

# Section 7: Testing

## 7.1 - Scenario Based Testing

### 7.1.1 - Testing Use Case 1

**Test:** Upon opening IoT, the “User” field and “Password” field should be inputted a random username and password. The program is expected to inform the user that they have inputted a wrong credential.



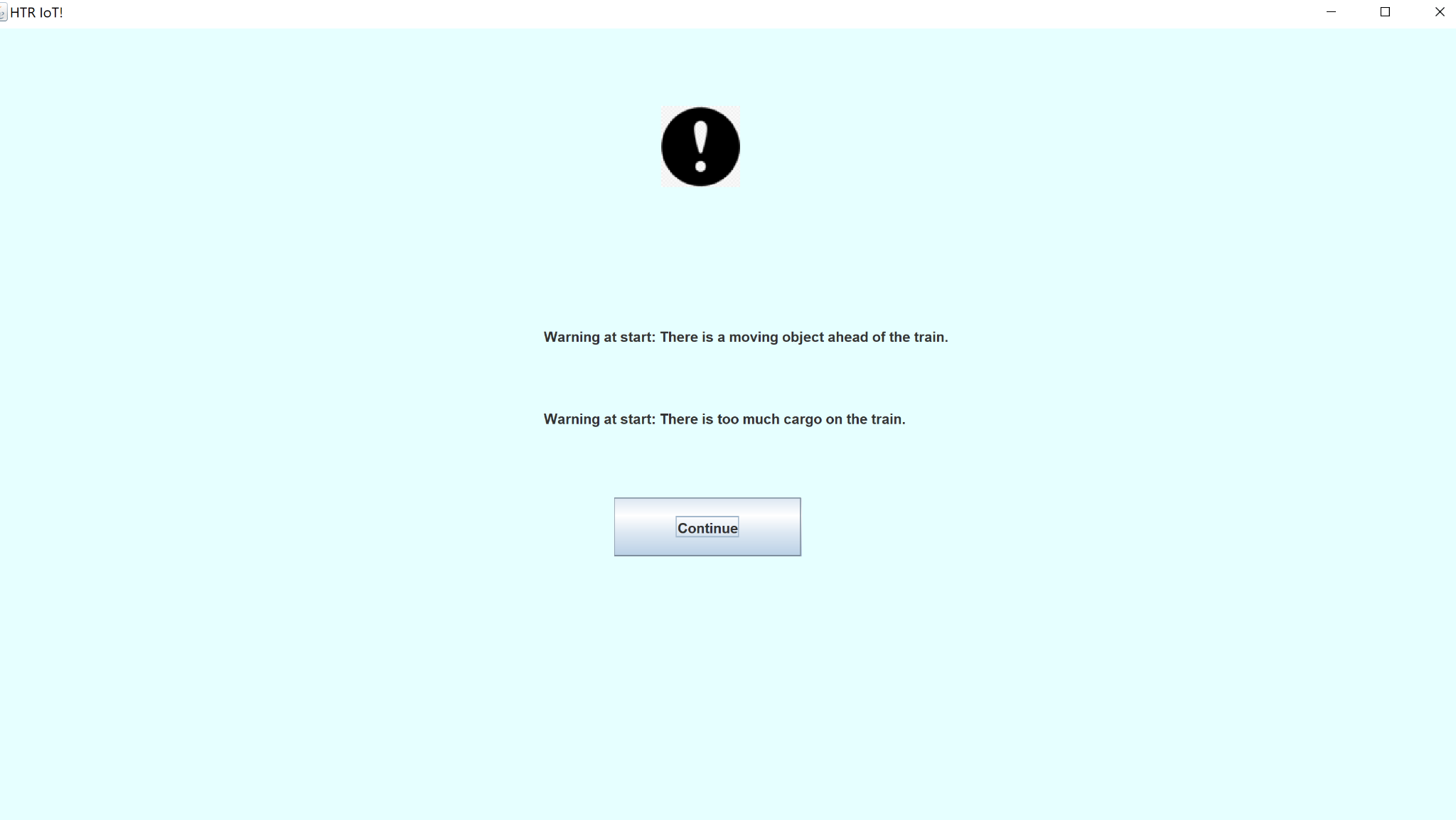
**Outcome:** Program outputs “Error: Invalid User Id or Password”

**Result:** PASSED

### 

### 7.1.2 - Testing Use Case 2

**Test:** To check if IoT has roughly equal power-to-weight ratio, the data given should contain a bigger cargo than the suggested cargo for the train. In that case, the program is expected to warn the LCO.



