University of Southampton

Electronics and Computer Science

ELEC6200: Group Design Project

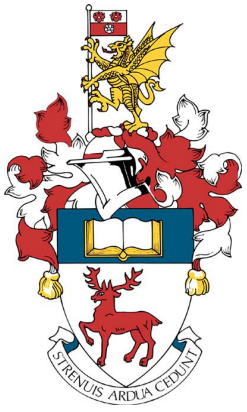
Detecting, mapping and verifying signage with computer vision and machine learning

Authors:
Charles Powell, Benjamin Sanati
Isaac Dunford, Killian Clarke,
Gerasim Tsonev

Supervisor:
Jonathon Hare

Second Examiner:
Matthew Turner

January 14, 2023



A group project report submitted for the award of
MEng Computer Science,
MEng Computer Science with AI,
MEng Electronic Engineering with AI

Abstract

At present, manual and time-consuming inspections are required to ensure a large number of signs present within a train vehicle are in conformance with regulations. This project proposes a digitised, semi-automated solution to existing methods by leveraging physical checkpoints to aid in the logging and identification of non-conformances within a train vehicle. To determine the feasibility of such an approach, this project presents - **AutoSign** - a proof-of-concept system that makes use of a mobile application to carry out inspections, coupled with computer vision and deep learning techniques to identify missing and damaged signs within a train. In testing, **AutoSign** demonstrated a consistent ability to detect non-conformances within both mock environments and live trains and was perceived positively amongst potential users. Based on these results, this project determines the approach to be feasible and presents a list of recommendations to support the transition of the proof-of-concept into a complete solution.

(24,695 Words)

Contents

List of Illustrations

Tables

Figures

Statement of Originality

- I have read and understood the ECS Academic Integrity information and the University's Academic Integrity Guidance for Students.
- I am aware that failure to act in accordance with the Regulations Governing Academic Integrity may lead to the imposition of penalties which, for the most serious cases, may include termination of programme.
- I consent to the University copying and distributing any or all of my work in any form and using third parties (who may be based outside the EU/EEA) to verify whether my work contains plagiarised material, and for quality assurance purposes.
- I have acknowledged all sources, and identified any content taken from elsewhere.
- I have not used any resources produced by anyone else.
- I did all the work myself, or with my allocated group, and have not helped anyone else.
- The material in the report is genuine, and I have included all my data/code/designs.
- I have not submitted any part of this work for another assessment.
- My work did not involve human participants, their cells or data, or animals.

Acknowledgements

The team would like to extend their deepest gratitude to the project supervisor Jonathon Hare, and Adam Wellings and Tom Bates of Stewart Signs, for their guidance, support and dedication throughout the project's duration.

The team acknowledges the use of the IRIDIS High-Performance Computing Facility, and associated support services at the University of Southampton, in the completion of this work.

SECTION 1

Introduction

1.1 The Problem

Currently, over 4500 train carriages are either in service or planned across the UK. The interior and exterior of each of these vehicles contain a large number of signs which address safety-critical, informational and accessibility aspects, and which must conform to manufacturer and regulatory standards. To adhere to these standards, the signs present in a vehicle must be inspected on a regular basis, with non-conformances (missing, damaged or incorrect positioning) documented and remedies scheduled for the next maintenance.

If a vehicle leaves maintenance with a problem, penalties are incurred on the responsible parties (e.g., train manufacturer/operator), which vary depending on the severity of the issue, and grow rapidly should the issue persist.

At present, inspections and remediations are carried out and logged manually by trained personnel. There exists no system to ensure reliable inspection, documentation, purchasing and remediation of a train vehicle's signs. The challenges presented by the current methods are experienced first-hand by Stewart Signs - a sign manufacturer, and this project's client.

1.1.1 Problem Statements

Problem Statements

- **PS1** There are a large number of train vehicles, which each have a large number of signs that require regular inspection and maintenance.
- **PS2** The current approaches to the inspection of a train vehicle, and the remediation of non-conformances are challenging and expensive, in terms of personnel, time and management.
- **PS3** At present, there exists no system to support the reliable inspection, documentation, purchasing and remediation of a train vehicle's signs.

1.2 The Proposed Solution

To address the problem statements listed above, this project seeks to develop a proof-of-concept system that serves as a digitisation of the existing approach, and that introduces machine learning techniques to provide automation. In particular, this project proposes a system that supports:

- Collection and storing of photos/videos of a train vehicle.
- Processing of the recorded data, and production of a report detailing the state of signs within the train.
- Purchasing of replacement signs for identified non-conformances.
- Updating of the train vehicle's maintenance status following remediation, and logging of this remediation.
- Viewing of a vehicle's current maintenance status, and history of inspections and remediations.

1.2.1 Project Goals

Project Goals

- **PG1** Develop a proof-of-concept system that supports the inspection of a train vehicle via collected photos/videos, the creation of a digital report detailing missing and damaged signs, the purchasing of replacement signs, the logging of remediations, and the viewing of a vehicle's current status and history of inspections and remediations.
- **PG2** Evaluate the performance of the developed system to detect the presence and type of non-conformance within a train vehicle, in terms of both time and accuracy.
- **PG3** Demonstrate the feasibility of the system as a commercial product.
- **PG4** Consider the perception of potential users towards the developed system.

1.2.2 Project Motivation

The current methods for train inspection leave manufacturers vulnerable to large penalties. The system proposed by this project aims to reduce the manufacturer's exposure to these penalties by:

- Digitising the inspection and remediation process.
- Increasing the speed of inspections and remediations.
- Lowering the frequency of inaccuracies within inspections.
- Linking the inspection/remediation workflow to the purchasing of signs.

By providing such a system to manufacturers, it is the hope of Stewart Signs, that manufacturers will be incentivised to use their services over competitors.

SECTION 2

System Design

2.1 Term Definitions

Listed below are the definitions for a set of key terms used throughout this project.

- **Train Vehicle**: A single train carriage.
- **Train Checkpoint**: A specific section of a train. This term is defined to provide separation between the various sections of a train.
- **Department**: A specific checkpoint within a train, that contains rows of seats and windows.
- **Platform**: A specific checkpoint within a train, from where passengers can enter/exit the train.
- **Sign**: A sign within a train vehicle. A type of sign may have multiple instances within a single train vehicle or checkpoint but is identifiable by its location within the vehicle. Each sign is governed by standards and regulations that determine both its required location and appearance.
- **Conformance Status**: Refers to the status of a sign, checkpoint or vehicle in relation to the standards and regulations placed on signs. A single sign is either conforming or non-conforming based on these standards and regulations, while a checkpoint or vehicle is only conforming if all of the signs within that checkpoint or vehicle are themselves conforming.
- **Inspection**: The inspection of an entire train vehicle, which is carried out by analysing each individual sign within each of the vehicle's checkpoints according to the relevant standards and regulations. The result of an inspection is that each sign, checkpoint, and vehicle as a whole is classified as conforming, or non-conforming.
- **Remediation**: The fixing of a non-conforming sign, such that it now becomes conforming.
- **Train Vehicle Status**: The current status of a train vehicle, which is dependent on the latest inspection of the vehicle, and any remediations carried out since this inspection.

2.2 Specification

When formulating a specification, two core components were identified - an interface the user could interact with, and a processor that handles the processing of inspections using machine learning techniques.

For the interface, a specialised device, and a mobile application were considered. **Table 2.1** provides a breakdown of these approaches, as well as a justification for the decision to use a **Mobile Application**.

Table 2.1: System Interface Design Choices

Options Considered	
Option	**Description**
Specialised Device	A specialised device, such as a camera system and monitor, is optimised to collect photos/videos of the inside of a train vehicle in order to carry out an inspection.

Mobile Application	A mobile application, capable of running on any mobile device, that uses the phone's built-in camera to collect photos/videos of a train vehicle in order to carry out an inspection.
Chosen Option	
Option	**Justification**
Mobile Application	• **Team Skill Set**: The skill sets and past development experience of the group members lend themselves to the development of a mobile application in favour of the specialised hardware device. • **Development Time**: The development of a mobile application can take place significantly quicker than that of a specialised device, given that the already functioning hardware and software of the mobile device can be used. • **Suitability**: A mobile application is an optimal solution for this setting, as they are widely accessible and capable of the required task.

For the processor, online processing within the mobile application, and offline processing on a dedicated processing server were considered. Table 2.2 provides a breakdown of these approaches, as well as justification for the decision to use a dedicated **Processing Server**.

Table 2.2: System Processor Design Choices

Options Considered	
Option	**Description**
Within Mobile Application	Machine learning programs are deployed on the mobile phone alongside the application, and that makes use of the phone's processor.
Dedicated Processing Server	A dedicated server running on a local machine communicates with the mobile application to carry out processing using the machine's processor.
Chosen Option	
Option	**Justification**
Dedicated Processing Server	• **Team Skill Set**: The skill sets of the group members lend themselves to a dedicated processing server, in favour of integration with the mobile application. • **Speed/Processing Power**: Complex machine learning algorithms are required in the system, and are likely unsuitable for a mobile phone processor.

However, by choosing to make use of disjoint systems for the interface and processor, their method of communication must be addressed. For this purpose, a dedicated data server, and a Backend-as-a-Service (BaaS) cloud server. Table 2.3 provides a breakdown of these approaches, as well as justification for the decision to use a **Cloud Server**.

Table 2.3: System Integration Design Choices

Options Considered	
Option	**Description**
Direct Communication	The mobile application and processing server communicate directly through specialised communication channels.
Dedicated Data Server	A dedicated data server running on a local machine that the mobile application and processing server use as an intermediary to communicate.
BaaS Cloud Server	A cloud server running on a BaaS platform that the mobile application and processing server use as an intermediary to communicate.
Chosen Option	
Option	**Justification**
BaaS Cloud Server	• **Team Skill Set**: The skill sets and past development experience of the group members lend themselves to the use of a BaaS cloud server. • **Time**: The time required to set up a BaaS cloud server is minimal compared to the alternatives. • **Cost**: A BaaS cloud server requires the minimal cost to maintain. • **Integration**: BaaS cloud servers are widely supported, and integrate well with application and processing tools. • **Performance, Reliability & Security**: BaaS platforms offer greater performance, reliability and security than the alternatives.

Additionally, to make use of machine learning techniques, and given the novelty of the problems being addressed, specialised **Datasets** will be required during the project, which represents the final component of the system. The architecture for the complete system - known as **AutoSign** - is summarised below, and illustrated by **Figure 2.1**.

- **Application**: A mobile application that allows users to carry out inspections by collecting footage of a train vehicle, purchasing replacement signs, logging remediations, and viewing the status of a vehicle and a history of its inspections and remediations. The application will upload gathered data, such as inspection media to the cloud server, and download reports posted by the processing server.
- **Processing Server**: A dedicated server that will download inspection media from the cloud server, process the media using machine learning techniques, produce a report detailing the status of the vehicle, and upload this report to the cloud server.
- **Datasets**: Specialised datasets are required for the machine learning techniques to be deployed on the processing server.
- **Cloud Server**: A BaaS cloud server that will provide a database for the system's data and storage for the systems files. The cloud server will act as an intermediary between the mobile application and the processing server, allowing them to communicate with each other and have access to the same data.

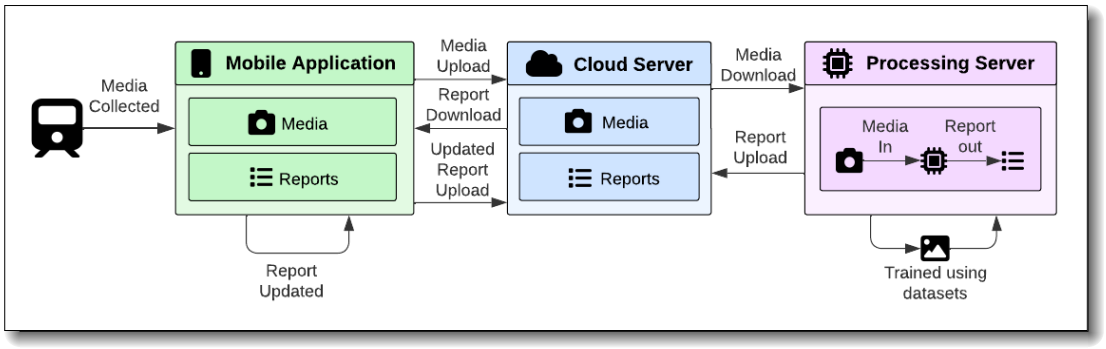


Fig. 2.1. Proposed system diagram

2.3 Analysis

2.3.1 Requirements

The following sections provide a breakdown of design/implementation requirements for **AutoSign**, divided based on system component. Note that no requirements are given for the datasets, as they are required only to enable necessary machine-learning techniques.

2.3.1.1 Application

Table 2.4: Application Design Requirements

ID	Title	Description	Priority
R1	Inspection Capture	The application **must** allow users to inspect a train carriage by collecting photos/videos of the carriage using the phone's camera.	**MUST**
R2	Inspection Upload	The application **must** upload photos/videos collected during an inspection to the cloud server.	**MUST**
R3	Inspection Viewing	The application **must** allow the user to view the results of an inspection, and download these results from the cloud server.	**MUST**
R4	Remediation Logging	The application **should** allow the user to log the remediation of a non-conforming sign, and record a photo of the remediation.	**SHOULD**
R5	Remediation Upload	The application **should** upload remediation data to the cloud server.	**SHOULD**
R6	Remediation Viewing	The application **should** allow the user to view a history of remediations performed on a train vehicle.	**SHOULD**
R7	Vehicle status	The application **should** allow the user to view the current maintenance status of a train vehicle, which takes into account the most recent inspections, and any remediations since this inspection.	**SHOULD**

R8	Purchasing Signs	The application **SHOULD** allow the user to purchase replacement signs for identified non-conformances.	**SHOULD**
R9	Adding Train Vehicles	The application **COULD** allow the user to add new train vehicles into the system.	**COULD**
R10	Inspection Capture Assistance	The application **COULD** provide capture assistance during the inspection process.	**COULD**

2.3.1.2 Processing Server

Table 2.5: Processing Server Design Requirements

ID	Title	Description	Priority
R11	Missing Sign Detection	The processing server **MUST** download inspection media from the cloud server, process this media and detect signs missing from a train.	**MUST**
R12	Inspection Results	The processing server **MUST** upload the results of processed inspection media to the cloud server.	**MUST**
R13	Damaged Sign Detection	The processing server **SHOULD** download inspection media from the cloud server, process this media, and detect signs missing from the train.	**SHOULD**
R14	Video Processing	The processing server **COULD** be able to process videos captured during inspections.	**COULD**
R15	Incorrectly Positioned Sign Detection	The processing server **WON'T** download inspection media from the cloud server, process this media, and detect incorrectly positioned signs within the train.	**WONT**

2.3.1.3 Cloud Server

Table 2.6: Cloud Server Design Requirements

ID	Title	Description	Priority
R16	Database	The cloud server **MUST** provide a database for system data.	**MUST**
R17	File Storage	The cloud server **MUST** provide file storage for application files.	**MUST**
R18	Communication	The cloud server **MUST** serve as an intermediary between the application and processing server, allowing both components to communicate with each other.	**MUST**

2.3.2 Stakeholder Analysis

Table 2.7: Stakeholder Analysis

Name	Type	Description
Inspector	Primary	Individuals responsible for inspecting the signs on a train vehicle. Inspectors will use the application to carry out a digital inspection by capturing footage of the vehicle.
Remediator	Primary	Individuals responsible for remediating any non-conforming signs within a train. Remediators will use the application to view a train's status, perform required maintenance, and log any remediations made.
Train Manufacturer	Secondary	The entity responsible for the production and maintenance of the train vehicle. Manufacturers are liable for any non-conformances present on a vehicle and will have their employees use the system to improve their detection.
Train Operator	Tertiary	The entity responsible for operating the train in service, which can only be done when the vehicle is conforming. As such, the operator is relying on the system to detect non-conformances.

2.4 Development Plan

Development of **AutoSign** has been split into two deliverables, which each aim to address various system requirements and provide a minimum viable product. Deliverable 1 aims to develop the basic system, while deliverable 2 aims to extend its functionality, and address extension requirements should time permit. This development plan is summarised in **Table 2.8**, and the following sections of this report are based on this deliverable structure. Note that the dataset component of the project is not included in this plan, as its only goal is to enable the required machine-learning techniques.

Table 2.8: System Development Plan

Deliverable 1		
Component	**Primary Goal**	**Req's**
Application	Develop an application capable of performing and viewing the results of an inspection via images.	**R1**, **R2**, **R3**
Processing Server	Develop a processing server capable of detecting missing signs within images.	**R11**, **R12**
Cloud Server	Develop a cloud server that supports the requirements of the application and processing server.	**R16**, **R7**, **R18**

Deliverable 2		
Component	**Primary Goal**	**Req's**
Application	Extend the application to support video-based inspections, the purchasing and remediation of non-conforming signs, and the viewing of vehicle status and remediation logs.	**R4**, **R5**, **R6**, **R7**, **R8**
Processing Server	Extend the processing server to support the processing of video-based inspections, and the detection of damaged signs.	**R13**, **R14**
Cloud Server	Extend the cloud server to support the additional requirements of the application and processing server.	**R16**, **R7**, **R18**

SECTION 3

Deliverable 1

3.1 Cloud Server

3.1.1 Implementation Choices

As leading BaaS providers that offer the required services (database and file storage), AWS Amplify, Back4App and Firebase were considered for the cloud server, with Firebase selected for the following reasons:

- **Team Skill set**: The skill sets and past experiences of the group members makes Firebase the most suitable option.
- **Simplicity & Feature Base**: Firebase is the simplest service to initialize and use, and offers the widest feature set.
- **Usage Limits & Pricing**: Firebase places fewer usage limits on its services, and offers the best pricing plans.
- **Documentation & Community**: Firebase provides a vast array of documentation and has a large support community.
- **Integration**: Firebase provides simple and well-documented integration with the chosen application and processing server technologies.

Additionally, Firebase's Firestore and Cloud Storage services were chosen for the system database and file store respectively.

3.1.2 Initialization & Set-Up

Firebase provides a web-console that can be used to create and manage 'projects', which serve as cloud server instances. Setting up the cloud server for the system can be achieved simply by creating a project within the console website, and initializing the required services within this project, as shown in Figure 3.1 and Figure 3.2 respectively.

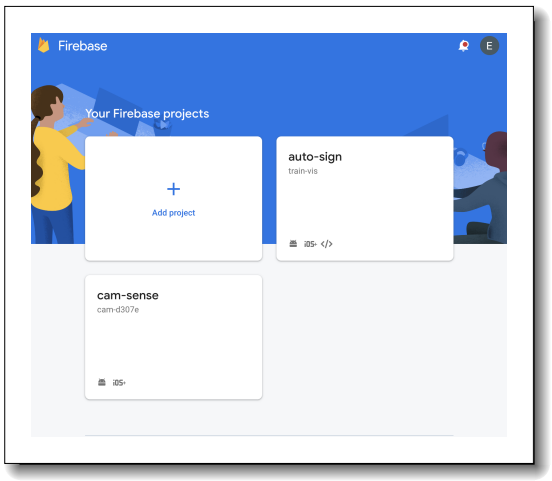


Fig. 3.1. A screenshot of the creation of a project within the Firebase console

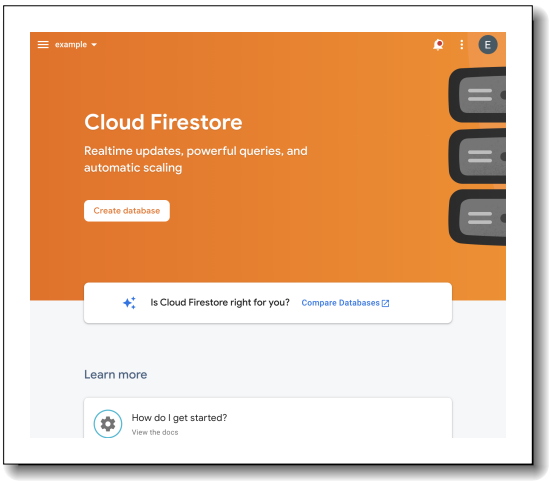


Fig. 3.2. A screenshot of the initialization of a Firebase service within the Firebase console

3.1.3 Data Model

Before the cloud server can be integrated with the system, a data model must be defined which accounts for how data is organised in both the database, and file store. This data model is described in the following sections.

3.1.3.1 Database

Figure 3.3 shows the organisation of data within the Firestore database. Objects are defined to model a train vehicle, and a checkpoint within a vehicle, which provides a list of signs present in the checkpoint. These objects serve as a gold-standard definition of the train, which can be used by the application to understand what must be captured during an inspection, and used by the processing server when comparing detected signs with the expectation. Additional objects are defined to model the inspection of a vehicle, and a checkpoint within a vehicle, which simply extend their standard counter parts to include conformance status, with values of “conforming" or “non-conforming". Additionally, a processing status field is added to the model of an inspection, with values “pending", “processing" and “processed", which can be used by the processing server to determine which inspections require processing. Note that these objects model only the information relating to an inspection, and not the media collected during one.

The general structure of the database model is flat, resulting in objects referencing related objects, rather than being sub-contained, in accordance with the official documentation [14].

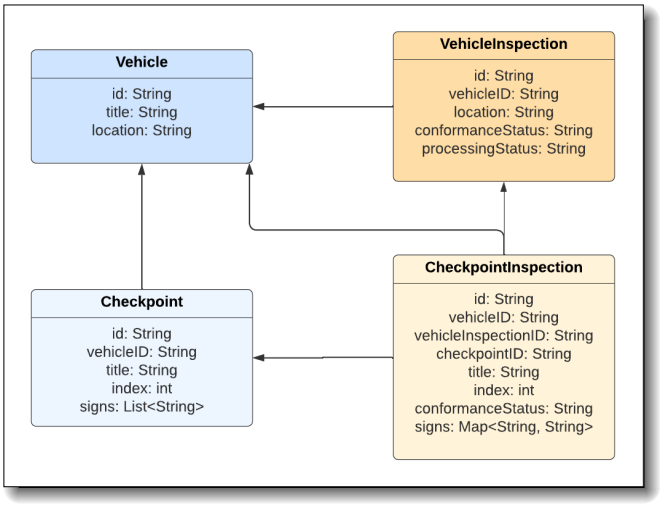


Fig. 3.3. Deliverable 1 database model

3.1.3.2 Storage

Figure 3.4 shows the organisation of files within the Cloud Storage drive, which is structured such that files/directories are named, and thus located, using the ID of the Firestore document they are relevant to.

The drive contains one directory for each vehicle, which in turn contains three items - an avatar image of the train, a directory of the gold-standard images of its checkpoints and a directory for the media captured during each inspection. The gold-standard images of the checkpoints are present to serve as examples during the inspection process in the application.

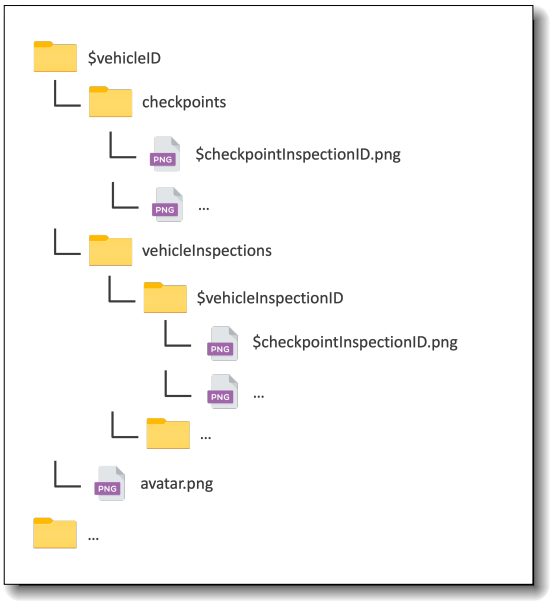


Fig. 3.4. Deliverable 1 storage model

3.1.4 Integration

Due to the project's implementation choices, the application and processing server are able to access the cloud server using simple API calls in their respective programming languages, and by referencing the configuration information provided by Firebase.

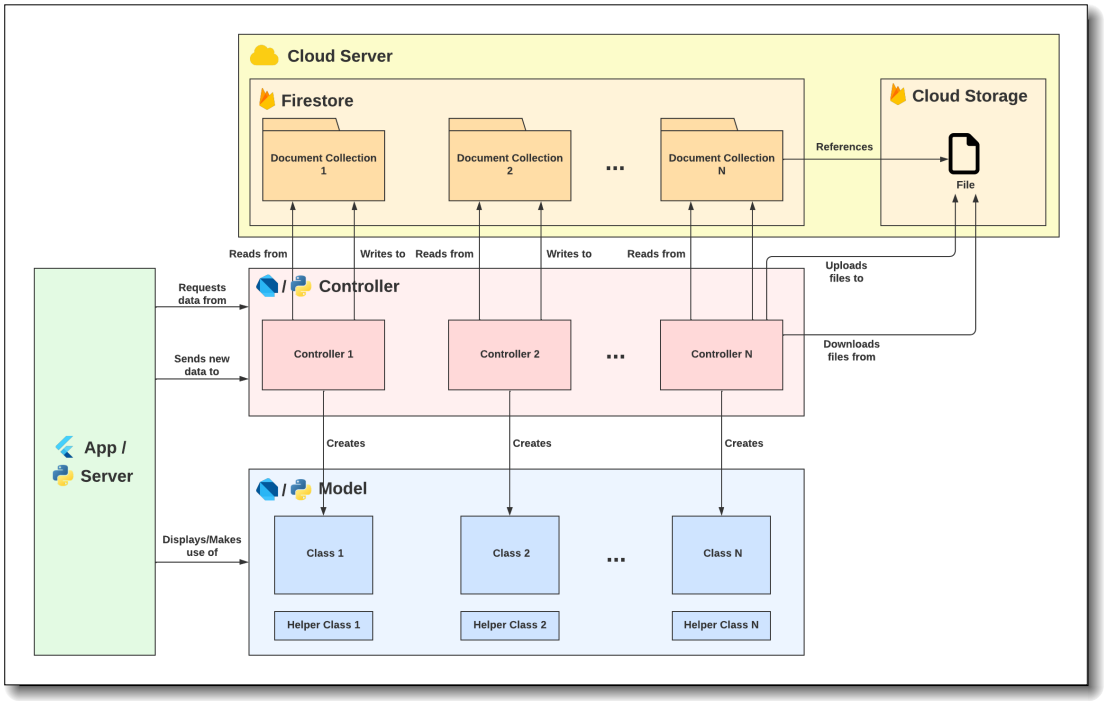


Fig. 3.5. The model-controller architecture used by the processing server and application when connecting to the cloud server

For improved structure, the application and processing server adopt a model-controller architecture when connecting to the cloud server, as shown in Section - 3.5. In this architecture,

each object is modelled by three components - a document within a Firestore document collection, and a controller and model class within the application or processing server. Additionally, system files are stored in Cloud Storage, but referenced by the associated Firestore Document, and helper classes are defined occasionally to model object related data. Based on information flow, the application and processing server will make read/write requests to the corresponding controllers which, in turn, connect to the appropriate Firebase endpoints. In the case of reads, the controller will return the gathered data in the form of a model class. This provides a robust flow of data with strict typing, reducing the risk of errors and improving the code structure.

3.2 Application

3.2.1 Design

3.2.1.1 Structure

Based on it's requirements, the following structure is proposed for the application:

- **Start Page**: Displayed when the application is started, which allows the user to enter the ID of the train vehicle they wish to view.
- **Profile Page**: The main page of the application. From here, all of the functionality associated with the current train vehicle is available to the user.
- **Inspect Page**: Allows the user to carry out an inspection of the train vehicle using their phone camera.
- **Inspections Page**: A page for viewing the results of an inspection.

3.2.1.2 Use Case Diagrams

Figure 3.6 models the use cases of the application, which address carrying out and viewing the results of an inspection. The relationships between the use cases are also modelled, which can then be translated into the navigation system of the application. Within the inspect sequence, it is required that the user capture footage and then upload that footage to the cloud server. They also have the option to review the footage. Furthermore, the use cases for the viewing of inspections show how each additional use case extends the previous one. The relationships between the use cases is then going to be translated into the mock-ups to define the navigation between them. Finally it is clear how all of the functionality of the application extends from the train carriages initial profile page.

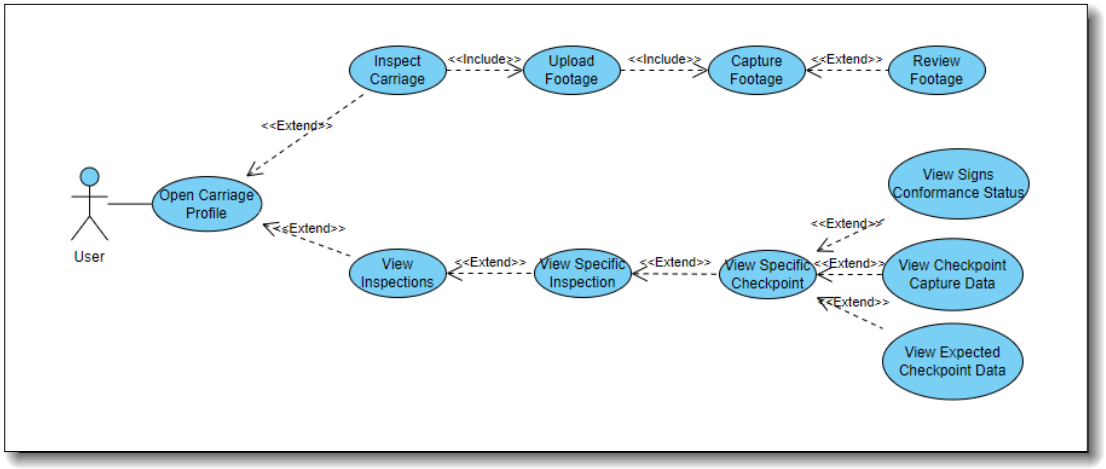


Fig. 3.6. Use case diagram for deliverable 1

3.2.1.3 Activity Diagram for Inspection

The primary process present in the deliverable 1 application is the carrying out of an inspection. To model this process, an activity diagram was defined, as shown in Figure 3.7.

During an inspection, the user will systematically capture, and then review, footage for a specific

train vehicle, before having the option to review all captured footage at once. Upon confirmation, the footage is uploaded to the cloud server to be processed. It should be noted that the user would be able to cancel the inspection at any stage, yet this logic is not included in the diagram to avoid clutter.

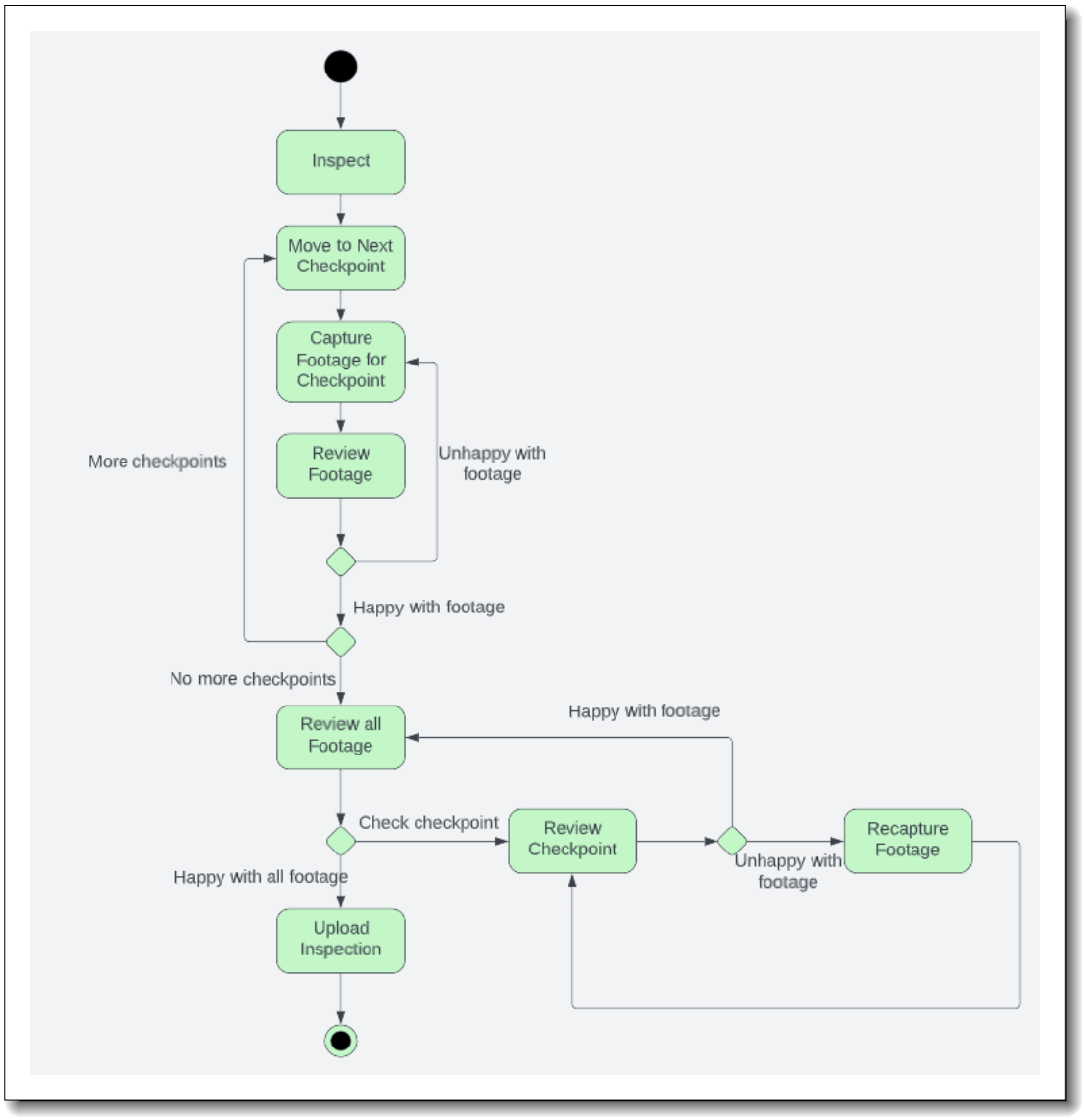


Fig. 3.7. Activity diagram for the inspection process within the application

3.2.1.4 Mock-ups

Before developing the application, mock-ups are required to define the user-interface of the application, as well as the flow between its various sections. Samples of these mock-ups are presented in the following sections.

To produce these mock-ups, the online tools Moqups and Figma were considered, with the group chosing Figma based on its capacity to produce mock-ups that more closely resemble standard Flutter user-interface. The result of this is that the mock-ups created through Figma can look identical to the final application, reducing the workload required during the implementation phase.

3.2.1.5 HCI and Usability Concepts

When designing the mock-ups for the application, a focus was placed on:

- **Consistency**: By having a consistent application design, the chance that the user ever becomes disoriented, confused or frustrated in their experience is reduced.
- **Familiarity**: The users should already have experience with mobile applications and by capitalizing on their already existing knowledge, the application experience can become more intuitive.
- **Usability**: The design should be as simple as possible whilst still achieving all of the desired functionality as having verbose or unnecessary parts of the application can damage the user experience.

These three criteria were achieved during the application design through consistent and appropriate use of colours and icons, which would ensure the system never feels confusing to the user.

For example, colours are often used throughout the application to imply sentiment, while icons are placed within system actions to provide a visual understanding of their outcome. In addition, the camera interface for the inspection process is modelled off of typical camera facilities that can be found on a mobile phone, providing a sense of familiarity to the user.

Finally, the widgets in the app, such as the scrollable lists as well as the camera widgets are ones commonly used in lots of applications and the positioning of icons, such as the backwards icon always being placed in the top left corner of the application, further increases the familiarity of the application.

3.2.1.6 Start Page

Figure 3.8 shows the mock-up for the start page of the application, which allows the user to enter the ID of a train vehicle to begin their workflow.

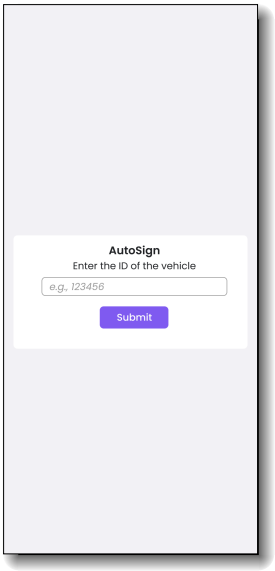


Fig. 3.8. Sample application start page mock-up

3.2.1.7 Profile Page

Figure 3.9 shows a range of sample mock-up interfaces for the profile page within the application, which allow the user to perform actions on, and view information related to their current train vehicle. Note that the current current status of the train is also shown in the profile page. This status is a reflection of the most recent inspection performed on the vehicle, and should the user select this option, they will be redirected to this most recent inspection.

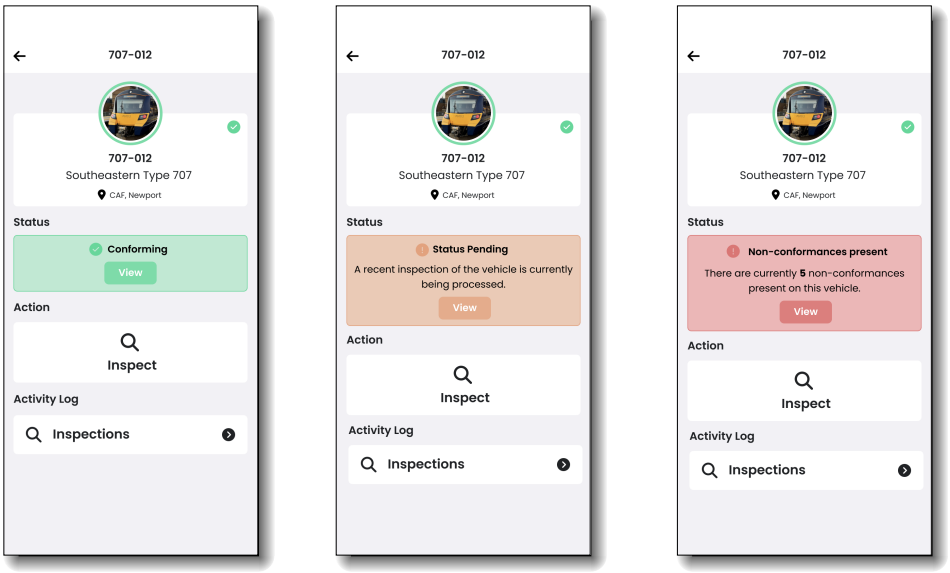


Fig. 3.9. Sample application profile page mock-ups

3.2.1.8 Inspect Page

The mock-ups shown in Figure 3.10 present the interfaces designed to support the capturing, reviewing and uploading of an inspection within the application.

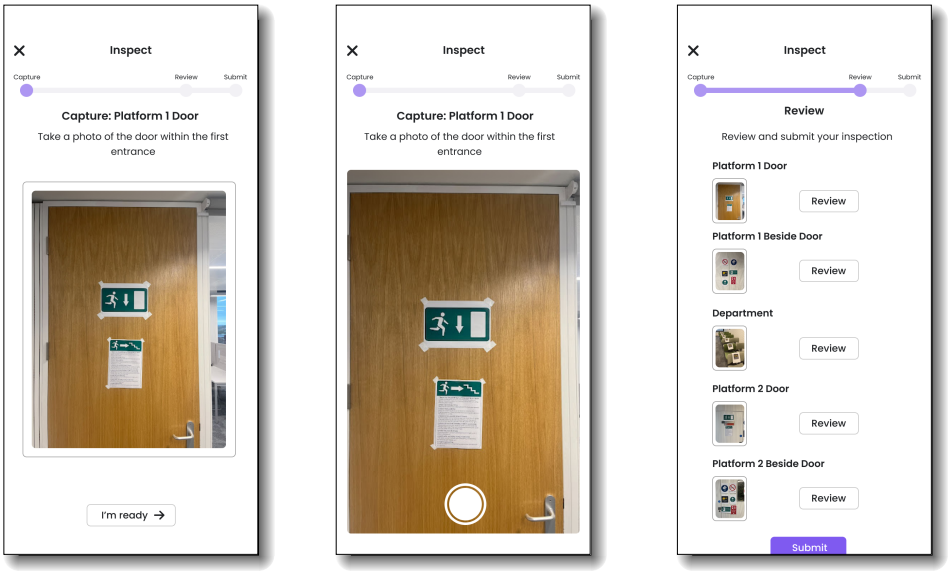


Fig. 3.10. Sample application inspect page mock-ups

3.2.1.9 Inspections Page

Figure 3.11 shows the mock-ups created for the inspections page. Inspections are sorted by date for the convenience of the user, and upon selecting an individual inspection, a breakdown of it's checkpoints and their corresponding conformance is provided. Furthermore, by selecting an individual checkpoint, a breakdown of the conformance of it's signs is listed, along with images that show both what was captured during the inspection, and what is expected of the checkpoint.

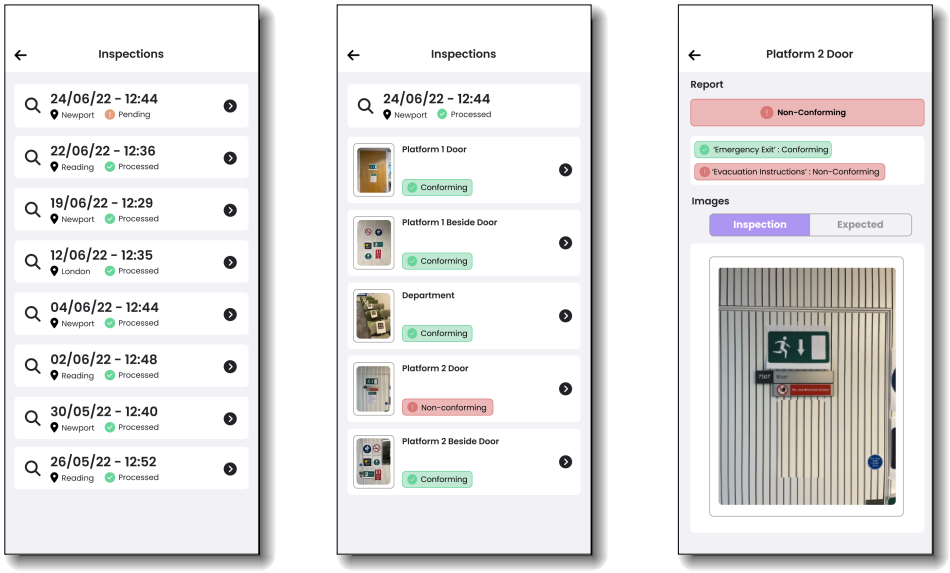


Fig. 3.11. Sample application inspections page mock-ups

3.2.1.10 Scenarios

In accordance with the mock-ups defined above, a number of scenarios are defined to model typical use of the system and expected performance, which are listed below. In testing, these scenarios can be used to ensure the application supports all of the required functionality.

In these scenarios, there are initial, alternative, and additional flows. Initial flows describe a series of actions, followed by the expected outcome, alternative flows take a step from an initial scenario and vary it somehow, and additional flows expand upon the initial scenario to define more steps.

Scenario 1 - User carries out full successful inspection of a train carriage

1. User enters ID for a train carriage and is presented with the profile page for that train carriage.
2. User selects option to perform a new inspection for the train inspection.
3. User confirms that they want to perform a new inspection.
4. User is presented with a template image for the first checkpoint and confirms they're ready to take a photo for it.
5. User takes a photo for the associated checkpoint.
6. User is presented with the photo they've taken and confirms they're happy with it.
7. User repeats the previous 3 steps for each checkpoint associated with the train currently being inspected.

8. User is given the option to review all of the photos they have taken before they choose to submit the inspection.
9. User selects to finish after the inspection has been uploaded to the cloud server.

Alternative Flow 1 at Step 6 - User decides to recapture footage from the train during the walkthrough

- User selects the option to retake a photo for a given checkpoint.
- User takes a new photo for the associated checkpoint.
- Process continues again from step 6.

Alternative Flow 2 at Step 8 - User decides to recapture footage from the train after the walkthrough during the review stage

- User selects the option to review a photo for a particular checkpoint.
- User selects the option to retake the photo taken for this checkpoint.
- User takes a new photo for the associated checkpoint.
- User is presented with the photo they've taken and confirms they're happy with it.
- Process continues again from step 8.

Alternative Flow 3 at Step 8 - User decides to check the footage from the train after the walkthrough during the review stage however they don't need to recapture any footage

- User selects the option to review a photo for a particular checkpoint.
- User confirms the photo for the given checkpoint is okay.
- Process continues again from step 8.

Alternative Flow 4 at any step from 4 to 8 - User starts an inspection and then cancels it

- User is presented with a dialog box where they confirm that they want to cancel the current inspection, discarding all of the currently captured footage.

Alternative Flow 5 at any step from 4 to 8 - User starts an inspection and then considers cancelling the inspection before deciding to continue from where they were currently

- User is presented with a dialog box where they confirm that they want to continue with the current inspection.

Scenario 2 - User views the output for an inspection of their choice

1. User enters ID for a train carriage and is presented with the profile page for that train carriage
2. User selects option to view a list of existing inspections for the train carriage.
3. User is presented with a list of inspections sorted by date for the given train carriage and selects the one they wish to view.

4. User is presented with the list of checkpoints for the train carriage where each one has it's associated conformance status.

Alternative Flow 1 at Step 4 - User views the output for an inspection of their choice which hasn't been processed yet

- User is presented with the list of checkpoints for the train carriages where all checkpoints have a conformance status of pending as the inspection hasn't been processed yet.

Additional Flow 1 after Step 4 - User views the output for a specific checkpoint within a chosen inspection with no non-conformances.