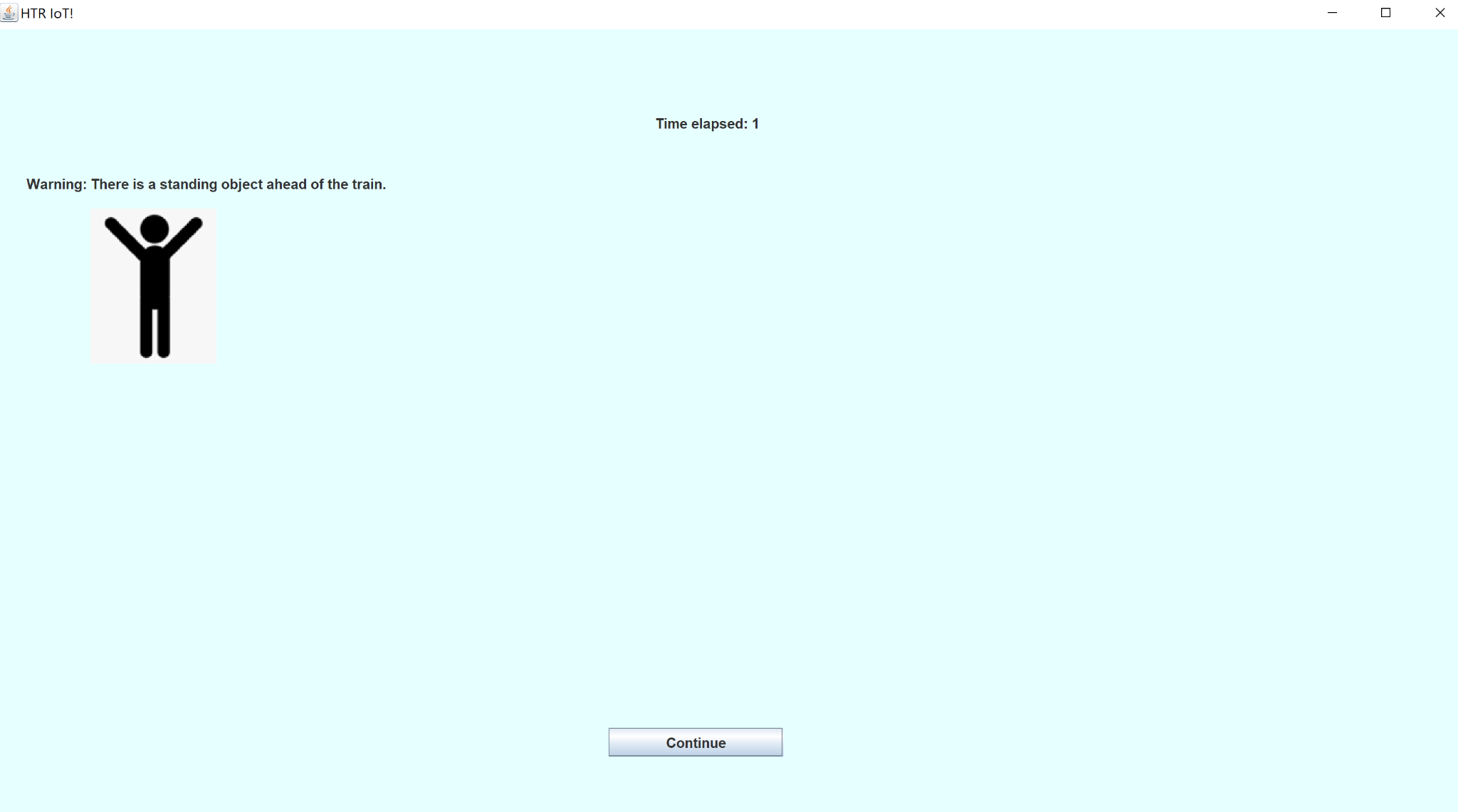
## 

**Outcome:** Program outputs “Warning at start: There is too much cargo on the train.”

**Result:** PASSED

### 7.1.3 - Testing Use Case 3

**Test:** IoT should check through the sensors if there is an object standing ahead in its path. In that case, warn the LCO.



**Outcome:** Program outputs “Warning: There is a standing object ahead of the train” along with an icon.

**Result:** PASSED

### 7.1.4 - Testing Use Case 4

**Test:** IoT should check through the sensors if there is an object moving ahead in its path. In that case, warn the LCO.