- User selects the checkpoint they wish to view with no non-conformances present.
- User is presented with the list of signs within the given checkpoint as well as their conformance status of conforming. User is also shown the captured footage for that checkpoint on the given inspection.

Additional Flow 2 after Step 4 - User views the output for a specific checkpoint within a chosen inspection where non-conformances were present.

- User selects the checkpoint they wish to view with non-conformances present.
- User is presented with the list of signs within the given checkpoint as well as their conformance status. User is also shown the captured footage for that checkpoint on the given inspection.
- User selects the expected option to view what a typical image with no non-conformances would look like for that checkpoint for the train carriage.

Additional Flow 3 after Step 4 - User views the output for a specific checkpoint within a chosen inspection which hasn't been processed yet.

- User selects the checkpoint they wish to view.
- User is presented with the list of signs within the given checkpoint as well as their conformance status which is pending as the inspection hasn't been processed yet. User is also shown the captured footage for that checkpoint on the given inspection.

Scenario 3 - User views the status of a train carriage.

1. User enters ID for a train carriage and is presented with the profile page for that train carriage.
2. User selects option to view the status for the train carriage.
3. User is presented with the most recent inspection for the train carriage.

3.2.2 Implementation

3.2.2.1 Implementation Choices

Following consideration of Flutter and Kotlin as development frameworks, Flutter was chosen for the following reasons:

- **Team Skill Set**: The skill sets and previous experiences of the application development team makes Flutter the appropriate choice.
- **Documentation & Community**: Flutter provides a vast array of documentation and has a large support community.

- **Development Time**: Flutter supports quick and efficient development of applications, allowing optimal time to be spent developing system features.
- **Integration**: Flutter provides simple and well-documented integration with the chosen cloud server technology.
- **Development Costs**: Flutter is a free-to-use technology,reducing the budgetary constraints on the project.

3.2.2.2 Workflow

Figure 3.12 illustrates the implementation workflow for the application, which can be broken down as follows:

- **UI Creation**: The basic interface for the application is defined and populated with arbitrary data.
- **Routing**: Links are defined between the basic interface components to support the required navigation between them.
- **Backend Connection**: The interface components are connected to the cloud server to display system data.
- **Functionality Testing**: The functionality of the developed system is evaluated and compared to the expected behaviour.

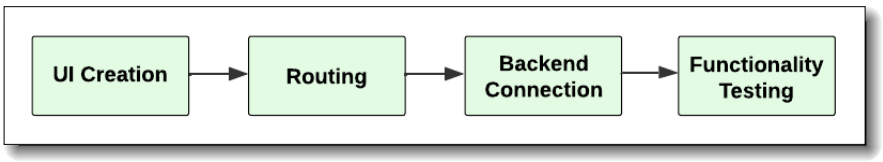


Fig. 3.12. Application development workflow

3.2.2.3 Application Routing

To support routing, the application makes use of the third-party navigation package - GoRouter - in favour of the native system provided by Flutter. This decision was made based on ease of implementation, and allows for the application's routing structure to be defined in a single file. GoRouter was chosen over alternative, and similar packages based on the past experiences of the development team.

3.2.2.4 Displaying Cloud Server Data

When displaying data from the Firebase cloud server, the application makes use of streams that leverage the declarative properties of Flutter to provide a real-time display of information. Such use is particularly important considering that the processing server will have access and make changes to the same data as the application, and thus the application must be sure to present the most up-to-date version of the data.

For the cases when streams of data are not appropriate, the application makes use of futures to carry out a one-time read of the data within the cloud server. Such use is particularly appropriate in the case of media display, which would be network intensive if streams were used.

3.2.2.5 Consistent Appearance

When building interfaces, pre-configured widgets were used that included standardised styling for the application. The use of these widgets ensured that the appearance of the application was consistent across its various pages, despite the multiple developers contributing to its development.

3.2.2.6 Inspection

When the user starts an inspection, the system first gathers all of the vehicle's checkpoints, and for each, creates two pages. The first page displays the example image of the checkpoint found in Cloud Storage, demonstrating to the user what should be captured. The second includes a custom camera interface to allow the user to perform the capture.

During an inspection, the remediate page sequences through each of these views, and records a mapping of checkpoint to captured image path in its internal state. When all checkpoints have been captured, the user is shown a review page, which allows for them to review their captured image for a given checkpoint, and re-capture it if they so desire. In the event of a re-capture, the remediate page updates its mapping of captured image paths for the new image. When the user has reviewed and submitted their inspection, they are taken to the the submit page, and the uploading of the inspection begins.

During the upload process, the system creates a new object to model the vehicle inspection, and uploads this to Firestore. Then, each checkpoint from the inspection is mapped to a checkpoint inspection object and is initialized to reference the newly created vehicle inspection before being uploaded to Firestore. Next, a directory is created in Cloud Storage, and the inspection media for each checkpoint is uploaded, with the relevant Firestore document IDs being used as references. Finally, the processing status of the vehicle inspection document is updated to "pending", to alert the processing server that the inspection is ready to be processed.

3.2.2.7 Class Diagrams

Class diagrams were created for the main processes within the application, like inspection, to display the planned relationships between the different classes in the system. These can be found in **A.1 - Deliverable 1 Class Diagrams**. Furthermore, the planned navigation between the different pages in deliverable 1 can be found in **A.3 - Routing Diagrams**

3.3 Mock Dataset

An annotated dataset must be prepared for a task prior to training an object detector. [30] has long been the standard dataset used for object detection research, therefore, its accruement and annotation strategies were closely followed. The general pipeline set by [30] for making an object detection dataset is as follows.

1. Define the set of object categories
2. Harvest an abundant amount of images consisting of the objects in the object categories
3. Annotate the objects in the images with their corresponding labels and bounding box coordinates

Unfortunately, getting to a train depot to create our dataset took longer and was more difficult than expected; due to complications regarding transport, the high safety standards on-site, and the required approval from numerous independent organizations. Therefore, a mock environment was created as a contingency plan to ensure project completion.

The mock environment allowed the team to prepare for the accruement of the real dataset at the depot, allowing the team to anticipate any challenges that may occur in advance. Subsequently, implementation began before the depot visit, removing our dependency on the client to begin development.

3.3.1 Dataset Accruement

The sign categories and their respective placements were specifically chosen to provide the most contextual consistency between the location of the sign and the content of the sign. The selected set of signs was subsequently printed for use in the mock environment.

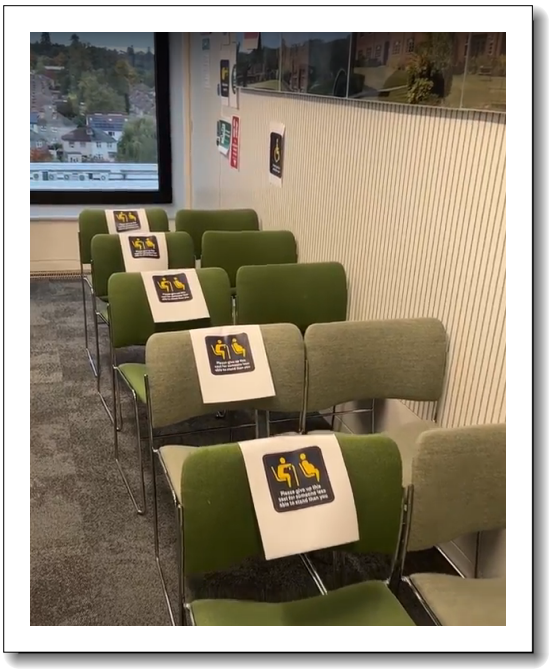


Fig. 3.13. An example of the department setup, with signs placed in positions that provide the most contextual consistency between sign position and sign content.

Once the sign categories were defined, the mock environment had to be created. The environment is laid out in a reproducible manner for testing and is similar to that of a train carriage.

The technical drawings of the trains provided by the client were used to recreate one side of a train carriage. This consisted of two platforms and one department. Additionally, all signs were specifically placed in positions where they provide context to their surroundings, e.g. seat-related signs should be placed on seats.

The platforms were replicated in the mock environment by placing the appropriate signs next to the doors. The department was then replicated by positioning seats next to a wall. The appropriate signs were placed on the seats and on a reflective area (mimicking a window) above the seats. An example image of the department is shown in **Figure 3.13**. For testing at later dates, the mock environment could easily be reproduced by using images, taken from the initial setup, as a guide.

Once the mock environment is created and the sign categories are defined, an image dataset is produced from the mock environment. This consists of harvesting an extensive catalogue of dissimilar images, with objects of interest in their natural scene. The images were taken in non-iconic object views (images with multiple objects in a canonical perspective), a standard set by [30]. The use of non-iconic images has been shown to be better at generalizing [46], and taking images in their natural scene should provide contextual information to the object detection models [30].

When making a dataset, it is important to have a roughly equal number of object instances for each object category. This mitigates the effects of unbalanced class categories on the performance of the object detection model. This was not a problem for the mock dataset as can be seen from **Figure 3.14**. The distribution of labels in the mock dataset is even, with the least populated classification category deviating from the average by 2.20% and being 3× the magnitude of the standard deviation.

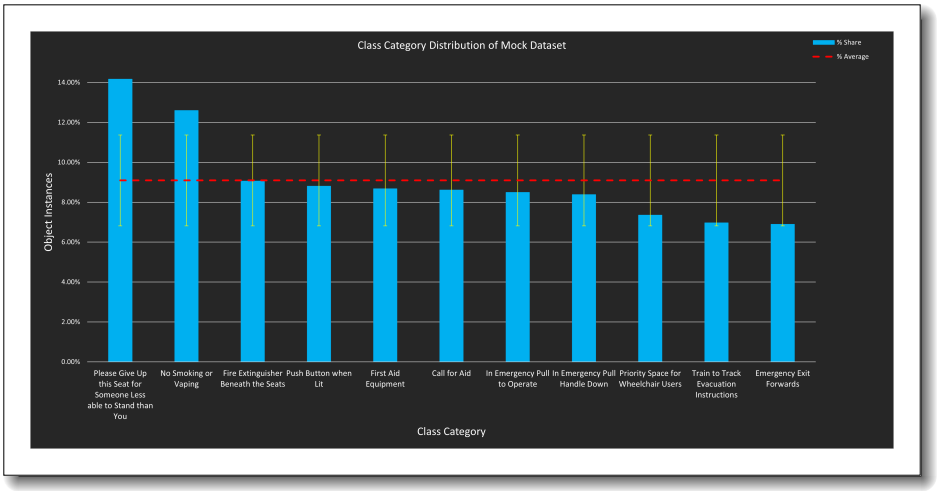


Fig. 3.14. A graph representing the distribution of each class category in the mock dataset, with the standard distribution represented by error bars deviating from the average. The mock dataset has a total of 466 images, with a total of 2,626 annotations. This comes out to an average of over 5 signs per image, proving the images were taken from non-canonical perspectives.

3.3.2 Dataset Annotation

LabelBox [26] was used to label each object in each image with their respective bounding box and classification. Consistent labelling standards were maintained by creating an instruction template (**B.1 - Dataset Instructions**) that each labeller in the team would have access to. Once all the images are annotated, each image is subsequently reviewed by a different member of the team, with any errors being logged and fixed. An example of a set of annotated

images can be seen in **Figure 3.15**. The entire annotation time took a little over 15 hours to annotate and review prior to the annotations being exported from LabelBox.

Fig. 3.15. A set of labelled images from the mock dataset. We see from the figure that the images were taken in non-iconic object views and in their natural scene, as advised by [30]. The exported annotations file contains each object instances classification, followed by its corresponding bounding box coordinates (labelled in COCO format (x, y, w, h)).

3.4 Processing Server

3.4.1 Object Detection

Object detection is the task of both object localization and classification. Common benchmarks for object detection models that are used in research are the Pascal visual object classes (VOC) [10] and Microsoft common objects in context (MS-COCO) [30] datasets.

There are two main types of object detectors: 1-stage and 2-stage object detectors [54, 21]. Two-stage detectors have specialized region proposals networks (RPN) that propose regions of interest (RoI) within the image. The RoI are subsequently passed to classification and localization heads to improve the precision of the bounding box coordinates and classify the objects within the regions [13].

Alternatively, one-stage detectors directly propose bounding box predictions from input images without specialized RPN. As such, one-stage and two-stage detectors both leverage a trade-off regarding inference speed and accuracy, with one-stage detectors opting to prioritise inference speed and two-stage detectors opting to prioritise object detector accuracy [21].

Object Detection Performance Metrics

Object detector performance is often evaluated using a set of performance metrics that are defined by using the Intersection over Union (IoU) to classify object predictions as a true positive (TP), false positive (FP) or false negative (FN).

The IoU is a measure of the overlap between two bounding boxes and is calculated using **Equation 1**.

$$IoU(B_1, B_2) = \frac{\texttt{area}(B_1 \cap B_2)}{\texttt{area}(B_1 \cup B_2)} \tag{1}$$

Once the IoU between two bounding boxes is calculated, a predefined IoU threshold, t, is used to classify each prediction [35].

- TP: a correctly localized ground-truth bounding box, formalized as a predicted bounding box having an $IoU > t$ with its respective ground-truth bounding box
- FP: an incorrect prediction of an object in an area where there is no object, formalized as a predicted bounding box with a $IoU < t$ for its respective ground-truth bounding box
- FN: when a ground-truth bounding box is undetected by the model
- TN: For the context of object detection, a TN is not defined

Once the TP, FP, and FN classifications are made for every bounding box prediction, a model's performance metrics can be calculated.

The precision and recall of an object detector offer key insight into the performance of the model. The precision, P, of an object detection model is a measure of the fidelity of the predictions [54, 55] and is calculated using **Equation 2**.

$$P = \frac{TP}{TP + FP} \tag{2}$$

Conversely, the recall, R, is a measure of the object detectors' ability to detect objects within an image [54, 55] and is calculated using **Equation 3**.

$$R = \frac{TP}{TP + FN} \tag{3}$$

An optimal detector will find all objects within an image ($R = 1$) with all predictions correctly detecting objects within the image ($P = 1$). This introduces a trade-off between precision and recall [35]. The f1-score is introduced to ensure a good balance is achieved between precision and recall by the object detector. The f1-score is calculated via the harmonic mean of precision and recall, therefore, providing a more complete metric than accuracy for the object detection task.

When comparing the performance of different object detectors on benchmark datasets, the mAP is the standard metric. The average precision (AP) is frequently used as a numerical measure of the performance of an object detector for a given class. To determine this value, the area under the curve (AUC) of a PR plot for a class is calculated. When the AP of every class category in the benchmark dataset is calculated, the mAP can be calculated as the mean AP for each class, with a $\texttt{mAP} = 1$ indicating a perfect detector [35].

In an attempt to improve object detector performance, object detectors are commonly used in tandem with techniques that aim to improve model attributes during learning or inference. The most prevalent techniques that were identified during the project are detailed below.

Non-maximum suppression (NMS)

Object detectors often produce several bounding box predictions for the same object. NMS is a post-processing technique that filters the undesired overlapping object predictions to just a single prediction per object [39, 54, 21]. NMS achieves this by removing all overlapping bounding box predictions that have an $\texttt{IoU} > \texttt{IoU}_{threshold}$, except for the prediction with the highest confidence score.

Feature pyramid networks (FPNs)

FPNs aim to improve the box-level scale imbalance problem, where objects with a wide diversity of scales (particularly small objects) prove difficult to detect [28, 34]. FPNs achieve this by creating multi-scale feature maps that are fed to RPNs, increasing the diversity in scale of region proposals and improving the average recall (AR) of the object detector [28].

Path aggregation networks (PANet)

PANet improves the feature-level scale imbalance problem that arises from the use of FPNs [31, 34]. When FPNs are placed within the model, they introduce an imbalance in low and high-level features, resulting in inconsistent predictions being made by the object detector [34]. PANet improves the imbalance by adaptive feature pooling [31], a technique that maps RoI to each layer of a FPN instead of associating a RoI to just one layer of the FPN [34].

3.4.2 Model Selection

Before selecting the appropriate model for the task, a wide variety of models were researched to ensure the optimal model is selected. The optimal model will have high detection performance, high temporal efficiency, and should be easily deployable for inference on images and potentially also videos. Both one-stage and two-stage object detection models were researched to identify the model that is the best fit for the task at hand. Below is a breakdown of the researched object detection architectures followed by a model comparison in table 1, used to explain why the chosen model was selected.

Faster R-CNN is a two-stage, region-based object detector [39] that provides an improvement in inference speed to its predecessors (R-CNN [13] and fast R-CNN [12]). Unlike its predecessors,

faster R-CNN uses a CNN RPN to predict RoI for a wider diversity of object sizes, mitigating the effects of the box-scale imbalance [39] and providing a speed-up to previous two-stage object detection models.

RetinaNet is a one-stage object detector that aims to improve the foreground-background class imbalance via the use of a focal loss [29, 34]. In most images, the background is often the most dominant feature, therefore, most predictions have "background" classifications leading to an imbalance [34]. These predictions are seen to be "simpler" to identify, therefore, the focal loss is a soft sampling technique [34] that gives a smaller weighting to these "simpler" predictions and gives more weighting to "harder" predictions [29]. RetinaNet also deploys FPNs [28] and uses a ResNet backbone [16], improving detection for a wider diversity of object sizes while mitigating the effect of the vanishing gradient problem via residual connections (further discussed in **Section - 4.5.2**).

At the time of writing, YOLOv7 is the latest iteration of YOLO one-stage object detectors [48]. The general pipeline for a YOLO [38] algorithm is as follows:

1. Pass the image through a CNN backbone to get a feature map
2. Combine the features using a FPN
3. Pass the features to heads for predictions to be made
4. Perform post-processing NMS to get the final predictions

YOLOv7 achieves state-of-the-art (SOTA) object detection accuracy with exceptionally fast inference speed. This is achieved by a plethora of architectural improvements upon previous iterations, including extended efficient layer aggregation (E-ELAN), model scaling, coarse-to-fine lead-guided assigners, and planned re-parameterization convolution [48].

E-ELAN takes advantage of the shortest gradient path to improve learning [48, 49]. Furthermore, model optimality is ensured by concurrently scaling the model's depth and width with the concatenation of model layers [48]. Due to the extreme depth of the model, the authors defined auxiliary, early-exiting heads that lie in the model of the network improving classification convergence [48]. Finally, the authors implemented re-parameterization techniques to improve the robustness of the model [48].

Transformer models such as DeTR [4] and Swin Transformers [32] were also briefly considered, however, were not deemed necessary due to their comparatively low performance on a wide diversity of object scales and their high computational requirements leading to low frames per second (FPS), as shown in **Table 1**. ViTs are further discussed in **Section - 4.5.2**.

Model	Backbone	Benchmark	AP_{50}	Batch 1 FPS
Faster-RCNN	ResNet-101-FPN+	MS-COCO	55.7%	20
RetinaNet-101	ResNeXt-101-FPN	MS-COCO	61.1%	-
YOLOv7-E6	-	MS-COCO	73.5%	56
DeTR	ResNet-101	MS-COCO	63.8%	10
Swin Transformer	RCN-N	MS-COCO	71.8%	11.6

Table 1: A breakdown of object detector performance on the MS-COCO object detection benchmark [30]. All FPS measurements are from [48] where a V100 GPU was used to calculate the measurements. As a result, no RetinaNet-101 measurement is taken into account, however, it should be noted that measurements above 30 FPS were observed by different papers using a different setup to YOLOv7.
Sources: [48, 29, 32]

Video feeds place emphasis on a requirement for high temporal efficiency, therefore one-stage models were deemed as the more appropriate model type early on in the research process. Due to the exceptional model inference speeds coupled with a high AP score on the COCO [30] benchmark in table 1, the decision was made to opt for the YOLOv7 model. When investigating the types of YOLOv7 model, the YOLOv7-E6 model [48] was subsequently identified as the best model for the application, using a similar line of reasoning. The additional ease of implementation using the HuggingFace [52] platform further showcases the model's superiority over alternatives.

3.4.3 YOLOv7 Training and Evaluation

The pre-trained YOLOv7-E6 [48] model was fine-tuned on the mock dataset using the HuggingFace [52] implementation. To ensure the required computational resources for the training of the model were available, the model was trained on the University of Southampton's Alpha Cluster.

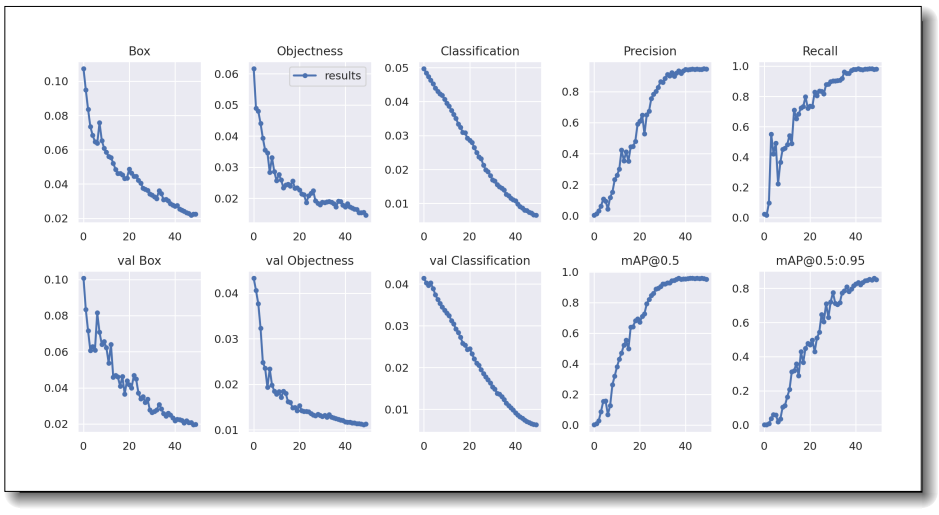


Fig. 3.16. Learning curves for the training of the YOLOv7 object detector on the mock dataset. It can be observed that the model has not fully converged at the end of model training, however, this model is sufficiently trained for the mock implementation.

In order to fine-tune the model, the dataset had to first be prepared for training and evaluation. A standard data split was employed, where 80% of the dataset is used for training and the remaining 20% is divided equally into a validation and a testing set. Additionally, the

annotation format was modified from the COCO format, (x_1, y_1, w, h), to the YOLO format, $(x_{centre}, y_{centre}, w, h)$, accepted by the HuggingFace implementation.

The predefined hyperparameter values for the model were empirically tuned such that they provide optimal training curves and loss convergence, shown in **Figure 3.16**, with the final assortment of parameter values listed in **C.1.1 - Mock Object Detection Model**.

After model training, the f1-plot is used to identify the optimal confidence threshold for model deployment. From **Figure 3.17**, $\overline{\tau}_{\texttt{conf}}$ is identified to be $\overline{\tau}_{\texttt{conf}} = 0.554$.