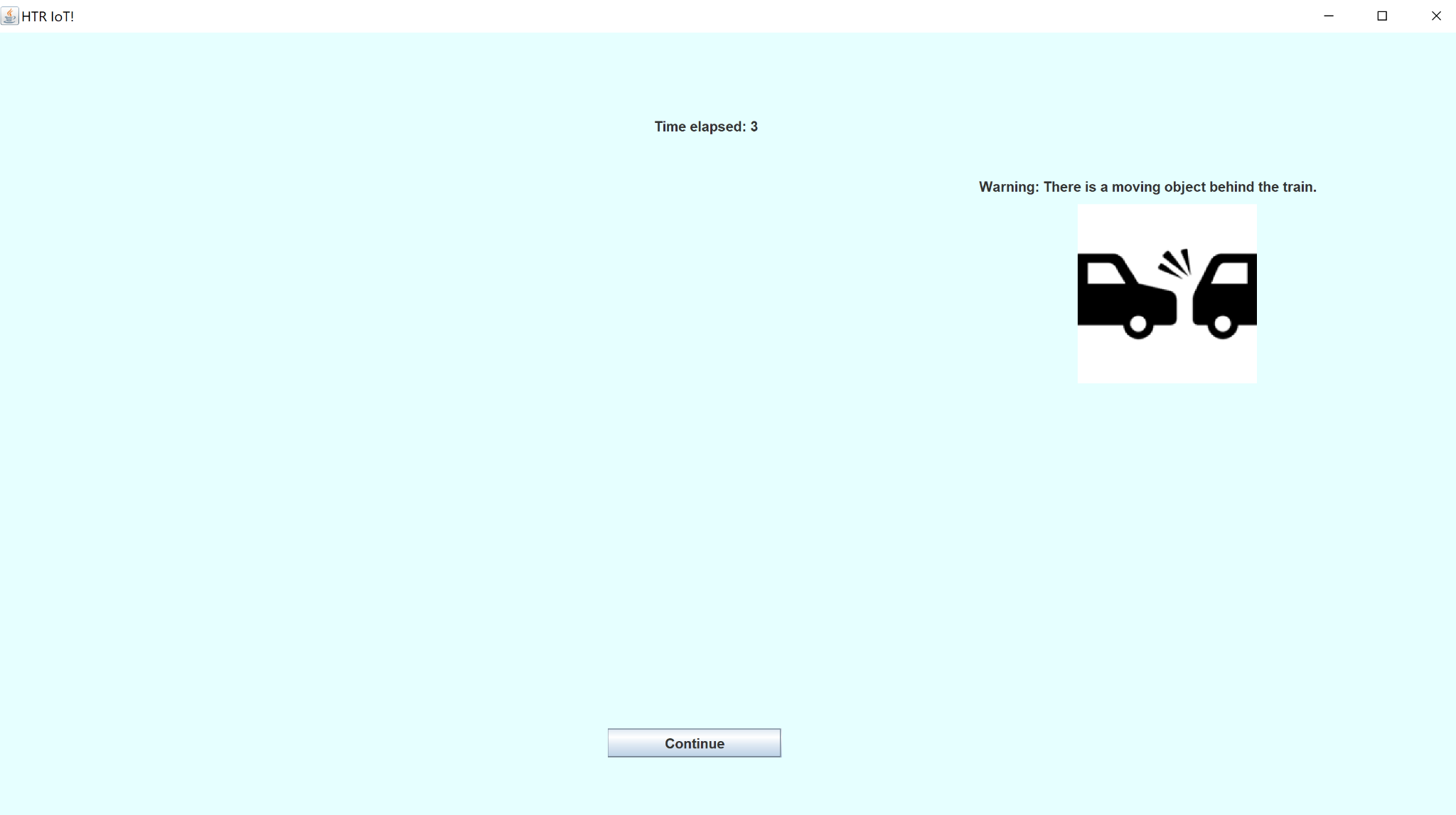


**Outcome:** Program outputs “Warning: There is a moving object ahead of the train.” along with an icon.

**Result:** PASSED

### 7.1.5 - Testing Use Case 5

**Test:** IoT should check through the sensors if there is an object moving behind in its path. In that case, warn the LCO.

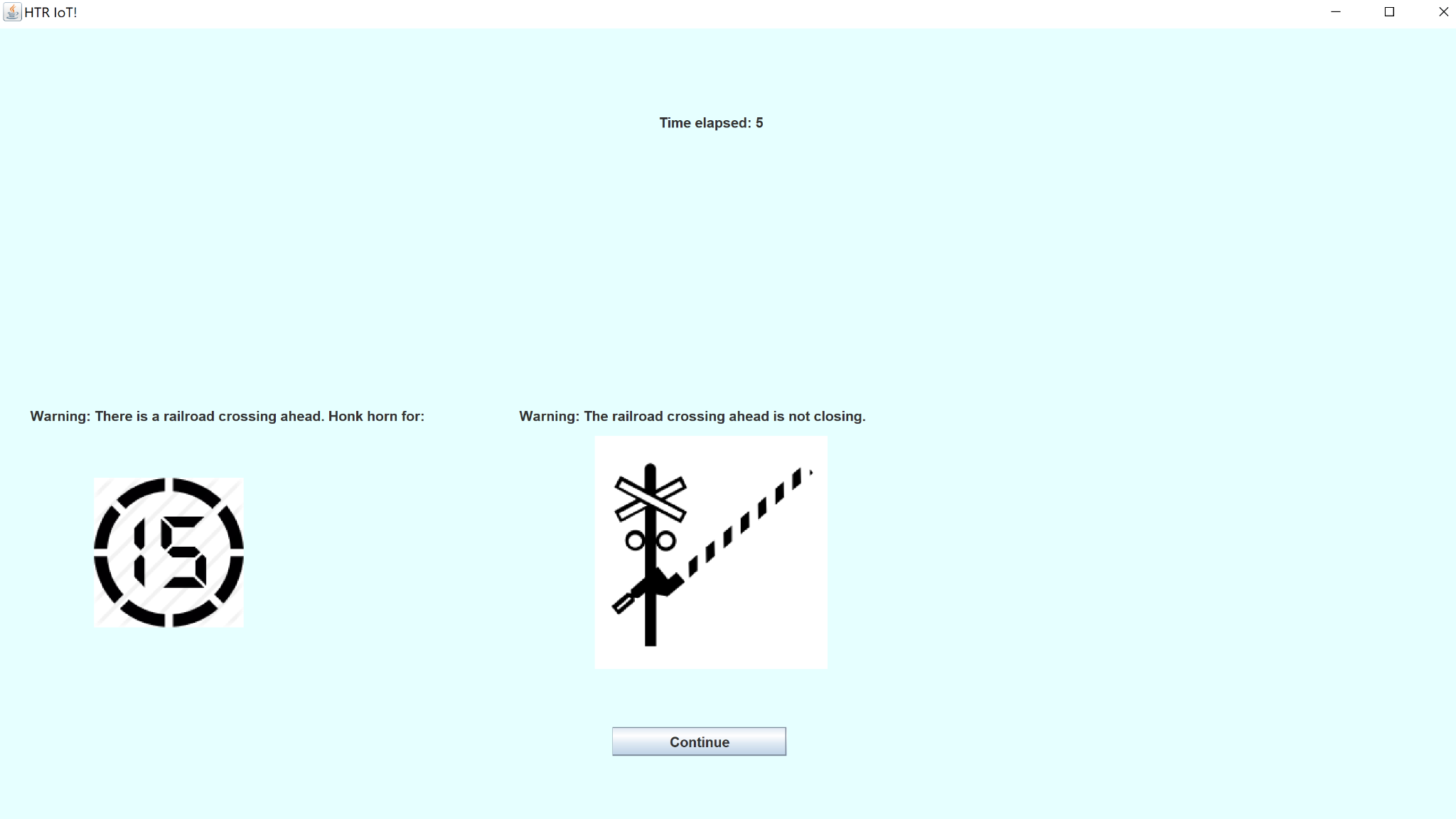


**Outcome:** Program outputs “Warning: There is a moving object behind the train.” along with an icon.

**Result:** PASSED

### 7.1.6 - Testing Use Case 6

**Test:** IoT should check if there is a gate crossing within a mile ahead of the train. If there is one, warn the LCO to honk the horn for 15 seconds. On top of that, if the railroad crossing ahead is not functioning correctly, also warn the LCO.