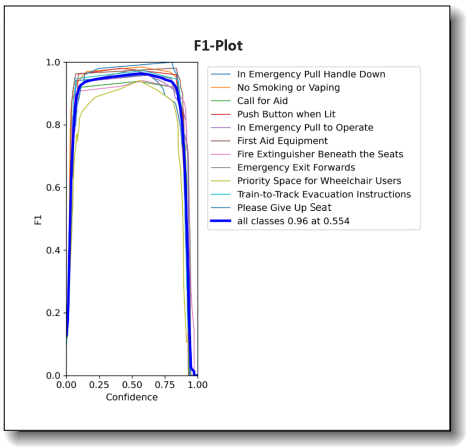


Fig. 3.17. The F1 plot produced during model training with a non-optimal confidence threshold value ($\tau_{\texttt{conf}} = 0.001$). This plot is used to identify the optimal value for the confidence threshold: $\tau_{\texttt{conf}} = 0.554$.

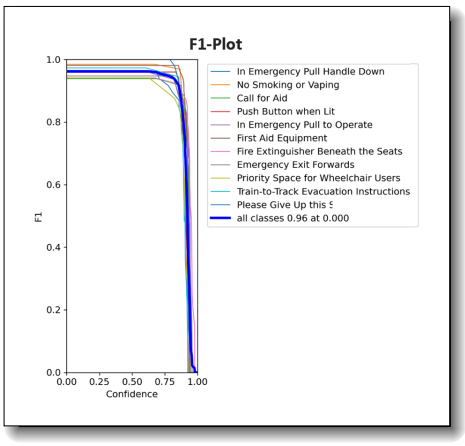


Fig. 3.18. The F1 plot produced during model testing with an optimal confidence threshold value ($\tau_{\texttt{conf}} = 0.554$). The area under the F1 plot is clearly increased to a value $\backsim 1$, denoting a model with high precision and recall.

Fig. 3.19. The F1 plots of the YOLOv7 object detector during training and testing. The optimal classification threshold is determined from the F1 during training plot in **Figure 3.17**. This optimal classification threshold is used to produce the F1 plot in **Figure 3.18**.

The f1-score is the harmonic mean of precision and recall. During training, the confidence threshold is set to 0.001, therefore, the recall plot is in its optimal state and the precision plot is in its worst state. The precision and recall plots in **Figure 3.22** are used to ensure that the confidence threshold is selected to a value that results in the best balance between the precision-recall trade-off.

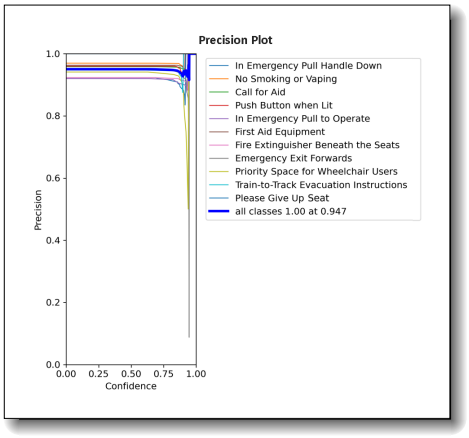


Fig. 3.20. The precision plot produced during testing with an optimal confidence threshold value ($\tau_{\texttt{conf}} = 0.554$).

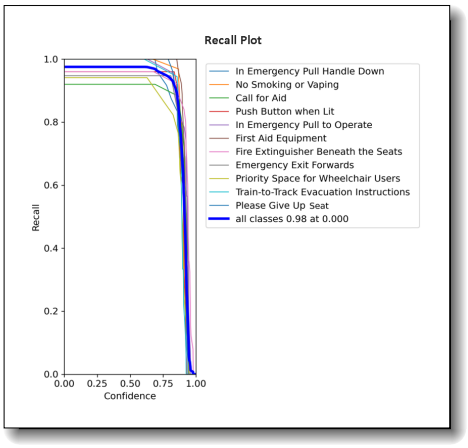


Fig. 3.21. The recall plot produced during testing with an optimal confidence threshold value ($\tau_{\texttt{conf}} = 0.554$).

Fig. 3.22. The precision and recall plots during testing. Using these plots, it can be observed that the classification threshold value identified from the F1 plots is optimal.

The final precision-recall plot provides the best insight into the models' performance. The aim is to minimize the precision-recall trade-off as much as possible, therefore, if increases in recall result in a negligible reduction in precision, then the model has high performance. The average precision and the area under the precision-recall curve are common ways to summarise such a plot. The model has a $\texttt{mAP} = 0.948$ indicating that the model is high-performing.

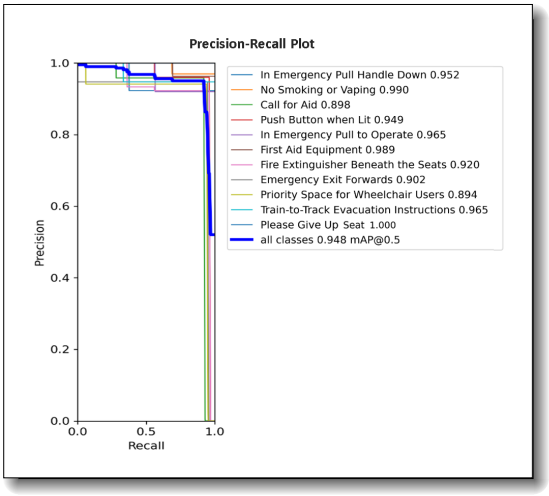


Fig. 3.23. The precision-recall plot of the object detector, produced during testing. Increases in recall seem to have minimal effect on precision. This is the case even until the recall value $\backsim 1$.

The confusion matrix in **Figure 3.24** reveals the model to have low confusion between object categories, reinforcing the model's high precision calculation. Upon close inspection of **Figure 3.24**, it is observed that unidentified objects within an image occur infrequently, reinforcing the model's high recall calculation.

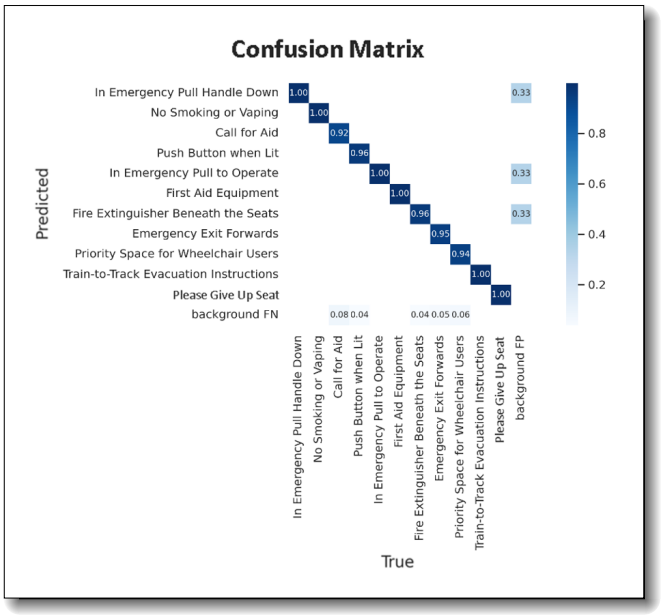


Fig. 3.24. The confusion matrix of the object detector, produced during testing. There is little to no confusion between class categories indicating a high performing model.

The YOLOv7 model has a high inference speed per image on the alpha cluster, $\backsim$ 20ms, however, this performance deteriorates when the model is deployed to systems with lower compute resources. During inference, the model provides object classification and localization labels, allowing processed figures to be produced and examined.

3.4.4 Server Workflow

The processing server is required to analyse the images collected by the application during an inspection and produce a report detailing the conformance status of the train vehicle. For deliverable 1, conformance status is dependent solely on sign presence.

To achieve this functionality, the server adopts a simple workflow, as shown in **Algorithm C.1**. This workflow begins with the server periodically checking the cloud server for unprocessed inspections. For each identified inspection, the server gathers all associated checkpoints and their respective inspection image. The image of each inspection checkpoint is then passed into the object detector detailed in **Section - 3.4.3**, with the output cross-referenced with the list of expected signs defined in the checkpoint to determine their conformance status. Finally, the conformance status of the inspection is updated based on the status of each of its checkpoints, and the results are posted to Cloud Firestore.

3.5 Testing

3.5.1 Scenario Testing

To assess the performance of the application, its functionality was tested against each of the scenarios defined in **Section - 3.2.1.10**. The results of this testing are summarised in **Table 3.1**, with any differences in the actual outcome of the application noted, and its cause identified. Additional analysis of the application was carried out in the form of storyboard testing, where the mock-up designs, and produced system are compared and differences justified. This storyboard testing is presented in **D.1 - Deliverable 1 Storyboard Testing**.

Table 3.1: Deliverable 1 Scenario Testing

Scenario	Actual Outcome	Notes
Scenario 1		
Scenario 1 - Inspection	The app performed as described in the scenario.	No notes
Scenario 1 Alternative Flow 1 - Recapturing Footage in Walkthrough	The app performed as described in the scenario.	No notes
Scenario 1 Alternative Flow 2 - Recapturing Footage in Review	The app performed as described in the scenario.	No notes
Scenario 1 Alternative Flow 3 - Reviewing Footage	The app performed as described in the scenario.	No notes
Scenario 1 Alternative Flow 4 - Cancelling Inspection	The app performed as described in the scenario.	No notes
Scenario 1 Alternative Flow 5 - Considering Cancellation	The app performed as described in the scenario.	No notes
Scenario 2		
Scenario 2 - Viewing Inspection	The app performed as described in the scenario.	No notes
Scenario 2 Alternative Flow 1 - Viewing Inspection Pending Processing	The app performed as described in the scenario.	No notes
Scenario 2 Additional Flow 1 - Viewing Conforming Checkpoint	The app performed as described in the scenario.	No notes
Scenario 2 Additional Flow 2 - Viewing Non-Conforming Checkpoint	The app performed as described in the scenario.	No notes
Scenario 2 Additional Flow 3 - Viewing Checkpoint Pending Processing	The app performed as described in the scenario.	No notes

Scenario 3		
Scenario 3 - Viewing Status	The app performed as described in the scenario.	No notes

3.5.2 Integration & Field Testing

To assess the integration of AutoSign's components, and its performance in a real-world situation, field testing was performed. In this testing, the mock environment was established, as defined for the mock dataset, and a number of test scenarios were declared. In each scenario, an inspection of the mock environment was carried out, with a varying number of signs missing. For all test cases, the system was able to function correctly and cohesively as a single unit, and the processing server was able to correctly report on the state of the environment. Figure 3.25 showcases the results of an example scenario, where-by one of the signs within the checkpoint was missing, and correctly identified by the system.

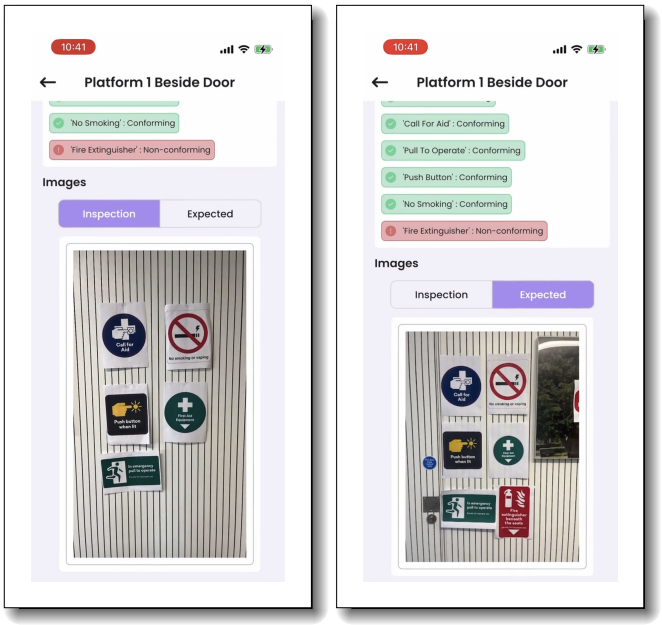


Fig. 3.25. Screenshots showing the detection of a missing sign within a checkpoint

SECTION 4

Deliverable 2

4.1 Cloud Server

4.1.1 Data Model

To support the required functionality of the deliverable 2 system, the data model of the cloud server must be updated to enable the remediation of signs, video-based checkpoints, and damaged signs. The following sections describe how this is achieved for both the database and file store.

4.1.1.1 Database

Figure 4.1 shows the updated organisation of data within the Firestore database. Firstly, vehicle and checkpoint objects now model their conformance status, to reflect that the most recent inspection of a vehicle is no longer representative of its current status, as a remediation may have since occurred. Additionally, the conformance status field now has values of “conforming", “missing", or “damaged", to support the new capabilities of the processing server.

Next, a capture type field with values of “photo" or “video" is added to checkpoint related objects to model the type of media that must be captured during an inspection of the checkpoint. Additionally, the definition of signs within checkpoint related objects is re-defined as a list of maps, where each sign is stored as a map of its name, conformance status and an ID. The purpose of the ID field is to allow for repeated instances of a sign within a checkpoint to be distinguished during remediation.

Finally, objects are defined to model the remediation of a single sign, and a group of remediations within a single vehicle, with remediation occurring by one of two actions - “cleaned" or “replaced". Additionally, the checkpoint object defines a field to model its current remediation status, with values of “none", “partial" and “complete".

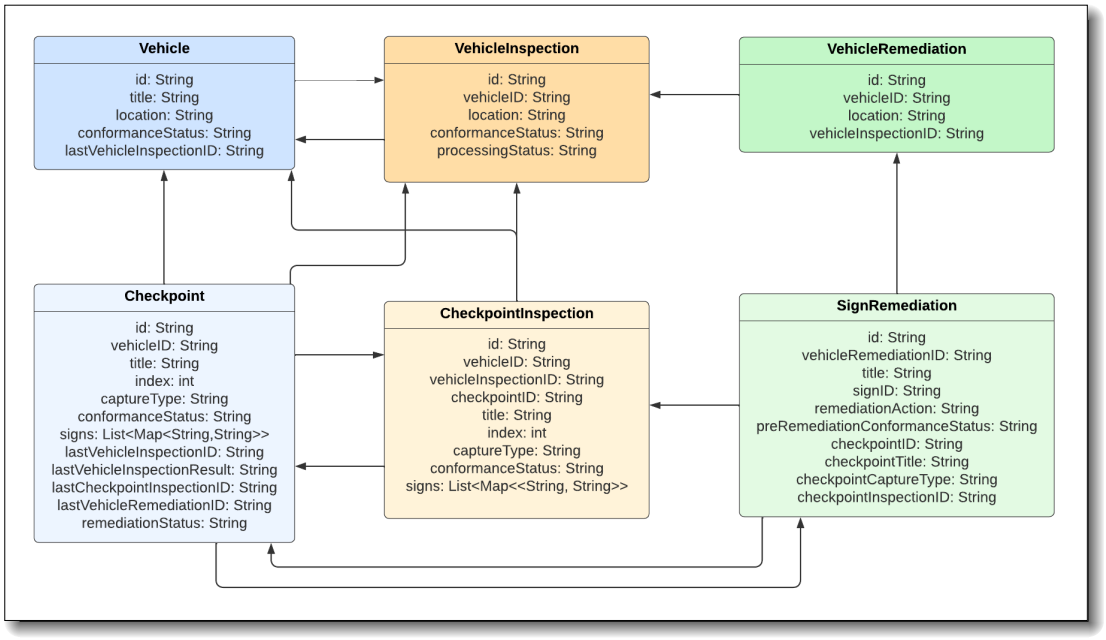


Fig. 4.1. Deliverable 2 database model

4.1.1.2 Storage

Figure 4.2 shows the updated organisation of files within the Cloud Storage Drive. Again, the drive contains one directory for each vehicle, but changes have been made to how this vehicle's data is stored. Two pieces of media are now stored for each vehicle checkpoint - the first being a showcase image of the checkpoint, and the second being an example inspection media, which could be an image or video based on the checkpoint's capture type. Additionally, the media for each checkpoint inspection is now an image or video based on the capture type. Finally, a new directory is defined to store the vehicle remediations associated with the vehicle, with one image stored for each sign remediation.

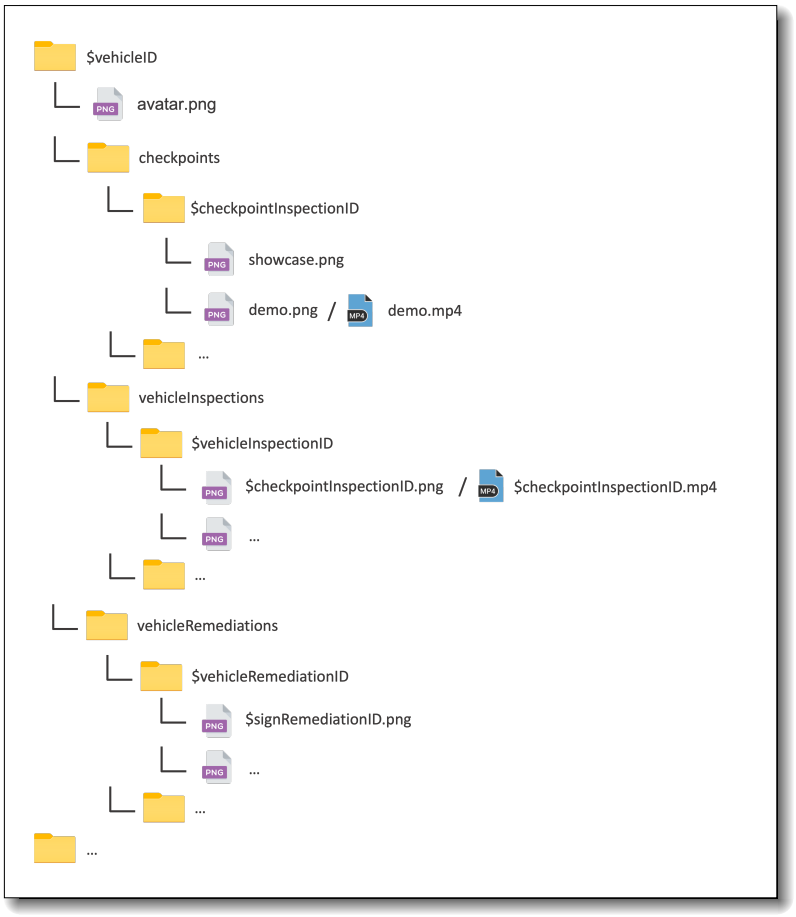


Fig. 4.2. Deliverable 2 storage model

4.1.2 Integration

To integrate the updated cloud server with the system, the same sssasamethodology and architecture implemented in deliverable 1 can be used (3.1.4). The application and processing server are required only to re-define their controller's and model classes to support the cloud server's updated data model.

4.2 Application

4.2.1 Potential Expansions

Listed below are a selection of potential expansions that were considered for development at the start of deliverable 2, including a justification of their usefulness.

- **Purchasing of new signs**: Showcasing how the system can be used to purchase new signs demonstrates to potential value of **AutoSign** to the client. Note, however, as **AutoSign** is a proof-of-concept, the purchasing of signs would be simulated, would not actually carry out an order.
- **Remediations**: This is an essential feature which our client has expressed to us so we will probably implement. This allows the user to log the remediations which have been made into the system.
- **Updating Inspections Manually**: It is the hope that the system developed should have a low error rate. However, in the event that a sign is classified incorrectly, such as being processed as conforming when it is actually damaged, then allowing the user to alter the inspection results themselves manually would fix this.
- **Desktop Application**: So far, our system is designed to be used by the inspectors on their mobile phones. However, a desktop application should add more administrative functions where information regarding larger groups of trains can be viewed.
- **Assisted Capturing**: Many of the issues with the processing of footage comes from the user capturing poor footage. As such, having additional facilities like orientation prompts or overlaying the expected media on top of the camera preview could help solve these issues.
- **Defining your own walkthrough process**: As this is a POC system, it currently only works with a single train carriage design so adding this feature would help further demonstrate the viability of our solution.
- **Physical mapping of train status**: This would emulate some of the documentation we were given detailing the signs and their location within given train carriages which would replace the current, list-based, approach for displaying an inspection. This would hopefully make our system easier to use for the current inspectors who are used to this documentation.

After consideration, the app development team decided to focus on adding the functionality for remediations, the purchasing of new signs, and the udpating of inspections manually. This is because several of the other ideas would require substantial modifications to the current structure of the application which may not be possible accounting for the deadlines of this project. Furthermore, all of these should provide substantial value to our client and users and are prioritised higher according to the MoSCoW prioritisation of our requirements at the beginning of this project.

4.2.2 Design

4.2.2.1 Updated Application Structure

To support the additional functionality of the deliverable 2 system, the following additional pages are defined within the application:

- **Remediate Page**: Allows the user to purchase new signs, and log the remediation of non-conforming signs.

- **Remediations Page**: Allows the user to view a history of remediations carried out on a vehicle.
- **Status**: Presents a report on the vehicle's status, which is a compilation of its most recent inspection, and any remediations that have occurred since this inspection.

4.2.2.2 Updated Use Case Diagram

Figure 4.3 showcases the updated use case diagram for the deliverable 2 application. Here, the functionality for remediations is included. The remediate use case relies on several of the same use cases as the inspect use cases such as capturing and uploading footage. Similarly, the information contained in a checkpoint breakdown such as sign conformance status and viewing checkpoint capture data is functionality used by both the "View Inspections" use case as well as the "View Remediations" use case. Additional use cases have also been made for the purchasing of new signs where it is necessary to first checkout and add items to the cart. Finally, the overall structure of the application has remained consistent with all of the functionality extending from the profile page for a given train carriage.

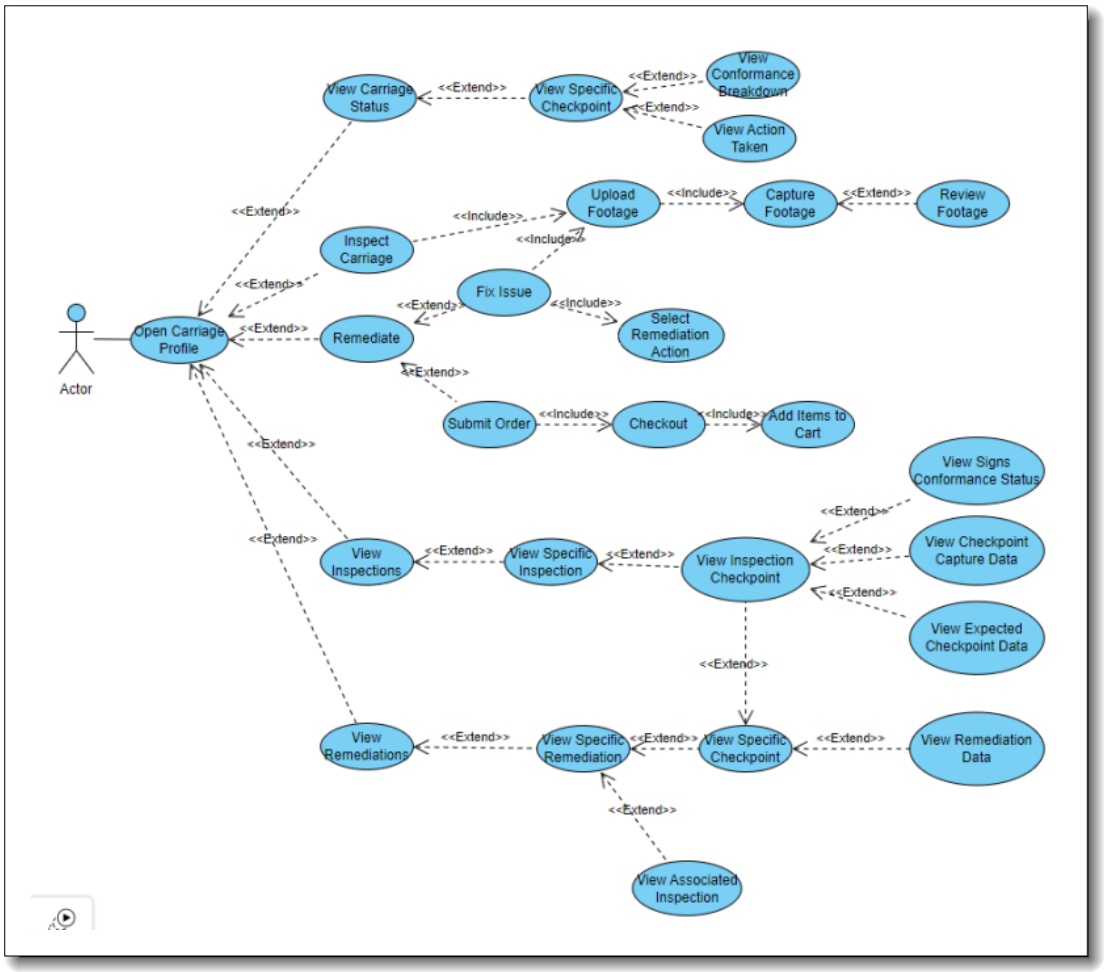


Fig. 4.3. Updated Use Case Diagram for Deliverable 1

4.2.2.3 Updated Activity Diagrams

Similarly to deliverable 1, activity diagrams have been created for the main processes within the application. Again, no activity diagrams are included for application features which focus on simply the presentation of information, rather than some distinct workflow.

Firstly, after reviewing the deliverable 1 application, it was decided that the review of each checkpoint immediately after capturing it within an inspection was unnecessary. As such, this step was removed and the user was instead given only the option to review the all footage at the end of the inspection process. This updated workflow is presented in **Figure 4.4**. Additionally, the process of logging a remediation after it has been made in the train carriage is represented in **Figure 4.5** where-by the user captures footage of the remediation and reviews it before uploading it to the server in a similar manner to the inspection process. Finally, the activity diagram in **Figure 4.6** shows-cases how the simulation of purchasing signs will be carried out within the application. This is done in a very similar way to most online shopping services, increasing the familiarity of the system to the user, allowing users to add items to a cart, review the cart and then submit their order.

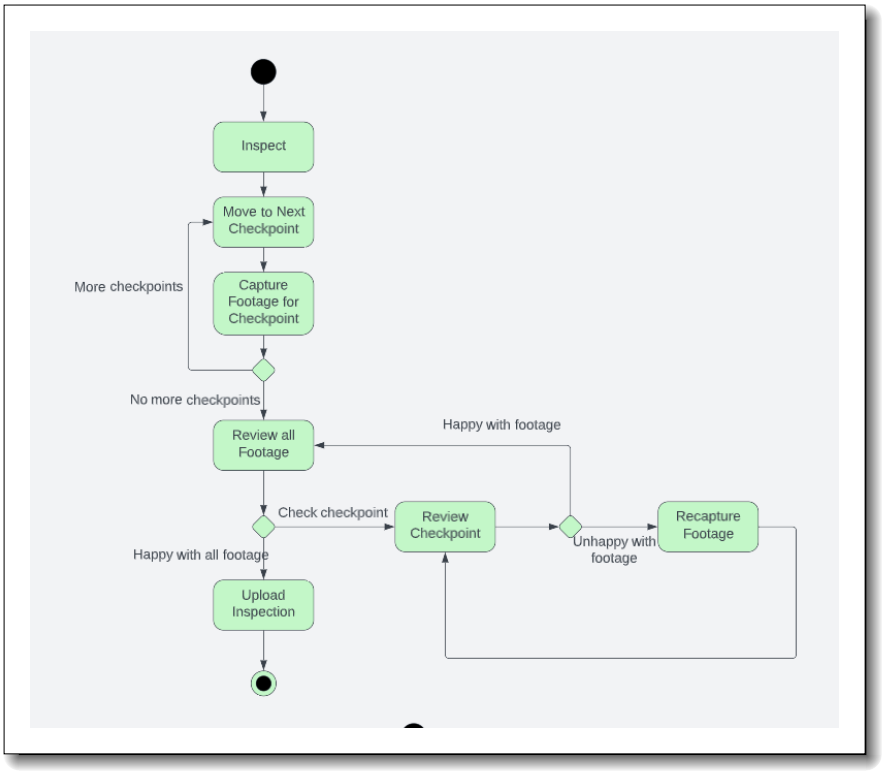


Fig. 4.4. Updated activity diagram for inspections in deliverable 2

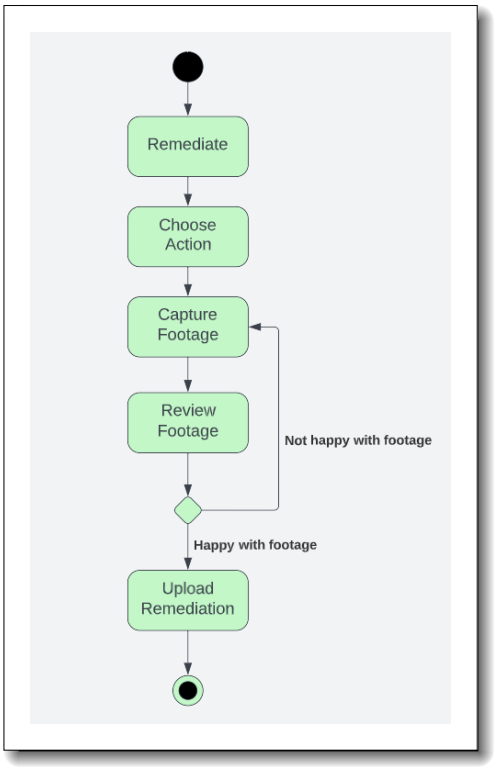


Fig. 4.5. New activity diagram for remediations in deliverable 2

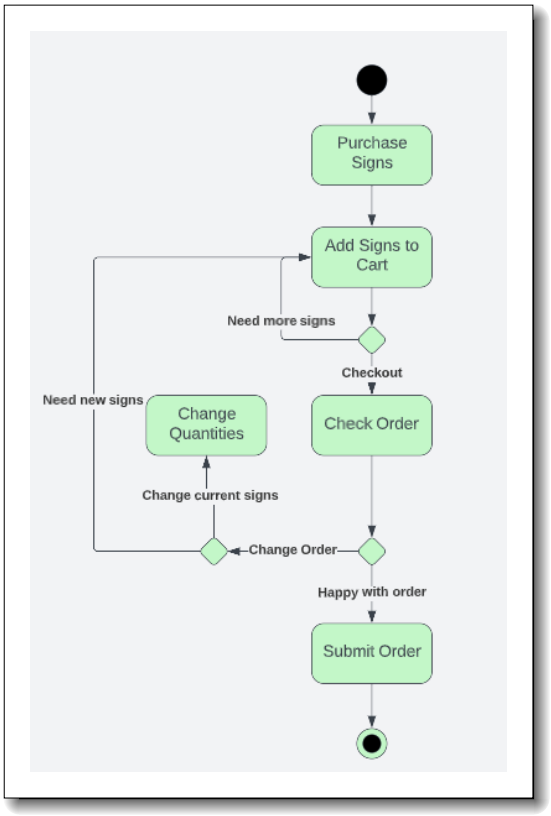


Fig. 4.6. New activity diagram for purchasing signs in deliverable 2

4.2.2.4 Updated Mock-Ups

4.2.2.5 Updated Profile Page

Figure 4.7 showcases the updated mock-ups for the profile page, which allow the user to access the remediate, and remediations sections of the application.

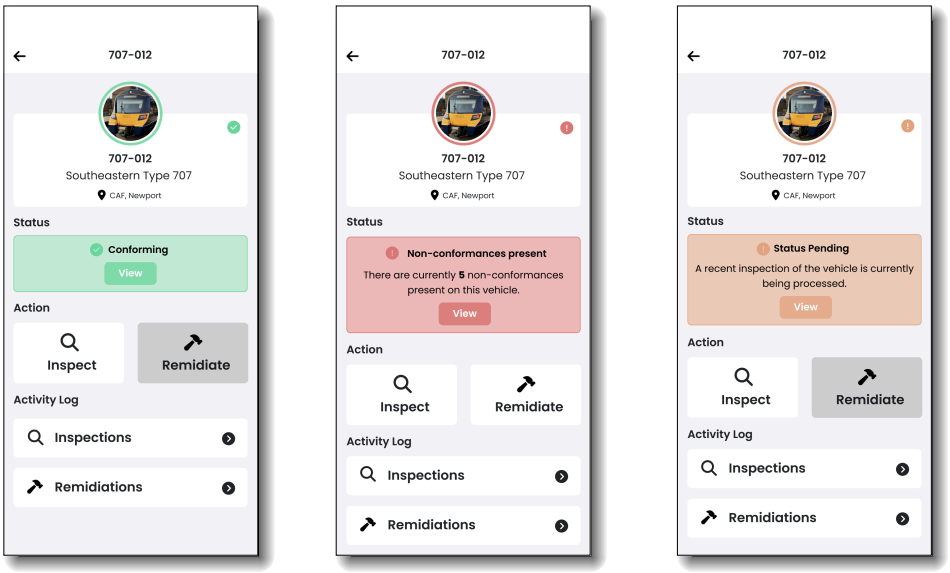


Fig. 4.7. Sample updated application profile page mock-ups

4.2.2.6 Status Page

The mock-ups in Figure 4.8 show the dedicated status page for a train carriage which combines the most recent inspection about a train carriage as well as any remediations which have been logged since then.

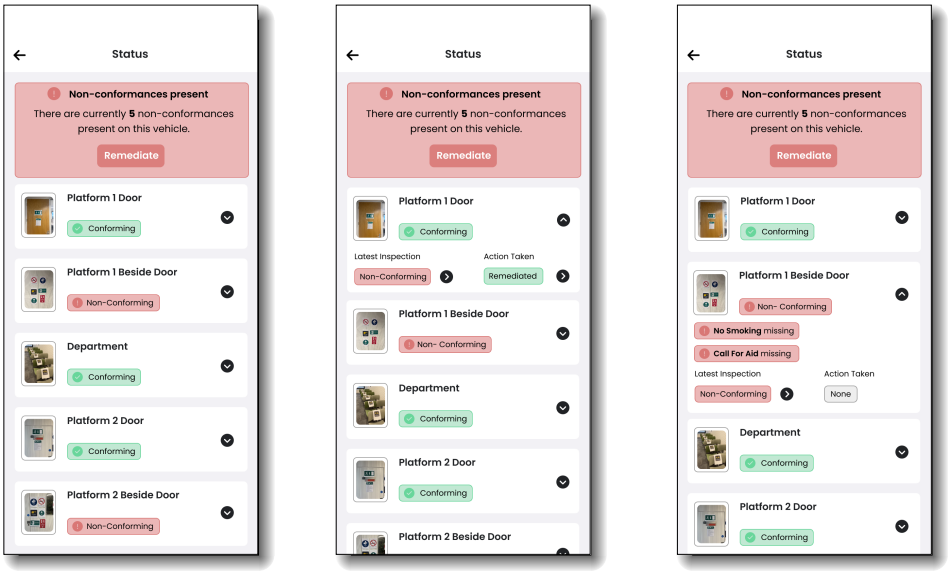


Fig. 4.8. Sample application status page mock-ups

4.2.2.7 Remediate Page

Figure 4.9 showcases the mock-ups for the remediate page, which allow the user to view a list of non-conformances present in the vehicle, purchase replacement signs, and perform remediations. When performing a remediation, the user is taken through a multi-stage process, where they are asked to select the remediation action, capture an image of the fix, and review this capture.

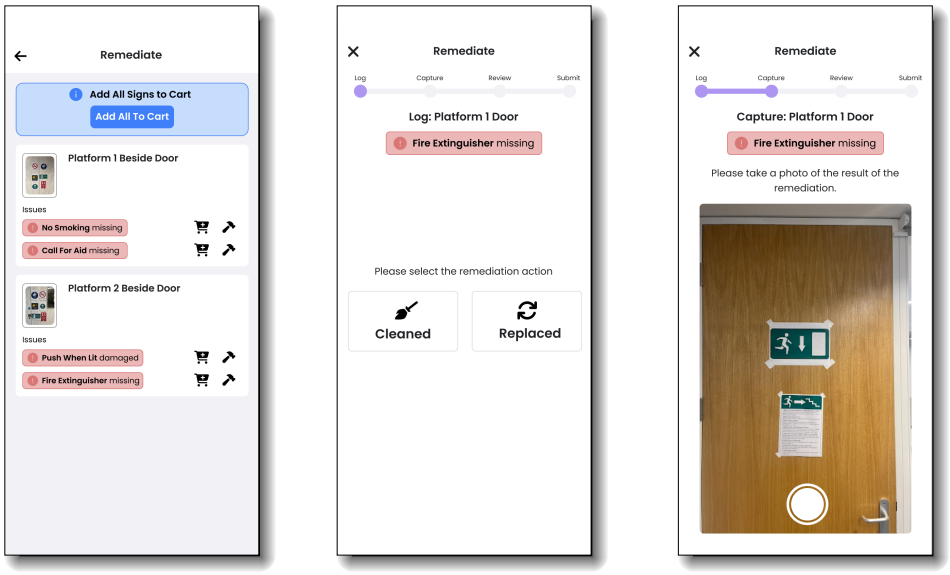


Fig. 4.9. Sample application remediate page mock-ups

4.2.2.8 Checkout Page

The mock-ups in Figure 4.10 demonstrate the interfaces presented when purchasing new signs, which are off of online shopping services by providing a "shopping-cart" which users can add items to before reviewing their order and "checking out". It is the hope that by designing the UI this way, it will automatically be intuitive for new users.

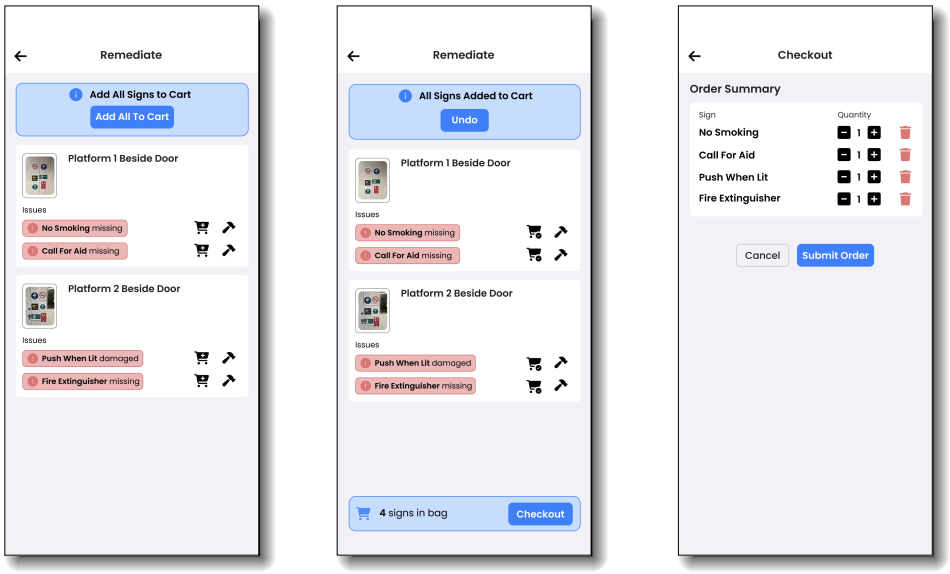


Fig. 4.10. Sample application Checkout page Mock-ups

4.2.2.9 Remediations Page

The mock-ups shown here in **Figure 4.11** show a similar layout to the mock-ups in **Figure 3.11**, where remediations are sorted by date, However now, the remediation breakdown consists of only checkpoints which contained non-conformances. Furthermore, there is also a tab in the checkpoint breakdown where the user can view the footage captured as evidence of a non-conformance in a checkpoint being remediated.

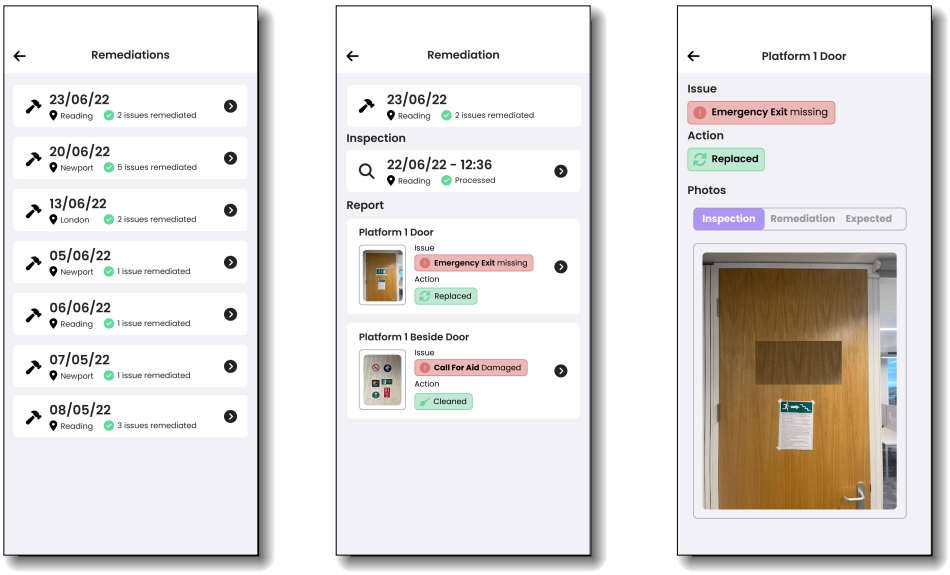


Fig. 4.11. Sample application remediations page mock-ups

4.2.2.10 Updating Inspections Manually

The mock-ups shown in **Figure 4.12** expand upon the mock-ups in **Figure 3.11**, allowing the user to open up a dialog box containing a drop down menu where they can manually update the conformance status of a sign.

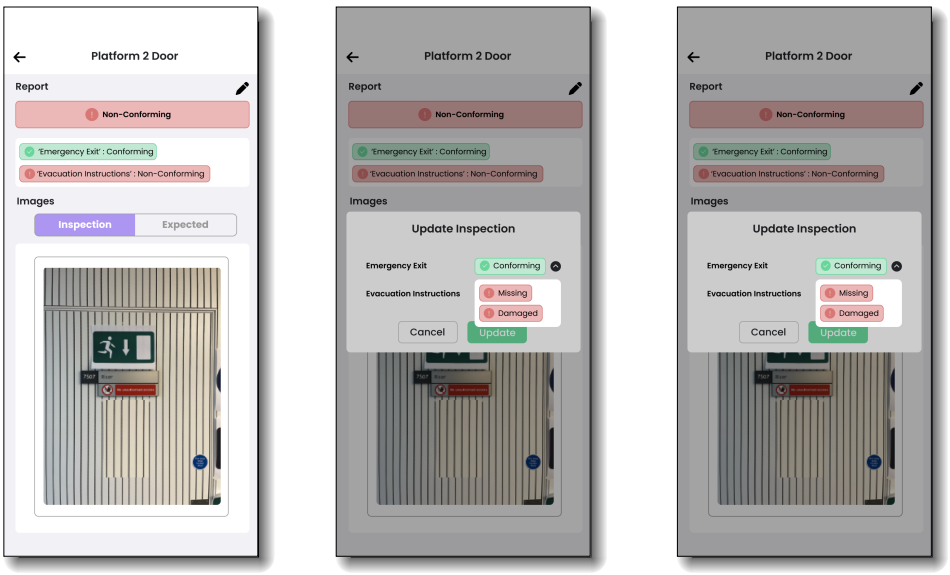


Fig. 4.12. Sample application mock-ups for the updating of an inspection

4.2.2.11 Additional Scenarios

In accordance with the mock-ups defined above, a number of scenarios are defined to model typical use of the extended system and expected performance, which are listed below. In testing, these scenarios can be used to ensure the application supports all of the required functionality.

Scenario 4 - Updated Inspection Process
This scenario is identical to Scenario 1 in deliverable 1 however step 6, and all alternative flows from step 6 are no longer present. Note that this means that the only repeated steps are steps 4 and 5 where the user is presented with a template image for the current checkpoint before they user actually takes the photo for that checkpoint.

Scenario 5 - User views the status of a train carriage

1. User enters ID for a train carriage and is presented with the profile page for that train carriage.
2. User selects option to view the status of the current train carriage.
3. User is presented with a view of each checkpoint in the train along with it's conformance status

Additional Flow 1 after step 3 - User views a checkpoint from the status page where there is a non-conformance which hasn't been remediated

- User selects the checkpoint they wish to view with non-conformances present.
- User is presented with the list of signs within the given checkpoint which were non-conforming as well as a prompt confirming that no action has been taken to remediate these non-conformances since the last inspection.

Additional Flow 2 after Step 3 - User views a checkpoint from the status page where there was a non-conformance which has since been remediated

- User selects the checkpoint they wish to view which had non-conformances present.
- User is presented with a prompt stating that non-conformances were present for that checkpoint in the last inspection as well as a prompt stating that an action has since been taken to remediate this meaning the checkpoint is now conforming.

Scenario 6 - User purchases new signs for the purpose of remediating non-conformances present within the last inspection

1. User enters ID for a train carriage and is presented with the profile page for that train carriage which is non-conforming.
2. User selects option to perform a remediation for the latest train inspection which was non-conforming.
3. User is presented with a list of all of the checkpoints which had non-conformances present in the latest inspection including which specific signs were non-conforming.
4. User then selects the option to add all of the signs to the cart.
5. User selects the option to checkout and is then taken to the order summary screen.
6. User selects the option to submit their order. They can then return to the carriage profile page after their order has finished submitting.

Alternative Flow 1 at Step 4 - User decides they don't want to log any remediations or purchase any new signs at this time

- User selects the back option returning them to the train carriage profile page.