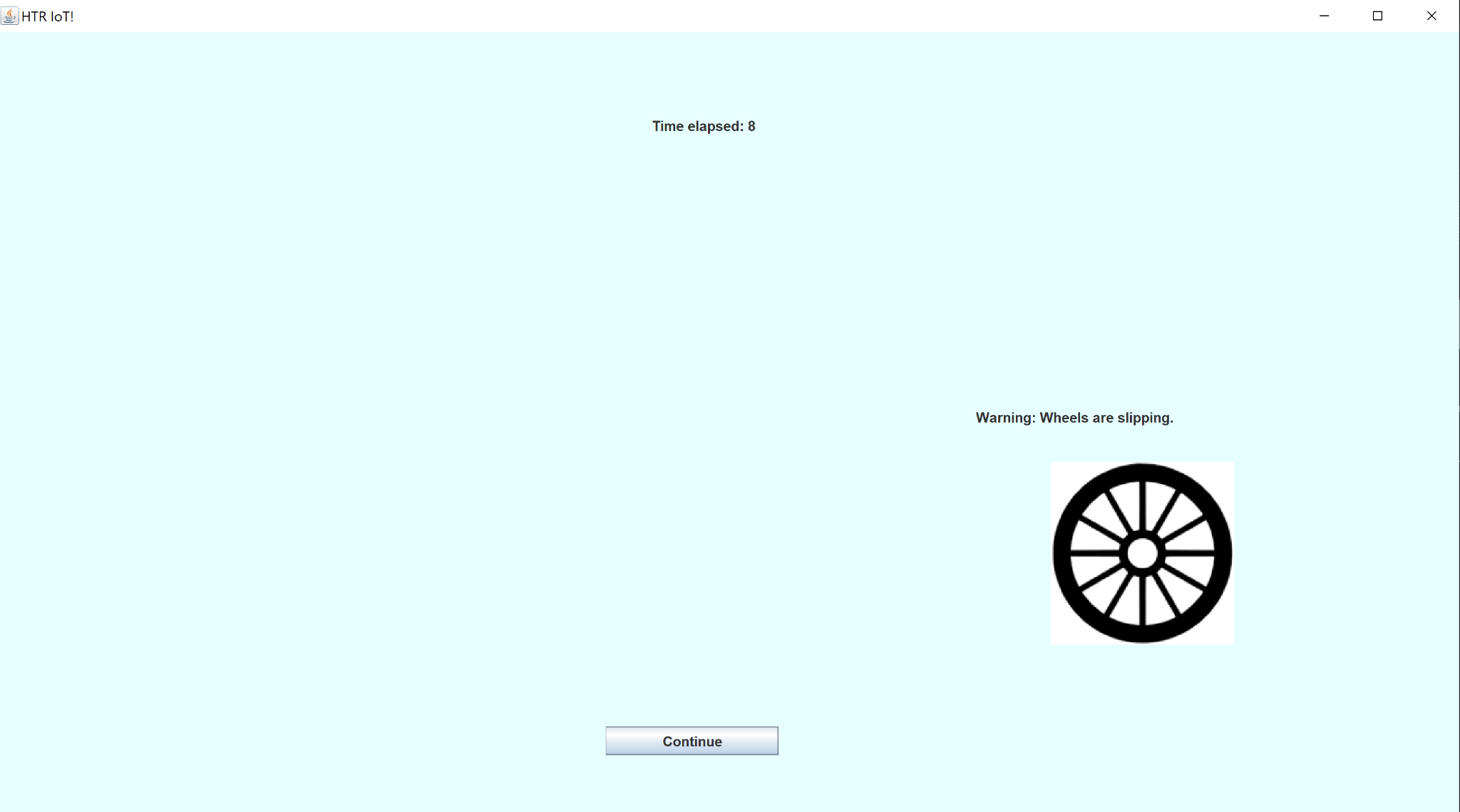


**Outcome:** Program outputs “Warning: There is a railroad crossing ahead. Honk the horn for:” along with an icon displaying the amount of time to honk the horn for. On top of that, the program also outputs “Warning: The railroad crossing ahead is not closing.”

**Result:** PASSED

### 7.1.7 - Testing Use Case 7

**Test:** IoT should check if there is wheel slippage in the train. If there is, then warn the LCO.

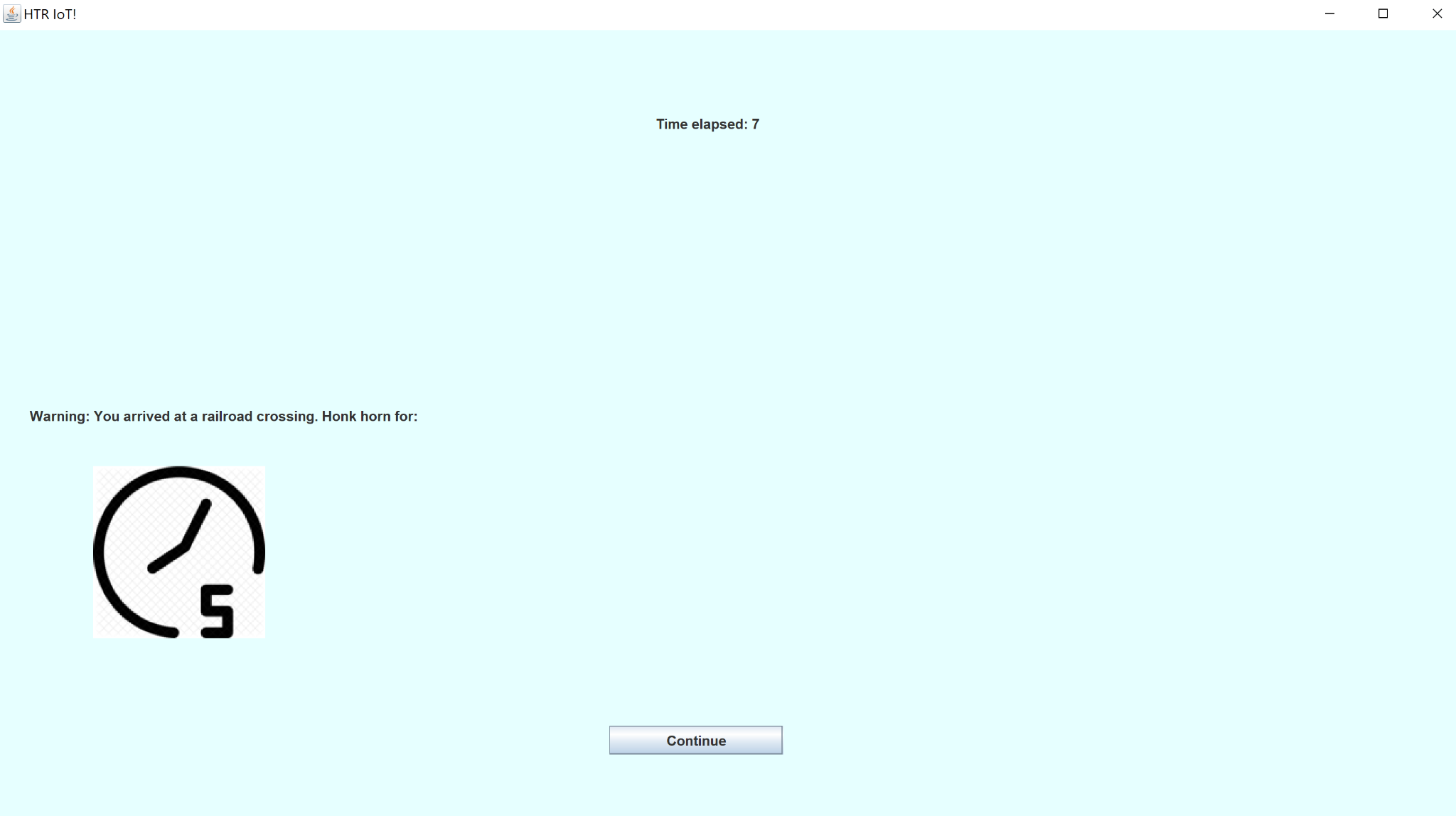


**Outcome:** Program outputs “Warning: Wheels are slipping.”