Alternative Flow 2 at Step 4 - User purchases some signs for the purposes of remediating the non-conformances within a checkpoint

- User then selects the shopping cart icon by the signs they wish to purchase.
- Process continues again from step 5

Alternative Flow 3 at Step 4 - User reconsiders buying all of the signs and instead purchases some signs for the purposes of remediating the non-conformances within a checkpoint

- User then selects the option to add all of the signs to the cart.
- User then selects the option to undo this action.
- User then selects the shopping cart icon by the signs they wish to purchase
- Process continues again from step 5

Alternative Flow 4 at Step 5 - User decides they need to change their order at the order summary screen

- User selects the option to checkout and then adjusts the quantities of signs they need accordingly by adding more signs, removing excess signs or removing orders for specific signs completely.
- Process continues from step 6

Alternative Flow 5 at Step 6 - User reconsiders their order and decides they don't want to order at this time

- User selects the option to cancel their order returning them to the remediate page.
- User selects the option to go back returning them to the train profile page.

Alternative Flow 6 at Step 4 - User has remediated a non-conformance and logs it in the system although struggles to photograph it correctly the first time.

- User then selects the hammer icon by the non-conforming sign for which they have performed a remediation.
- User then selects the action they have taken within for their remediation which will be replacing the sign or cleaning the sign.
- User then takes a photo for the remediation they have carried out.
- User is presented with the photo they've taken and selects the option to retake their photo.
- User then takes another photo of their remediation.
- User then confirms their happy with the photo by submitting their remediation
- User selects to finish after the remediation has been uploaded to the cloud server

Scenario 7 - Continues after Step 4 in Scenario 2 in Deliverable 1 - User updates the conformance status of a sign manually after finding out that the system has processed it incorrectly

1. User selects the checkpoint they wish to view.
2. User is presented with the list of signs within the given checkpoint as well as their associated conformance status.

3. User selects the pencil icon and is presented with a list of all of the signs within the given checkpoint as well as an associated drop-down menu with each signs current conformance status.
4. User uses the drop-down menu to update the conformance status to the correct status.
5. User selects the option to update which returns them to the checkpoint viewing menu.

Alternative Flow 1 at Step 4 - User decides they no longer need to update the conformance status for any signs within the checkpoint

- User selects the option to cancel which returns them to the checkpoint viewing menu.

Alternative Flow 2 at Step 4 - User incorrectly updates the conformance status of a sign and rectifies it

- User uses the drop-down menu to update the conformance status.
- User realizes their mistake, and uses the same drop-down menu to change the conformance status back to what it was before.
- Process continues from step 5.

Scenario 8 - User views a remediation they have logged in the system

1. User enters ID for a train carriage and is presented with the profile page for that train carriage.
2. User selects option to view a list of existing remediations for the train carriage.
3. User is presented with a list of remediations sorted by date for the given train carriage and selects the one they wish to view.
4. User is presented with the list of checkpoints for the train carriage where each one has it's associated non-conformance and remediation.
5. User selects the checkpoint they wish to view.
6. User is presented with the issue detected within that checkpoint as well as the action taken during the remediation.
7. User views the footage captured during the initial inspection where a non-conformance was detected.
8. User views the expected image for that checkpoint.
9. User selects the remediation option to view the footage captured for the remediation after whatever issue was present was resolved.

4.2.3 Implementation

4.2.3.1 Updated Class Diagrams

The class diagrams, which can be viewed in **A.2 - Deliverable 2 Class Diagrams** have been updated for the new status and remediate pages. The controllers class diagram has been updated from deliverable 1 for the new shop and remediation controllers required and the new relationships are shown. The routing diagram was also updated for this deliverable which can also be found in **A.3 - Routing Diagrams**.

4.2.3.2 Transitioning to Video

To support the addition of a video-based checkpoint, additional interfaces are defined to handle the capture and display of videos within the application. At any point where a checkpoint capture or display is required, the system first checks the capture type of the checkpoint before displaying the appropriate interface.

4.2.3.3 Purchasing New Signs

To purchase new signs, the user can navigate to the remediate page, where they are shown a list of non-conforming signs within the vehicle. From this page, toggle buttons can be used to add particular signs to the cart, or add all signs to the cart at once using the convenience button at the top of the screen. The contents of the cart are defined by a list of sign names stored within the internal state of the page, with this list being passed onto the checkout page when the user selects to checkout. The checkout page then condenses this list, such that signs with the same name are grouped together under a single quantity value. The user is also able to configure the quantity of each sign from within this page, before submitting their order. As no connection is made to a vendor, no further action is taken with this order information, and the user is returned to the remediate page after submission.

4.2.3.4 Remediation

As mentioned previously, when in the remediate page, the user is shown a list of the currently non-conforming signs within the checkpoint. In addition to adding a sign to the cart, the user can also select to remediate a sign, which results in them being taken through a three-page process.

Firstly, the user is asked to select the action taken to remediate the sign - “cleaned” or “replaced”. In the next page, the user is asked to capture an image of the result of the remediation. Finally, the user is asked to review their capture, and is able to re-take if they so desire. During this process, the internal state of the remediate page records the selected remediation action and path to the captured image. When the user confirms their capture, they are taken to the submit page, and the uploading of the sign remediation begins.

During the upload process, the system first creates an object to model the remediation of the sign, and associates it with the vehicle remediation object from the current day. If a vehicle remediation object does not exist for the current day, one is created, uploaded, and referenced by the sign remediation object. The sign remediation object is then uploaded to the Firestore database, and its image uploaded to Cloud Storage using the relevant document IDs.

4.3 Real World Dataset

The team travelled to the CAF Newport depot, where a single train carriage of a Transport for Wales 197 trainset had been prepared. The state of the train during the visit was taken as the gold standard, from which an image dataset was to be acquired. The carriage consisted of two departments, two platforms and a single toilet, with signs placed by the client in predefined positions throughout the carriage.

4.3.1 Dataset Accruement

The image dataset at the train depot was acquired using the same standards used in the mock dataset (**Section - 3.3.1**), with images taken of objects in their natural scene and in non-iconic object view.

When visiting the depot, there was only one train carriage available to the team with signs placed according to regulations. As such, there was an intrinsic imbalance in the sign categories (a naturally occurring imbalance of the frequency of sign category occurrence [22]), with each sign category having low variance in its background. This can result in the object detector placing an over-dependence upon the object context when detecting objects [53], reducing the robustness of the models in varying environments.

Furthermore, there were more object categories present in the train, with a larger variance in class category representation [22]. We also observed that these infrequent signs were often in remote, isolated locations, whereas more frequent signs were clustered together. The number of sign categories and the number of instances per image are directly related to the amount of contextual information an image contains [30]. Therefore, there was more contextual information in images containing the more frequent signs. This will result in an undesired imbalance in the model's performance for different sign categories.

The distribution of labels in the real dataset is very uneven, with the least populated classification category being one-fifth the magnitude of the standard deviation. This is visualized by the long-tailed class representation distribution shown in **Figure 4.13**, indicating that the dataset has a foreground-foreground imbalance [34]. The impact of a foreground-foreground imbalance is discussed in **Section - 4.5.1**.

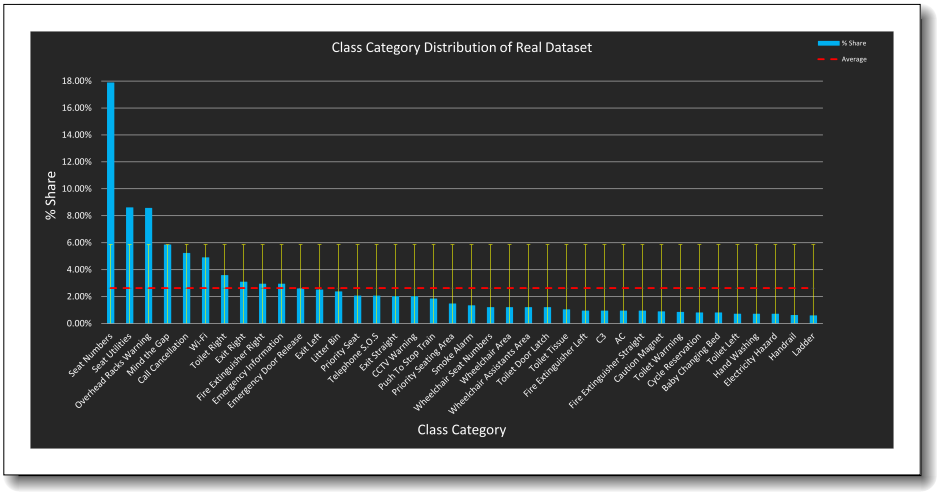


Fig. 4.13. A graph showing the class representation distribution in the real dataset, with the standard distribution represented by error bars deviating from the average. We see from the graph, that the data is long-tailed and unbalanced. The real dataset has a total of 777 images, with a total of 2,243 annotations, resulting in an average of just under 3 signs per image.

These problems are primarily the result of limited access to the environment, therefore, their impact should diminish if access to a wider variation of trainsets is available. Nevertheless, these problems will still arise and can be resolved by using methods described in **Section - 4.5.1**.

4.3.2 Dataset Annotation

Based on the size of the real dataset, and the time consumed annotating the mock dataset, the group decided to outsource its annotation. While professional providers such as Google were considered, the group ultimately chose to make use of a private individual via the website Fiverr [11]. The individual was provided with a list of annotation guidelines, and a set of pilot annotations were performed and reviewed prior to their full contribution. Upon completion, the annotations were reviewed by the group members and adjusted as required.

4.4 Damaged Dataset

Sign damage classification is accomplished by the use of an image classification model on signs in iconic object view. Before training the image classification model, an annotated dataset of damaged signs must be created. The dataset accruement method used in **Section - 3.3** and **Section - 4.3** could not be employed here, as the team did not have access to a train with damaged signs. As such, the dataset creation process had to be produced with an automated, image augmentation pipeline of the already acquired dataset.

4.4.1 Defining Damage Class Categories

Defining the class categories for damaged signs is a non-trivial task due to the coarse granularity of a damaged classification. It was observed that the most common types of damage that can occur to signs are either torn or scribbled upon signs. These types of damage can be classified by either a coarse classification ('damaged') or a finer set of classifications ('torn', 'scribbled' and 'torn and scribbled'). To investigate the effects of the different labelling hierarchies, two annotation files are produced for the damaged dataset: one with a fine set of classifications and one with a coarse set of classifications. The effects of the differing granular classifications on the image classification model are investigated in **Section - 4.5.2**.

4.4.2 Dataset Creation

An annotated dataset was created via an automated, image augmentation pipeline, shown in algorithm 4.1.

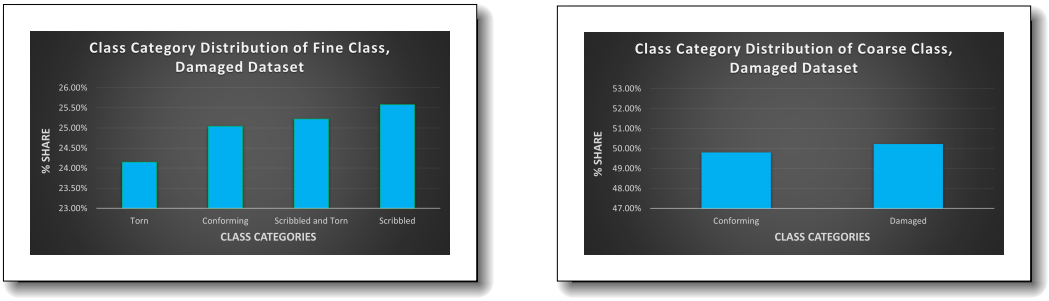
Algorithm 4.1 The algorithm used for the creation of the damaged dataset. The dataset object passed to the procedure contains each image and corresponding label as an enumerated set.

```
1: procedure CreateDataset(dataset)
2:     for (image, label) in dataset do
3:         apply_damage ← random(0, 3)          ▷ Define type of damage to apply
4:
5:         ApplyScribble(apply_damage)
6:         ApplyTear(apply_damage)
7:         TakeHomography(apply_damage)
8:
9:         annotations_file.append(img_file, damage_type)          ▷ Add annotation to
   annotations file
```

Using the bounding box annotation data, the damage is randomly applied to the signs using scribbled, or torn damage masks. The type of damage applied in the fine-grained classifications dataset is based on the random value of `apply_damage`:

- `apply_damage = 0` → apply no damage to sign in image
- `apply_damage = 1` → apply scribbled damage to sign in image
- `apply_damage = 2` → apply torn damage to sign in image
- `apply_damage = 3` → apply scribbled and torn damage to sign in image

This promotes an equal representation of each class in the dataset. The same method was applied to the coarse classifications dataset, however, the set of `apply_damage` values was restricted to $\{0, 1\} = \{$`'conforming', 'damaged'`$\}$. Proof of the balanced representation distribution for the two datasets can be seen in **Figure 4.14**.



(a) Class representation distribution for the fine-grained classification dataset. The dataset has an average % `share` = 25%.

(b) Class representation distribution for the coarse-grained classification dataset. The dataset has an average % `share` = 50%

Fig. 4.14. The class representation distributions for the two datasets. It can be observed that both datasets have well-balanced representations for each class, improving the robustness of any model trained on either of the datasets. Both damaged datasets have a total of 2,243 images and 2,243 corresponding classifications.

To simulate someone having drawn on the sign, 25 images of unique scribbles with a transparent background were produced to act as damage masks. For each image in the dataset, the damage mask was re-scaled to be between $\frac{1}{4} - \frac{3}{4}$ the size of the sign, then overlaid on top of the sign to have it appear drawn-on. Before overlaying the damage, the damage mask is augmented to improve the model's robustness for damage classification.

To simulate torn signs, a torn mask is overlaid on top of the corners of the sign. All tear masks are polygons with coordinates that have noise applied to them. This is done to ensure no tear mask is the same as another tear mask in the dataset, improving the ability of the model to generalize to different tears. The masks are filled with the pixel value taken from the edge of the bounding box to have it appear the same colour as the background. An example of an augmented image prior to the homography transform is shown in **Figure 4.15**.

Fig. 4.15. An example of an image with 4 signs of varying damage. The sign classifications are as follows: {'conforming', 'torn', 'scribble', 'scribble and torn'}={top-left sign, bottom-left sign, middle sign, right-most sign}.

The final step in the image-augmentation pipeline is to extract each sign in an image and display it from a canonical perspective. Formally, this problem is referred to as a perspective correction problem and is applied to each sign in the image. Details regarding the perspective correction of a sign are outlined below.

The plane camera model is a model that maps points from the world plane to points in an

image plane via a plane-to-plane homography [43]. This idea of mapping one image plane to the next can be used for any image plane, with the transform described by the matrix $\mathbf{H} \in \mathbb{R}^{3\times3}$. The transform is defined in **Equation 4**, transforming image plane $\mathbf{x}_1$ to the image plane $\mathbf{x}_2$ using $\mathbf{H}$.

$$\mathbf{x}_2 \sim \mathbf{H}\mathbf{x}_1 \tag{4}$$

The best technique used to solve for $\mathbf{H}$ is the homogeneous estimation method [6]. The homogeneous form of **Equation 4** is given in **Equation 5**. Here $\mathbf{h} \in \mathbb{R}^9$ is the flattened homography matrix $\mathbf{H}$ and $\mathbf{A} \in \mathbb{R}^{2n\times9}$ is acquired by a computation match obtained by rearranging **Equation 4** [6].

$$\mathbf{A}\mathbf{h} = 0 \tag{5}$$

By performing an SVD of $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}$, we get $\mathbf{h}$ from the column of the right-singular vector $\mathbf{v}_c$ that corresponds to the column, c, with the smallest singular value σ_c. $\mathbf{h}$ can then be reshaped into an $\mathbb{R}^{3\times3}$ matrix to acquire the homography transform $\mathbf{H}$ [7].

Using this technique, a perspective correction is performed for every sign in the image, completing the image-augmentation pipeline. Examples of images post-image-augmentation in the damaged dataset are shown in **Figure 4.16**.

Fig. 4.16. A grid of examples within the damaged sign dataset. In the image, the only signs that are included are damaged signs, however, within the actual dataset, there are also conforming signs with no damage applied to them.

4.4.3 Dataset Limitations

Although the image augmentation pipeline worked as expected, multiple unexpected limitations were encountered upon implementation of the trained models in the real world.

The initial version of the damaged dataset had a major flaw that did not allow for the correct classification of damage on real-world data. This is because the initial image-augmentation pipeline applied masks after a perspective correction. This resulted in the signs having a lower resolution than the damage masks (**Figure 4.17**), meaning the dataset did not replicate the scenarios that would be met during deployment. This resulted in low classification performance and required the image-augmentation pipeline to be altered to **Algorithm 4.1**.

(a) An example of the initial image-augmentation pipeline, where damage masks were applied after a perspective correction. It is observed that the damaged masks have a higher resolution than the sign.

(b) An example of the updated image-augmentation pipeline, where damage masks are applied before a perspective correction. It is observed that the damaged masks are of equal resolution to that of the sign.

Fig. 4.17. A comparison of the two image-augmentation pipelines. FIGURE 4.17B is a better representation of a scenario that would be met during deployment.

Upon improvement, an unexpected error regarding inconsistent perspective corrections was identified. This results in blurry, unintelligible normalized signs being used to train the model, as shown in **FIGURE 4.18**. Upon inspection, it can be observed that these inconsistencies are a result of background signs being normalised to a canonical perspective. This has a simple fix: only sampling from bounding boxes of a minimum area $A >$ `Th` would ensure that these unintelligible signs are not used in the dataset.

Fig. 4.18. An example of an unintelligible normalised sign in the damage dataset. Although examples of these are scarce throughout the dataset, cleaning up these unwanted images would improve the quality of the dataset.

A problem that there was no time to address was the inconsistent performance of the model on video inputs. This is due to the real dataset consisting of media acquired from photos in focus. This is inconsistent with the state of the media when it is acquired from a video feed, as the extracted key-frames are often out of focus and contain blur. This results in an additional blur after a perspective correction is performed. To combat this problem while using the same video logic used in **SECTION - 4.5.5**, the main improvement is the use of media acquired from videos out of focus in the dataset.

Despite this technique likely resulting in an improvement in the model's performance, the recommended method to improve this problem is the use of different video logic to that used in the project. This is further expanded upon in **SECTION - 4.5.5**.

4.5 Processing Server

4.5.1 Unbalanced Object Detection Datasets

A class distribution imbalance was identified in **Section - 4.3.1**. For the object detector task, this is defined as a foreground-foreground imbalance [34] and can often hinder object detector performance. This is because the detector will often overfit to the more represented data in the dataset and disregard underrepresented data. The effects of foreground-foreground imbalances can be mitigated by merging similar classes to increase the representation of the resulting class or by dataset sampling. There are two common sampling techniques:

- Undersampling - discarding overrepresented data to balance the overrepresented classes with other underrepresented classes
- Oversampling - augmenting images containing underrepresented data to balance their representation with the overrepresented data

The decision was made to merge similar classes to increase the representation of the resulting class, followed by the oversampling of underrepresented object categories.

Class Merging

Deciding which classes to merge for this task is a trivial task upon inspection. It is observed that 'x Left' and 'x Right' signs are visually similar, as seen in **Figure 4.21**, and can be represented by a single class category. The following class category merges of underrepresented class categories were defined, reducing the total number of class categories from 38 to 35.

- 'Toilet Left' and 'Toilet Right' $\Longrightarrow$ 'Toilet'
- 'Exit Left' and 'Exit Right' $\Longrightarrow$ 'Exit'
- 'Fire Extinguisher Left' and 'Fire Extinguisher Right' $\Longrightarrow$ 'Fire Extinguisher'

Fig. 4.19. 'Fire Extinguisher Left' sign example.

Fig. 4.20. 'Fire Extinguisher Right' sign example.

Fig. 4.21. An example of two sign categories that were merged into one, increasing the total representation of the category.

Data Oversampling

Subsequently, copies of images containing underrepresented class categories were augmented randomly using one of the following image augmentations techniques:

- Horizontal flipping
- Image scaling ($0.7 - 1.3\times$ the original image size)
- Image rotation (-30 to 30 degrees)

- HSV colour augmentation

Fig. 4.22. Examples of augmented oversampled images within the training and validation dataset.

The copies of images containing underrepresented categories were augmented until each underrepresented category distribution is similar to the initial representation of the most over-represented category. Examples of different image augmentations with bounding box predictions are shown in **Figure 4.22**.

Two disadvantages are identified from the oversampling of the image dataset. Although the total representation of all object categories is improved by oversampling, the representation of the most over-represented class category is also undesirably increased, as shown in **Figure 4.23**.

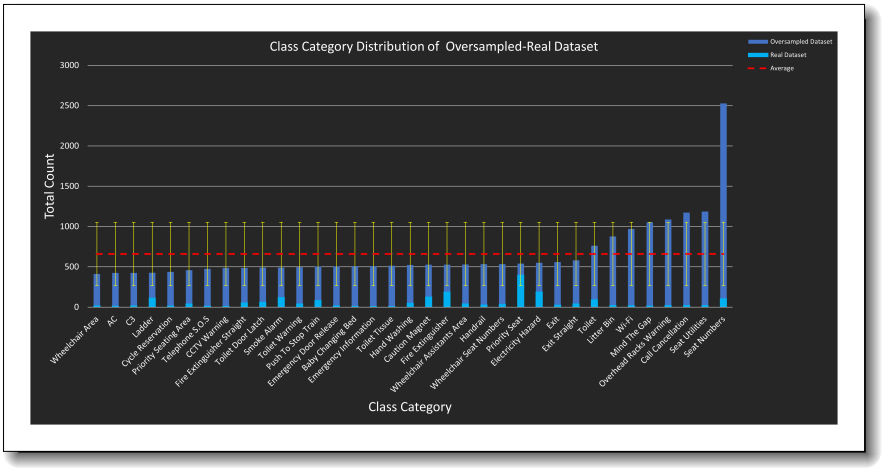


Fig. 4.23. A graph representing the distribution of each class category in the oversampled dataset. An improvement in class distribution is achieved compared to **Figure 4.13**. The oversampled dataset has a total of 6,715 images, with a total of 23,099 annotations.

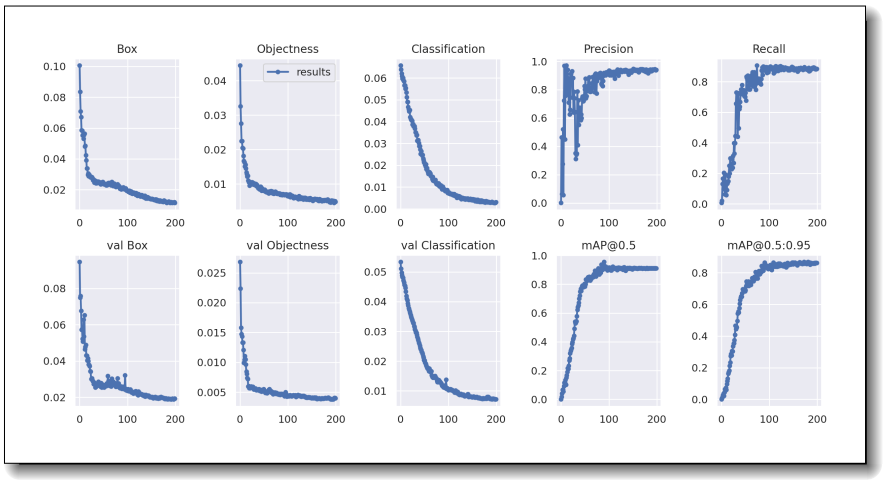
This is due to the frequent natural occurrence of the object category in the environment, resulting in the object categories' frequent presence within images that contain underrepresented objects. A second issue is the potential of the model to overfit to oversampled, underrepresented

data. This is especially prevalent for object categories that have very infrequent absolute instances (e.g. 'Ladder' with 11 total instances). The distribution of labels in the oversampled dataset is much more even than the unsampled dataset, with the magnitude of the least populated classification category being greater than the magnitude of the standard deviation.