**Result:** PASSED

### 7.1.8 - Testing Use Case 8

**Test:** IoT should warn the LCO to honk the horn for 5 seconds after they have arrived at a railroad crossing.

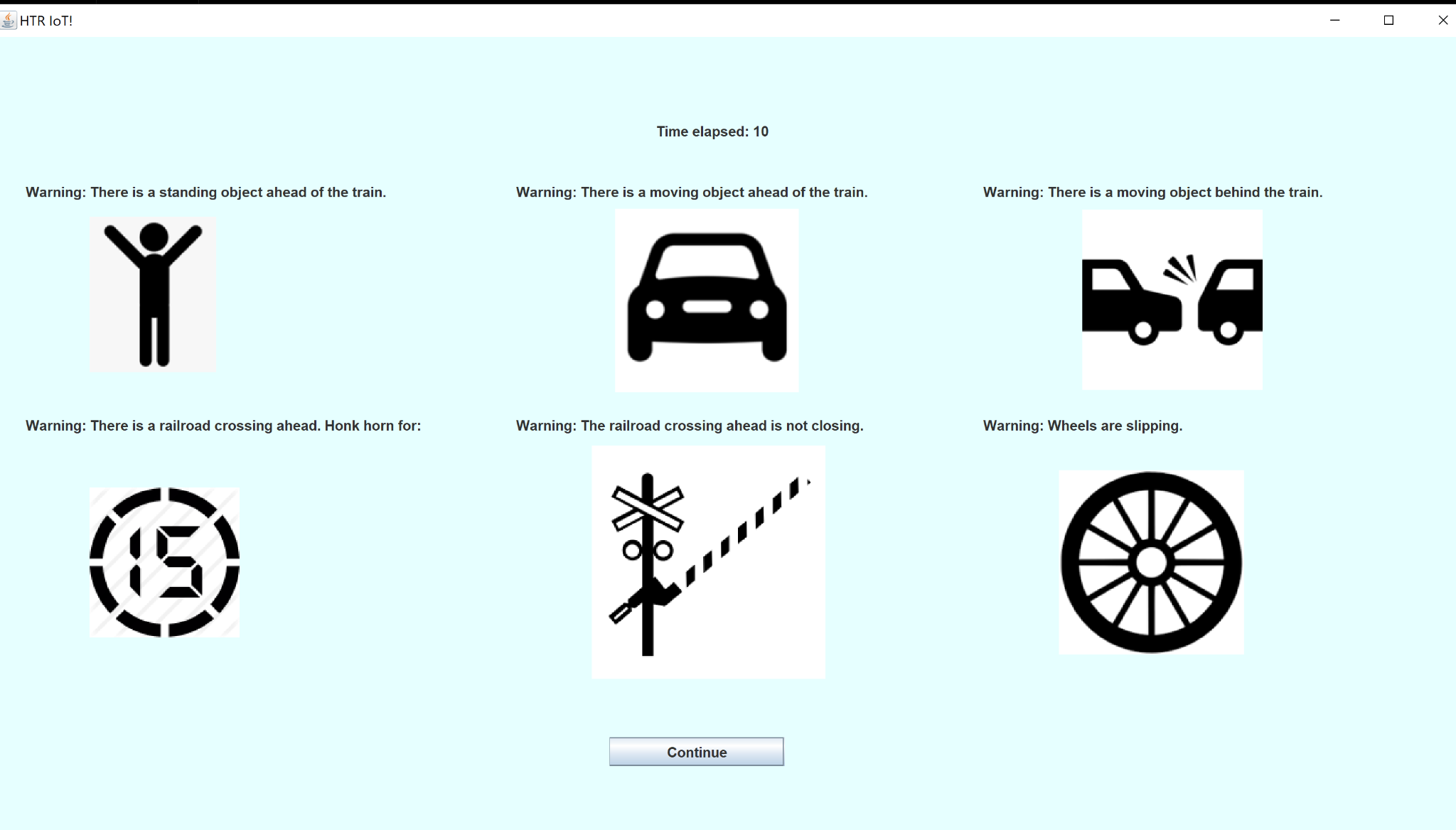


**Outcome:** Program outputs “Warning: You arrived at a railroad crossing. Honk horn for:” and an icon displaying 5 seconds is also shown.