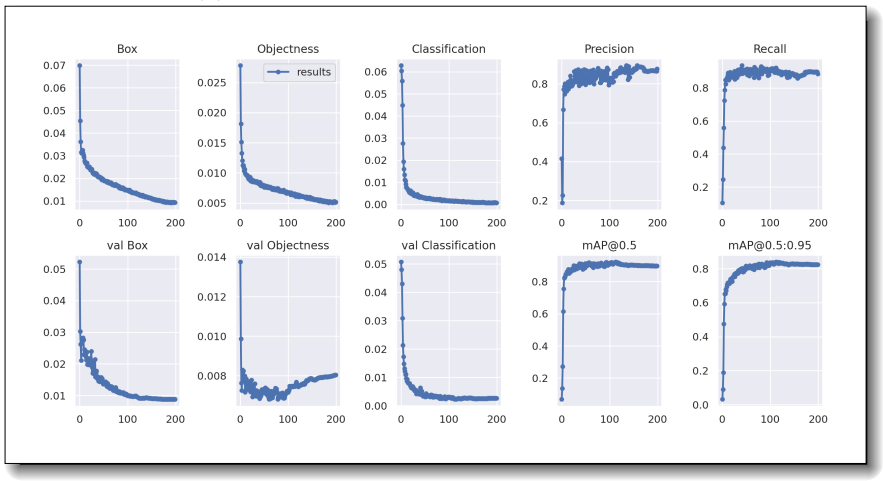
A more robust solution is the use of a generative method like ASDN (adversarial spatial dropout network) [50] to generate artificial samples. This would, once again, improve the distribution bias, but with greater control over potential side-effects like overfitting and increases in over-representation of already over-represented classes.

Model Training

To evaluate the improvements of the augmented dataset compared to the standard dataset, two YOLOv7 [48] models were fine-tuned, in a similar manner to **Section - 3.4.3**, and compared. To speed up the training process and ensure loss convergence is reached, the training was performed over 200 epochs on 2 GPUs using the University of Southampton's Alpha cluster. The predefined hyperparameter values for the model were empirically tuned in a similar manner to **Section - 3.4.3**, with the final assortment of parameter values listed in **C.1.2 - Object Detection Model**.



(a) The learning curves for the unsampled dataset.



(b) The learning curves for the oversampled dataset.

Fig. 4.24. Plots that illustrate the improvement in learning when a more even representation of class categories is used for training data.

The learning curve comparisons, supplied in **Figure 4.24**, illustrate the improvement in the classification loss, precision, recall and mAP convergence for the model trained on the oversampled dataset.

Subsequently, the f1-plots produced from training were used in a similar manner to **Section - 3.4.3**, to identify the optimal confidence threshold, $\overline{\tau}_{\texttt{conf}}$, for the two models:

- $\overline{\tau}_{\texttt{conf}}^{(original)} = 0.561$
- $\overline{\tau}_{\texttt{conf}}^{(oversampled)} = 0.602$

The models were then tested using their respective optimal confidence threshold values on the test set. The final precision-recall plots in **Figure 4.27** provide the best comparison of the two models' performance. Although negligible improvement in model recall was identified, the model's precision was greatly improved. This is reinforced by $mAP_{oversampled} > mAP_{original}$ providing evidence that the oversampled model is superior.

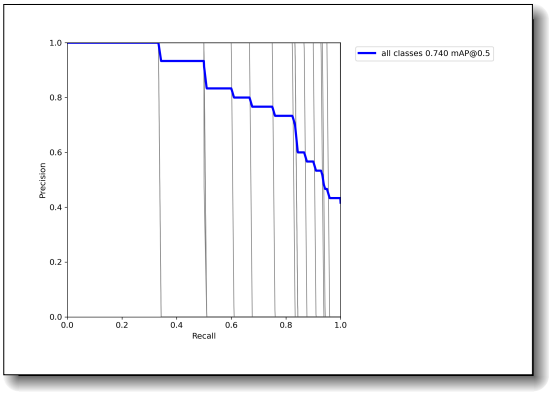


Fig. 4.25. PR plot of the model trained on the unsampled dataset.

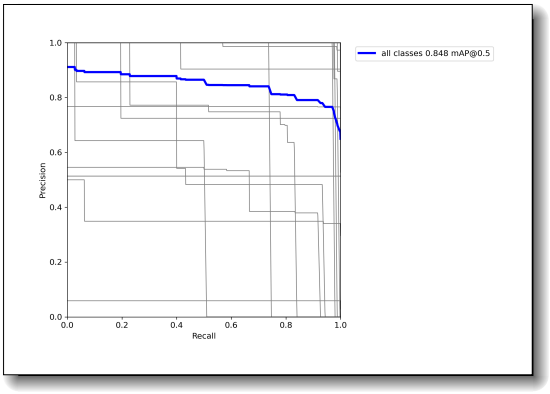


Fig. 4.26. PR plot of the model trained on the oversampled dataset.

Fig. 4.27. A comparison of the two models' PR plots. An improved balance between precision and recall is identified when a more even representation of the classes is present in the dataset.

When comparing the two confusion matrices in **Figure 4.30**, a clear reduction in confusion between underrepresented object categories can be observed for the oversampled model.

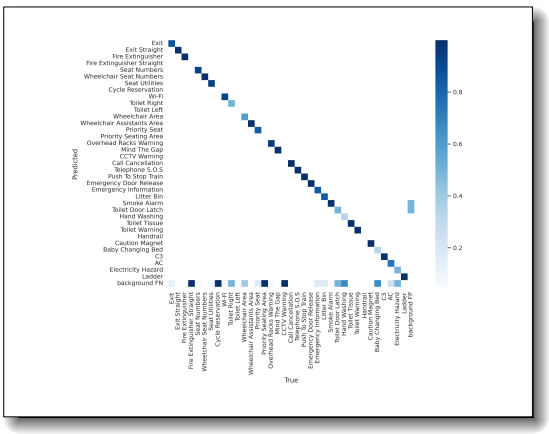


Fig. 4.28. Confusion matrix of the model trained on the unsampled dataset.

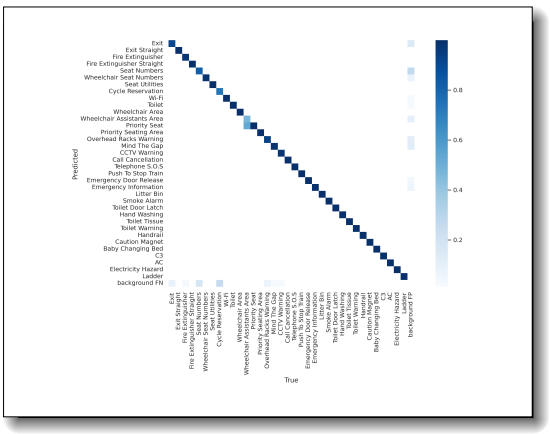


Fig. 4.29. Confusion matrix of the model trained on the unsampled dataset.

Fig. 4.30. A comparison of the two models' confusion matrices. There is a clear improvement in category classification when using the model trained on the oversampled dataset.

4.5.2 Image Classification

Image classification, for single-label classification tasks, is the task of assigning a predefined label to a given input image [37]. The performance of an image classification model is often categorized with an accuracy metric, however, the precision, recall and f1-score are often also used to provide more information regarding the classification performance of models for individual label classifications. Common benchmarks for image classification models that are used in research are the ImageNet [8] and CIFAR10/100 [24] datasets. The two types of image classification models that currently dominate the field are convolutional neural networks (CNN) and vision transformer (ViT) models.

Convolutional Neural Networks (CNNs)

CNNs have long been the standard technique used for image classification, being first introduced in [27] and later becoming the standard for deep learning computer vision models. CNNs utilize the convolution operation with a trainable kernel to identify salient features within an image, producing feature maps that lose spatial information from the input but learn rich feature representations, with the layer depth being proportional to the complexity of identified features [23, 1]. Additionally, CNNs utilize pooling layers to downsample feature maps while maintaining as much of the feature representation as possible [27, 23]. Combining convolution and pooling layers promotes a high inductive bias for locality and translational equivariance, improving the CNNs performance on unseen data [23].

CNNs are commonly used in tandem with techniques that aim to improve model attributes during learning or inference. Common techniques include:

- Image preprocessing (image augmentation) [36]
- Dropout [17]
- Residual connections [16]
- Batch normalization [20]

Image augmentation is a common preprocessing technique that is applied to the training dataset prior to model training. Image augmentation techniques apply a multitude of image transformations that increase the diversity of the image dataset, thus mitigating the likelihood of overfitting and improving model robustness after training [36].

Dropout is a regularization technique used in neural networks (NN) that removes perceptrons within a layer based on a predefined probability. This reduces any dependencies on individual perceptrons and reduces the likelihood of overfitting.

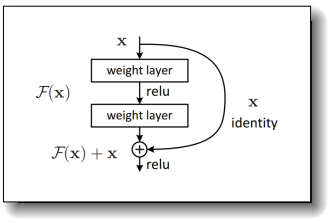


Fig. 4.31. An example of a residual connection that skips two convolution layers.
Source: [16]

Residual connections [16] are used to reduce the effects of the vanishing gradient problem. When training deep models using a gradient-based approach, like backpropagation [40], the gradient signal often diminishes during its propagation back to the beginning of the model [3]. Residual

connections, shown in **Figure 4.31**, mitigate this effect by providing pathways that facilitate the propagation of the gradient signal through the network [16].

Batch normalization mitigates the problem of internal covariate shift [20]. Every layer in a deep model has an input with a different data distribution. The data distribution is affected by changes in model parameter values and changes in each layer's input data at every training iteration. The effect on the distribution of the input to each layer by these changes during training is termed internal covariate shift by the authors of [20]. Batch normalization standardizes the inputs to hidden layers of a deep network and is shown to improve the convergence rate of the model during training [20].

Vision Transformers (ViTs)

Vision transformer models (ViT) [9] are based upon transformer models that utilize the self-attention mechanism to derive internal representations of sequences [47]. Multi-headed attention is used to produce multiple attention scores for each element in a sequence, allowing for subtle representations between the elements in a sequence to be identified [47]. The desirable trait of transformers is that large-term dependencies can be taken into account, an improvement upon similar models that learn sequence representations like RNNs [41], LSTMs [18] and GRUs [5]. These similar models have an inherent characteristic that the relative dependency distances are limited to a maximum distance, limiting the model's ability to form larger dependencies.

Vision transformers utilize the same multi-headed attention mechanism, however, their inputs are images. As such, the images must be tokenized, prior to being passed to the ViT. This is achieved by splitting an image into $p \times p$ image patches and subsequently using the image patches to acquire a sequence of linear embeddings that are passed as input to the ViT encoder stack [9].

Unlike CNNs, ViTs have low inductive bias and do not possess beneficial computer vision (CV) capabilities like locality and translational equivariance. Therefore, ViTs require a large amount of pre-training prior to being trained on downstream tasks. Despite this, ViTs often provide better performance than CNNs, especially when they are subject to more training data.

4.5.3 Model Selection

A wide variety of models were researched to identify the optimal model for the image classification task. The optimal model is selected based on the same criteria as **Section - 3.4.2**: high classification performance, high temporal efficiency, and should be easily deployable for inference. Both CNN and ViT models were researched due to their current dominance in the SOTA for the image classification task. Below is a breakdown of the researched image classification architectures followed by a model comparison in table **Table 2**, used to explain why the chosen model was selected.

CNN Models

AlexNet [25] was a groundbreaking paper that achieved SOTA results on the ImageNet [8] challenge. AlexNet introduced the ReLU non-linearity, an activation function that was shown to improve the convergence rate of CNNs during training by mitigating the vanishing gradient problem [25]. Furthermore, the model was the largest CNN model of the time, with over 62 million

trainable parameters, allowing more complex features to be identified. AlexNet was trained with image augmentation techniques to increase the data distribution diversity during training [36], and with dropout regularization [17] to reduce model overfitting.

VGGNet investigated CNN models of larger depth but with fixed convolution filters ($f = 3$) and pooling layers ($f = 2$), resulting in a larger model with over 138M parameters. VGGNet uses a fixed-sized convolution filter to reduce the number of trainable parameters in the model for a fixed depth while replicating the receptive field of variable-size convolution filters [44]. This increases the efficiency of larger models and allows the classification performance to be improved.

As NN models increase in size, the vanishing gradient problem becomes more of a prevalent issue. Like VGGNet [44], ResNet consists of fixed convolution layers of size $f = 3$, resulting in a model with over 11M parameters, however, residual connections are placed between every two CONV layers, as shown in **Figure 4.31**. ResNet learns residuals to mitigate the vanishing gradient problem [3, 16], allowing deeper models to be trained more efficiently.

In image data, frequently, there is a large variance in the size of the salient features within any image. The scale of the kernel is proportional to the scale of the identified salient features that the model will learn. Prior architectures like VGGNet and ResNet used fixed-size kernels, meaning the scale of the identified salient features is fixed. As an improvement, InceptionNet trains the model on a range of kernel sizes, increasing the diversity of identified salient features, and improving model generalization [45].

The MobileNet image classification series introduce techniques that can be used to improve the computational efficiency of deep learning models such that they can be deployed onto mobile/embedded devices. MobileNetV1 introduces a more computationally efficient convolution operation, termed the depth-wise separable convolution. This consists of a depthwise convolution with a kernel that has only one channel and a pointwise convolution with a kernel that has a filter size of 1 [19]. This provides a computational saving for a trade-off in classification performance. MobileNetV2 further improves this pipeline with its bottleneck block that appends an expansion operator before the depth-wise separable convolution to increase the amount of information to be learned from smaller blocks [42]. This allows for a further improvement in inference speed, with negligible impact on the classification performance.

ViT Models

ViTs broke the trend of CNN models dominating the computer vision field by deploying transformer networks for the image classification task [9]. However, two major pitfalls regarding ViTs were their quadratic computational complexity and fixed scale tokens, resulting in an inferior performance for image tasks with high resolutions and varying object scales. Swin transformers implement hierarchical feature maps and patch merging to address the inferior performance of image tasks on varying object scales [32]. Additionally, the use of window and shifted window multi-headed self-attention reduces the quadratic computational complexity of the original ViT [32], improving the classification performance of the model.

The final investigated model was the BERT pre-training of image transformers (BEiT) model. BEiT presents a novel pipeline for CV tasks that was the first to achieve superior performance with self-supervised pre-training of ViTs compared to the conventional supervised pre-training process. Self-supervised learning (SSL) consists of learning one section of the input from another part of the input [15]. The knowledge obtained by the model during the pre-training task is subsequently transferred from the pretext model to a downstream task. The pre-training objective of the BEiT model is to predict visual tokens of corrupted image patches [2]. After pre-training, task layers are appended to the BEiT architecture and then fine-tuned on the image classification downstream task.

The Selected Model

Model	Benchmark	# of Parameters=	Top-1 Error
AlexNet	ImageNet	62M	37.5%
VGGNet	ImageNet	138M	23.7%
ResNet-101	ImageNet	95.7M	19.87%
InceptionNet	ImageNet	11.1M	21.7%
MobileNetV1	ImageNet	4.2M	18.3%
MobileNetV2	ImageNet	3.4M	18.3%
Swin-L	ImageNet	197M	12.7%
BEiT-L	ImageNet	331M	11.4%

Table 2: A breakdown of the performances of the researched models on the ImageNet image classification benchmark [8].
Sources: [25, 44, 16, 45, 19, 42, 32, 2]

After extensive research of image classification models was completed, the decision was made to opt for the BEiT image classifier [2] due to its high classification performance on the ImageNet [8] benchmark as well as its ability to transfer knowledge to a downstream task. The damage classification task is a coarsely defined problem that must avoid overfitting to a specific type of damage. As such, the decision was taken to opt for a model that had been pre-trained on a pretext image task, hoping the knowledge transfer would help the model in generalizing well to different types of damage.

4.5.4 BEiT Training and Evaluation

In order to fine-tune the selected pre-trained model and evaluate its performance, the HuggingFace [52] implementation of BEiT [2] was used. Similarly to **Section - 3.4.3**, the University of Southampton's Alpha Cluster was utilised to provide enhanced computational resources.

As damage is such a coarsely defined label, two BEiT models were trained to investigate the effect of different classification granularities. One model classified coarse labels and the other model classified fine labels. Each model was trained for 50 epochs on the damaged dataset, detailed in **Section - 4.4**, consisting of 3012 images, according to an $85:15$ training:testing split, with 10% of the training dataset being used for the validation dataset. For the coarse classification model, annotations of {`'conforming', 'damaged'`} were used, while the fine classification model made use of finer damage labels: {`'conforming', 'scribble', 'torn', 'scribbled and torn'`}. The predefined hyperparameter values for the model were tuned such that they provide optimal training curves and loss convergence, shown in **Figure 4.32**, with the final assortment of parameter values listed in **C.1.3 - Damage Classification Models**.

The learning curves given in **Figure 4.32** illustrate that the coarse class classification model outperforms the fine class model on the validation set, with a lower validation loss at convergence ($\texttt{L}_{coarse} : 0.0731 < \texttt{L}_{fine} : 0.1617$), and higher validation accuracy ($\texttt{accuracy}_{coarse} : 98.33\% > \texttt{accuracy}_{fine} : 96.11\%$) . Both models, however, give extremely high inference speed, with approximately 5 images processed per second ($\approx 20ms$ inference time).

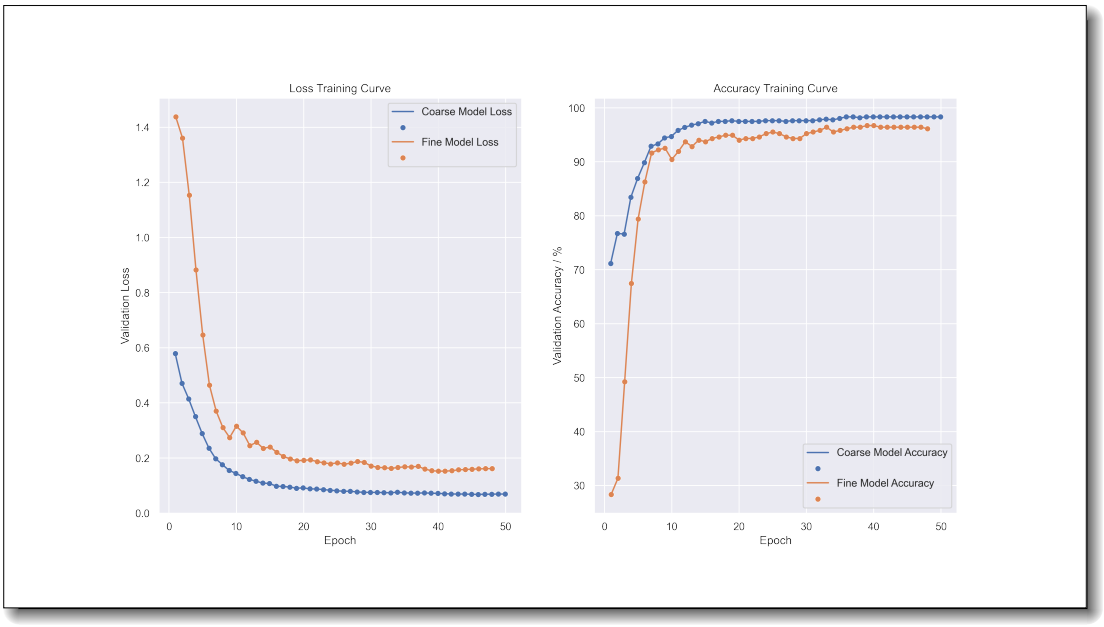


Fig. 4.32. Learning curves for both fine and coarse BEiT image classification models on the damaged validation dataset. The coarse model has a lower validation loss at convergence and a higher validation accuracy.

Analysis of the models' class-specific performance is given by their confusion matrices in **Figure 4.35** produced using the test set. From these matrices, it can be observed that both models have high-class classification performance, with little confusion between categories. Additionally, while the perfect performance of the course model is unexpected, further testing on additional data has validated this result. It is concluded, therefore, from the high performance on the validation and test sets, that the models have trained well, however, the coarse classifier may have overfit to the training data, potentially reducing its performance on real-world data.

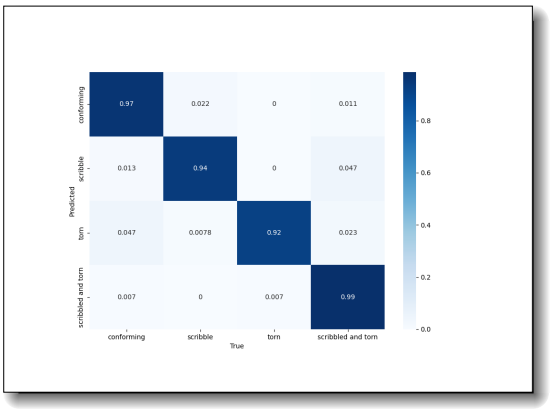


Fig. 4.33. The confusion matrix of the fine class image classification model.

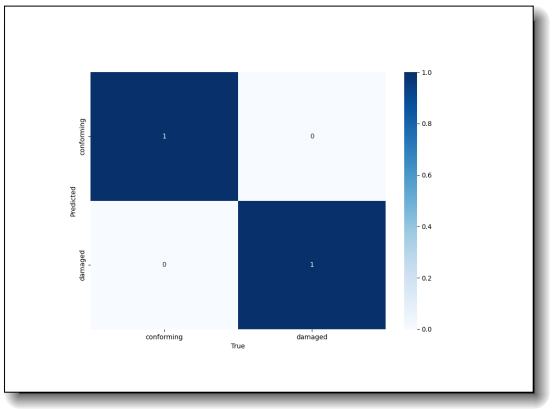


Fig. 4.34. The confusion matrix of the coarse class image classification model.

Fig. 4.35. A comparison of the confusion matrices for the two image classification models. It is clear that the coarse classification model outperforms the fine classification model on the test dataset as well as the validation dataset (shown in **Figure 4.32**).

Table **Table 3** provides additional analysis of the models' performance according to standard evaluation metrics. Notably, the recall of 'torn', and the precision of 'scribbled and torn' classifications underperform compared to the other classification categories. The torn images serve

as a common denominator in these concerns, prompting an investigation into their legitimacy. Upon inspection, it can be seen that although most digitally created tears are clearly visible to the human eye, some remain challenging to identify, indicating the presence of misrepresentative data within the dataset.