**Result:** PASSED

### 7.1.9 - Extra Testing 1

**Test:** All warnings and icons should be displayed at once, if they are triggered, not one at a time.

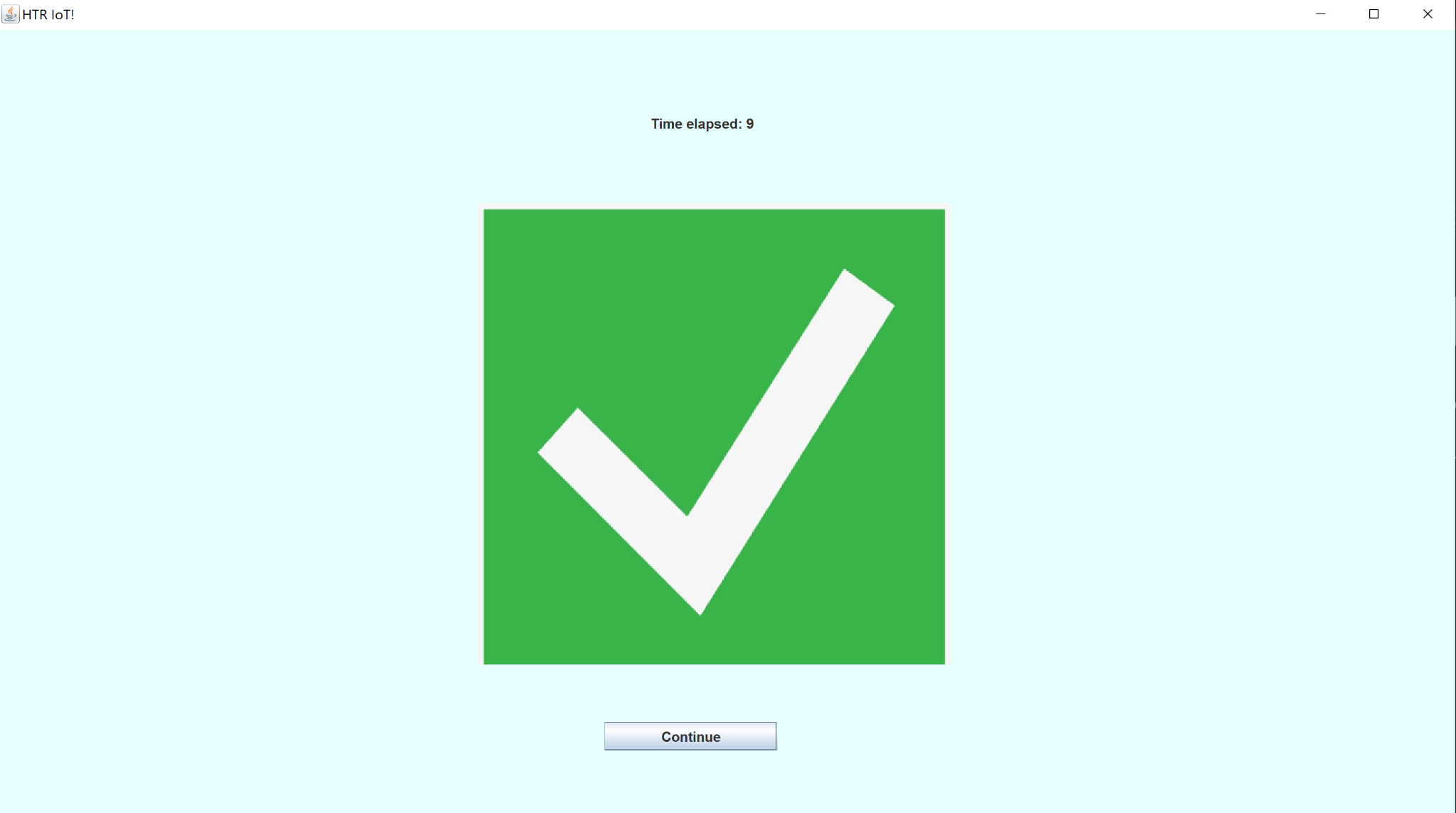


**Outcome:** All warnings and icons are displayed when triggered.

**Result:** PASSED

### 7.1.10 - Extra Testing 2

**Test:** If there are no warnings to be displayed, then the IoT should let the LCO know that there is no problem.



**Outcome:** A green checkmark is shown when no warnings are displayed.

**Result:** PASSED

## 7.2 - Validation Testing

### 7.2.1 - Non-functional Requirements Testing

**R - 1:** IoT is operable under extreme weather conditions.

**Result:** PASSED

**R - 2:** Strong components were attached to IoT.

**Result:** PASSED

**R - 3:** IoT is operable even when dropped.

**Result:** PASSED

**R - 4:** Test cases were conducted successfully.

**Result:** PASSED

**R - 5:** IoT responds faster than .5 seconds to whatever input is given.

**Result:** PASSED

**R - 6:** IoT is able to read all sensors correctly.

**Result:** PASSED

**R - 7:** IoT lets the operator know all local information.

**Result:** PASSED

**R - 8:** Train operators are only able to access the IoT with a User ID and Password.

**Result:** PASSED

**R - 9:** When using IoT, the interface prompts the user to input an id and password before allowing any further use of the IoT.

**Result:** PASSED

**R - 10:** If correct credentials are inputted, the user is allowed access to the information gathered by the IoT sensors.

**Result:** PASSED

**R - 11:** If incorrect credentials are inputted, the prompt remains and the user is not allowed access into the IoT network.

**Result:** PASSED

**R - 12:** The HTR runs on updated versions of Windows 10.

**Result:** PASSED

### 7.2.2 - Functional Requirements Testing

**R - 1:** Standing objects on the path can be detected.

**Result:** PASSED

**R - 2:** Sensors detect objects up to 1 mile.

**Result:** PASSED

**R - 3:** Sensors find the distance between the object and the train.

**Result:** PASSED

**R- 4:** If a standing object is a mile ahead of the train, IoT provides a warning.

**Result:** PASSED

**R - 5:** Sensors are installed in front and back of the train.

**Result:** PASSED

**R - 6:** Sensors detect objects up to 1 mile ahead and behind the train.

**Result:** PASSED

**R - 7:** Sensors provide the speed of moving objects detected.

**Result:** PASSED

**R - 8:** If an object is detected in front of the train going at a slower speed than the train’s, the LCO is warned of the fact.

**Result:** PASSED

**R - 9:** If an object is detected behind the train going at a faster speed than the train’s, the LCO is warned of the fact.

**Result:** PASSED

**R - 10:** Through GPS, IoT detects if there is a gate crossing within a 1 mile radius.

**Result:** PASSED

**R - 11:** Ifa gate is not correctly functioning, IoT should warn the LCO.

**Result:** PASSED

**R - 12:** Warn LCO if train is approaching any railroad crossing.

**Result:** PASSED

**R - 13:** Sensors on both sides of the train detect the RPM of the wheels.

**Result:** PASSED

**R - 14:** GPS Data is checked every 5 seconds by the IoT to detect if the wheels are slipping from the rail.

**Result:** PASSED

**R - 15:** LCO is warned when wheels are slipping.

**Result:** PASSED

**R - 16:** At start, weight detectors check that the train has equal power to weight ratio.

**Result:** PASSED

**R - 17:** LCO is warned when the current weight of the train exceeds its capacity.

**Result:** PASSED

**R - 18:** IoT constantly checks that a gate crossing is within a mile, as it should.

**Result:** PASSED

**R - 19:** In the presence of gate crossing, LCO is warned and suggested to honk the horn for 15 seconds.

**Result:** PASSED

**R - 20:** Upon reaching the gate, LCO is warned to honk the horn for 5 seconds.

**Result:** PASSED