Table 3: BEiT image classification performance metrics on the damaged testset.

	precision	recall	f1-score	support
conforming	0.95	0.97	0.96	178
scribble	0.97	0.94	0.95	150
torn	0.99	0.92	0.96	129
scribbled and torn	0.92	0.99	0.95	142
accuracy			0.95	559
macro average	0.96	0.95	0.95	559
weighted average	0.96	0.95	0.95	559

(a) Fine Classificatier Performance

	precision	recall	f1-score	support
conforming	1.00	1.00	1.00	325
damaged	1.00	1.00	1.00	611
accuracy			1.00	936
macro average	1.00	1.00	1.00	936
weighted average	1.00	1.00	1.00	936

(b) Coarse Classificatier Performance

4.5.5 Transitioning to Video

Typical train carriages contain extended sections, such as the seated areas shown in **Figure 4.36**, that do not lend themselves to the image-based approach used in deliverable 1. As such, the group decided to explore the use of video capture within an inspection.

Initially, to avoid the need for additional processing algorithms, the concept of a video-to-panorama system was explored. In this approach, the user would take a single video as they walk through the section, which would then be stitched together into a single panoramic image, and passed onto the pre-existing processing algorithms. The trialed system worked by breaking down the video into its frames, and stitching consecutive frames together using a feature-matching algorithm. While results were promising for a small number of frames, the quality of the panorama deteriorated as the number of frames increased (**Figure 4.37**). As such, this approach was discounted.

Fig. 4.36. Example section of a train that is unsuitable for an image-based inspection

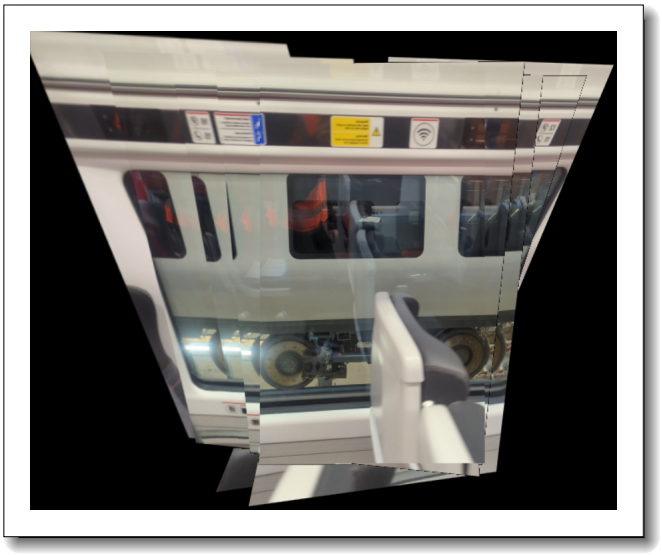


Fig. 4.37. Example results from the video-to-panorama section

Due to the difficulties presented by the video-to-panorama approach, a problem-specific multiple object tracking (MOT) was designed and implemented. The MOT algorithm follows the detection-based tracking (DBT) [33] paradigm, utilizing the predictions from the object detector for all frames within a video and then identifying unique objects using the algorithm in **Section - C.2**.

The DBT algorithm is broken into 3 main stages:

1. Detect signs with the object detector for every frame
2. Filter signs on a frame-by-frame basis
3. Identify unique sign objects within the video

Once the object detector has detected all of the signs in the video, the signs are filtered based on the following criteria.

Any sign that is within the padded area of the frame is removed. A theoretical, padded area is defined by the algorithm that decreases the frame width and height. The padding size, p, is a hyperparameter that defines the padded area, A_{padded}, for an image of $A = H \times W$ by equation 6.

$$A_{padded} = (H - 2p) \times (W - 2p) \tag{6}$$

This constraint is placed due to the object detectors diminishing performance for occluded/partially occluded signs. DBT algorithms are highly dependent on the object detector performance [33], therefore, to ensure the correct signs are identified, any bounding box prediction that has coordinates outside of the padded area is discarded.

Additionally, any non-repeated signs are discarded. A repeated sign is defined as a sign that is predicted in at least two contiguous frames. This aims to discard any signs that are misidentified by the object detector, reducing the DBT algorithms dependency upon it.

After the signs are filtered, the unique sign identification algorithm is deployed. This algorithm receives an $f \times n$ list (where f is the number of frames and n is the number of signs in the frame) of filtered signs and returns a single list of unique sign objects within the video. The algorithm can be found in appendix C.2 and is summarised by the following steps.

1. Get the camera movement direction in the video
2. Initialize a list of `prior_frame` signs with all signs in frame 0
3. Loop through every frame in video and every sign in frame
   (a) If a sign that was not in the prior frame is detected, then the sign is a unique unidentified sign and should be appended to the list
   (b) If a sign that was in the prior frame is detected
      i. If a signs change in position opposes the direction of the camera movement, then the sign has already been identified
      ii. If a signs change in position follows the direction of the camera movement, then the sign is a unique unidentified sign and should be appended to the list
   (c) Replace `prior_frame` signs list with all signs in the current frame
4. Return list

Acquiring the camera movement direction is a trivial task when the bounding box information is present. For a repeating sign, the initial, $x^{(i)}_{coord}$, and subsequent, $x^{(i+1)}_{coord}$, bounding box x-coordinates are acquired and the following logic identifies the camera movement direction.

- If $x^{(i)}_{coord} < x^{(i+1)}_{coord} \implies$ camera movement direction is to the left
- If $x^{(i)}_{coord} > x^{(i+1)}_{coord} \implies$ camera movement direction is to the right

There are, however, multiple disadvantages regarding the algorithm. The most apparent limitation is the assumption regarding the constant camera movement in a horizontal direction. This forms a heavy bias towards the application. Furthermore, the algorithm places a high dependency on the object detector, potentially resulting in problems with occluded signs in the video. Finally, the algorithm is not efficient with a time complexity of $O(n^2)$ leading to quadratic computation times for linear increases in the number of frames within a video.

A more robust approach would be the use of DeepSORT such that object appearance is considered in conjunction with the result of the object detector [51]. This improves the MOT performance while also being independent of camera direction and pose, increasing the robustness of the solution.

4.5.6 Updated Server Workflow

To satisfy the requirements of the deliverable 2 system, the processing server must be updated to support the processing of videos, and detection of damaged signs.

To achieve this workflow, each checkpoint is now processed based on their media type, and extra steps are added for damage detection, as shown in algorithm **Algorithm C.2**. Note that only the "ProcessCheckpoint" procedure is listed, as the remainder of the workflow is unchanged. To detect the presence of signs, no changes are made to image-based checkpoints, while the video handling algorithm discussed in **4.5.5** is utilised for video-based checkpoints. To detect damaged signs, a normalised version of each detected sign is gathered using the techniques discussed in **4.4.2**, and passed into the damage detector detailed in **4.5.4**. Regardless of the classification granulairty, the data model supports only "conforming" or "damaged" results. In the case of video-based checkpoints, the normalised sign images are extracted from the first frame in which the video-handling algorithm detects the sign. Finally, the conformance status of the inspection objects, and the vehicle are updated to reflect the results of the processing, and posted to the cloud server.

4.6 Testing

4.6.1 Scenario Testing

To assess the performance of the updated application, its functionality was tested against each of the scenarios defined in **Section - 4.2.2.11**. The results of this testing are summarised in **Table 4.1**, with any differences in the actual outcome of the application noted, and its cause identified. Additional analysis of the application was carried out in the form of storyboard testing, where the mock-up designs, and produced system are compared and differences justified. This storyboard testing is presented in **D.2 - Deliverable 2: Storyboard Testing**.

Table 4.1: Deliverable 2 Scenario Testing

Scenario 4		
Scenario	**Actual Outcome**	**Notes**
Scenario 4 - Updated Inspection Process	The app performed as described in the scenario	No notes
Scenario 5		
Scenario	**Actual Outcome**	**Notes**
Scenario 5 - View Status	The app performed as described in the scenario	No notes
Scenario 5 Additional Flow 1 - Unremediated Non-Conformance	The app performed as described in the scenario	No notes
Scenario 5 Additional FLow 2 - Remediated Non-Conformance	The app performed as described in the scenario	No notes
Scenario 6		
Scenario	**Actual Outcome**	**Notes**
Scenario 6 - Sign Purchase	The app performed as described in the scenario	No notes
Scenario 6 Alternative Flow 1 - Remediation Cancelled	The app performed as described in the scenario	No notes
Scenario 6 Alternative Flow 2 - Specific Sign Purchase	The app performed as described in the scenario	No notes
Scenario 6 Alternative Flow 3 - Removing All Signs	The app performed as described in the scenario	No notes
Scenario 6 Alternative Flow 4 - Change Order	The app performed as described in the scenario	No notes
Scenario 6 Alternative Flow 5 - Cancel Order	The app performed as described in the scenario	No notes
Scenario 6 Alternative Flow 6 - Log Remediation	The app performed as described in the scenario	No notes

Scenario 7		
Scenario	**Actual Outcome**	**Notes**
Scenario 7 - Update Conformance Status	The app performed slightly differently to the scenario	Result of change in UI design to only have a "finish" button for convenience.
Scenario 7 Alternative Flow 1 - Conformance Status Unchanged	The app performed slightly differently to the scenario	User now only needs to select "finish" without making any changes to the conformance status.
Scenario 7 Alternative Flow 2 - Rectify Incorrect Update	The app performed slightly differently to the scenario	User now only needs to change the value back before clicking "finish"
Scenario 8		
Scenario	**Actual Outcome**	**Notes**
Scenario 8 - View Logged Remediation	The app performed as described in the scenario	No notes

4.6.2 Integration & Field Testing

To assess the integration of **AutoSign**'s extended components, and the performance of its new algorithms, field testing was carried out. While, this testing was due to take place on the same train that was used for the collection of the real dataset, a substitute train was unfortunately used in its place, which contained very few signs, and was still in construction in part. Although the train did not provide a suitable testing environment, the system was minimally trialled and in all cases, was able to detect the missing signs across the train.

Due to the nature of the project, the group were unable to plan an additional visit to the train depot for further testing, and instead decided to establish a new mock environment, using printed signs from the real dataset. While this would provide a more suitable testing space, and give greater control over simulating damage, it was hypothesised that the system would underperform due to the environmental differences between the space, and the environment used to train the underlying models.

Similarly to deliverable 1, a number of scenarios were defined, where an inspection of the environment was carried out with varying amounts of non-conforming signs present. In this testing, the system was again able to function correctly and cohesively as a single unit. While the processing server was able to accurately classify damaged signs, the object detector under-performed in comparison to its normal rates of accuracy, as hypothesized. Additionally, the system struggled when presented with more extensive and diverse damage, which can likely be attributed to the small set of damage masks used to train the model. **Figure 4.38** showcases the results of an example scenario, where-by two signs within the checkpoint were missing, and correctly

identified by the system.

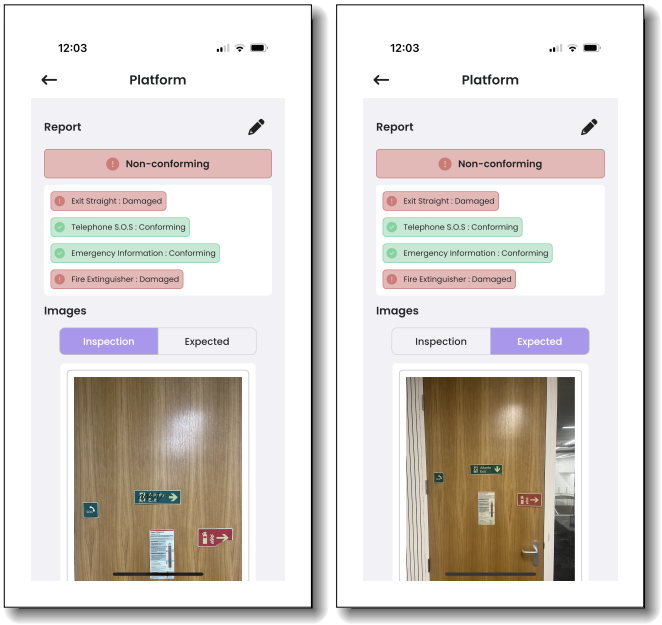


Fig. 4.38. Screenshots showing the detection of damaged signs within a checkpoint

While the results of this testing were not as conclusive as in deliverable 1, the group is not concerned by the system's performance, attributing the difficulties faced by the object and damage detectors to the difference in environment, and basic training set used respectively. In future deployments, these issues can be resolved using more expansive and representative datasets.

SECTION 5

Project Management

5.1 Time Management

5.1.1 Time Plan

There were four components to the project's time plan: formulation, two development iterations, and documentation. During the formulation, the group met with the client and supervisor. A detailed specification for the project was constructed, including its goals, constraints, and design. Next, and as discussed in **2.4 - Development Plan**, the development of the system was split into two deliverables, with these deliverables structured around the project deadlines and university term dates. Following the development, the remaining time was dedicated to minor bug fixes, additional testing and documentation, including the group report, final presentation and project poster. Further testing and project evaluation were also carried out during this time. Additionally, at a weekly level, the group adopted an agile mindset to work, dynamically updating time plans based on progress and encountered problems. This time plan is summarised in **Figure 5.1**, with the project Gantt chart provided in **E.1 - Project Gantt Chart**.

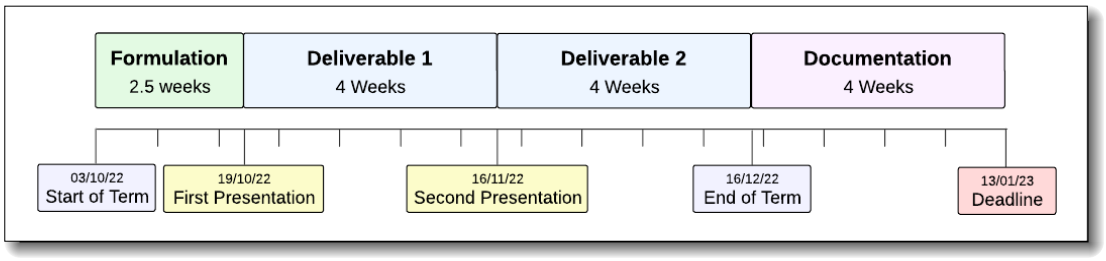


Fig. 5.1. An overview of the project's time plan

5.1.2 Techniques

The group made use of many time management techniques throughout the project's duration, including:

- **In-Depth Time Planning**: The group maintained a detailed and adaptable time plan within a Gantt chart
- **Skill-Based Workload Estimation**: When planning, skill-based estimation was used to assess the completion time of tasks, with additional time allocated to account for under-estimation.
- **Contingency Planning**: Contingency plans and time were put in place for critical tasks, as to avoid the impact of issues.
- **Regular Progress Meetings**: The group met regularly to discuss progress comparatively with the agreed time plan and make adjustments where required.

5.2 Workload Management

5.2.1 Division of Responsibilities

During the project's formulation, the group elected a leader and divided the responsibility of the system's development among group members. For the major components - the application and processing server - two sub-teams were formed based on skill sets, past experiences and estimated workload. For the datasets and testing, all group members were declared equally responsible, while a single individual was given the task of implementing the cloud server, based on the minimal work required to do so. Table 5.1 summarises this information.

Table 5.1: Development: Group Responsibilities

Item	Group Members
Project Lead	Charles Powell
Application	Charles Powell, Isaac Dunford
Processing Server	Benjamin Sanati, Killian Clarke, Gerasim Tsonev
Cloud Server	Charles Powell
Datasets & Testing	Whole Team

5.2.2 Development Contributions

Given the division of responsibilities described above, Table 5.2 provides a breakdown of the contributions of each of the group members to the development of the system's main components, with **D** and **W** referring to relative difficulty and workload respectively (1 low, 5 high).

Table 5.2: Development: Group Contributions

Application			
Item	**Group Members**	**D**	**W**
Structure & Interface Design	Charles Powell, Isaac Dunford	4	4
Home Page	Charles Powell	2	2
Profile Page	Charles Powell	2	3
Status Page	Charles Powell	2	3
Inspect Page	Charles Powell	4	5
Remediate Page	Charles Powell	3	4
Inspections Page	Isaac Dunford	3	3
Remediations Page	Isaac Dunford	3	3
Processing Server			
Item	**Group Members**	**D**	**W**
Research	Benjamin Sanati	5	5

Object Detection	Benjamin Sanati	3	3
Damage Detection	Benjamin Sanati	4	3
Video Processing	Benjamin Sanati	5	2
Server Workflow	Benjamin Sanati, Charles Powell	2	4
Video-to-Panorama	Killian Clarke, Gerasim Tsonev	3	3
Seat Number Recognition	Killian Clarke	2	1
Cloud Server			
Item	**Group Members**	**D**	**W**
Data Model Definition	Charles Powell	3	3
Setup	Charles Powell	1	1
Integration	Charles Powell	1	1
Datasets			
Item	**Group Members**	**D**	**W**
Mock Dataset Accruement & Annotation	Whole Team	1	4
Real-World Dataset Accruement	Whole Team	1	2
Real-World Dataset Annotation	Outsourced	N/A	N/A
Oversampled Real-World Dataset	Benjamin Sanati	4	2
Damaged Dataset	Benjamin Sanati	4	5
Testing			
Item	**Group Members**	**D**	**W**
Application Testing	Isaac Dunford	2	3
Processing Server Testing	Benjamin Sanati	2	3
Integration & Field Testing	Charles Powell, Benjamin Sanati, Killian Clarke	2	4

5.2.3 Documentation Contributions

In general, each member was responsible for the documentation of their development, with non-development sections divided equally amongst members based on knowledge, preference and estimated workload. This strategy was applied to the project's group report and presentations. **Table 5.3** provides a breakdown of the contributions of each of the group members to the project's documentation, with **D** and **W** referring to relative difficulty and workload respectively (1 low, 5 high).

Table 5.3: Documentation: Group Contributions

Group Report			
Item	**Group Members**	**D**	**W**
Introduction	Charles Powell	2	2
System Design	Charles Powell, Isaac Dunford	2	3
Deliverable 1: Application	Isaac Dunford	3	4
Deliverable 1: Mock Dataset	Benjamin Sanati	3	3
Deliverable 1: Processing Server	Benjamin Sanati	5	5
Deliverable 1: Cloud Server	Charles Powell	2	2
Deliverable 1: Testing	Isaac Dunford, Killian Clarke, Gerasim Tsonev	2	2
Deliverable 2: Application	Isaac Dunford	3	4
Deliverable 2: Real World Dataset	Benjamin Sanati	4	3
Deliverable 2: Damaged Dataset	Benjamin Sanati	3	3
Deliverable 2: Processing Server	Benjamin Sanati	5	5
Deliverable 2: Cloud Server	Charles Powell	2	2
Deliverable 2: Testing	Isaac Dunford, Killian Clarke, Gerasim Tsonev	2	2
Project Feedback	Charles Powell	2	3
Project Management	Charles Powell	2	2
Evaluation	Charles Powell, Isaac Dunford	3	3
Client Recommendations	Killian Clarke	3	3
Conclusion	Charles Powell	3	2
Presentations			
Item	**Group Members**	**D**	**W**
First Presentation	Whole Team	2	2
Second Presentation	Whole Team	3	3
Final Presentation	Whole Team	3	2
Group Poster	Charles Powell	2	3

5.3 Team Dynamics

5.3.1 Communication

Discord was chosen as the group's primary communication channel based on its support for the exchange of a wide variety of media, and structured communication flows. A private server was established, with multiple different channels created to organise communication based on the subject matter, allowing for efficient retrieval.

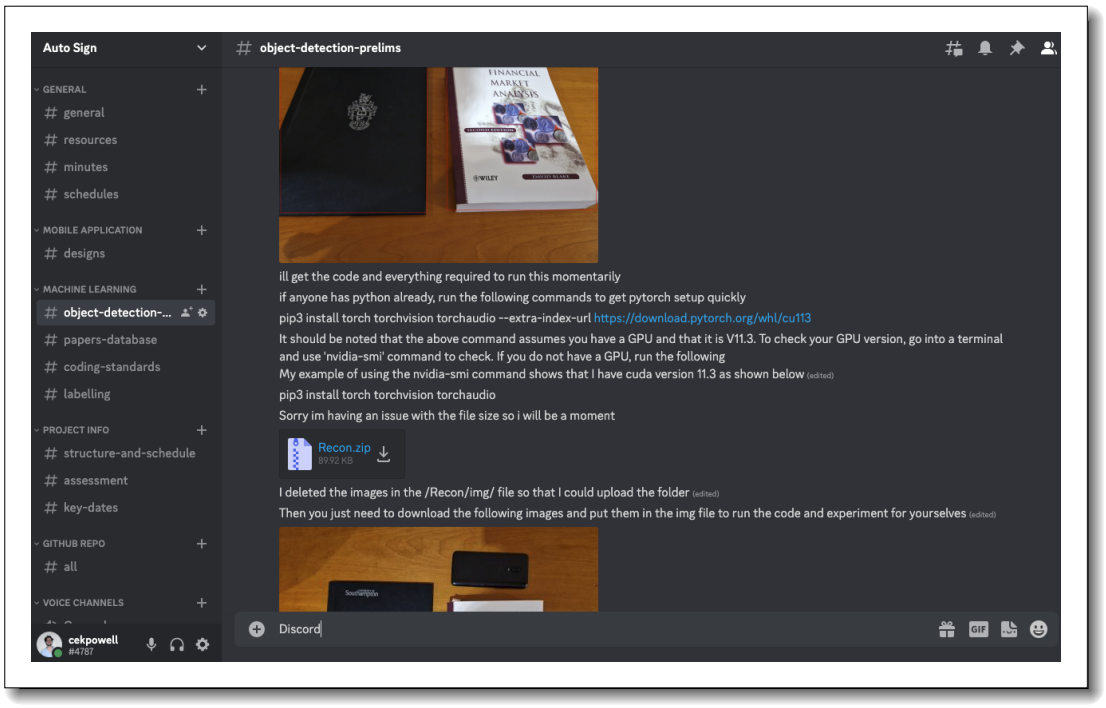


Fig. 5.2. A screenshot taken from the group's discord server showing how information was broken down based on subject

5.3.2 Meetings

Typically, group meetings were held on a bi-weekly basis - at the start and end of every week - with additional meetings scheduled as and when they were required. The first weekly meeting aimed to devise a plan for the upcoming week based on the current status of the project, while the second aimed to reflect on the progress made during the week, and identify any problems encountered in order to address them within the upcoming supervisor meeting. For each meeting, an agenda and minutes were recorded and placed in the group's communication channels for future reference. **Figure 5.3** shows an example of such logging.

5.3.3 Supervisor Relations

In general, the group met with the supervisor on a weekly basis and sent progress updates via email when such a meeting was not possible. These meetings were used to discuss progress and problems encountered from the week prior, with advice being used to formulate the upcoming week's time plan. If additional contact was required, the group would reach out to the project supervisor via Microsoft Teams.

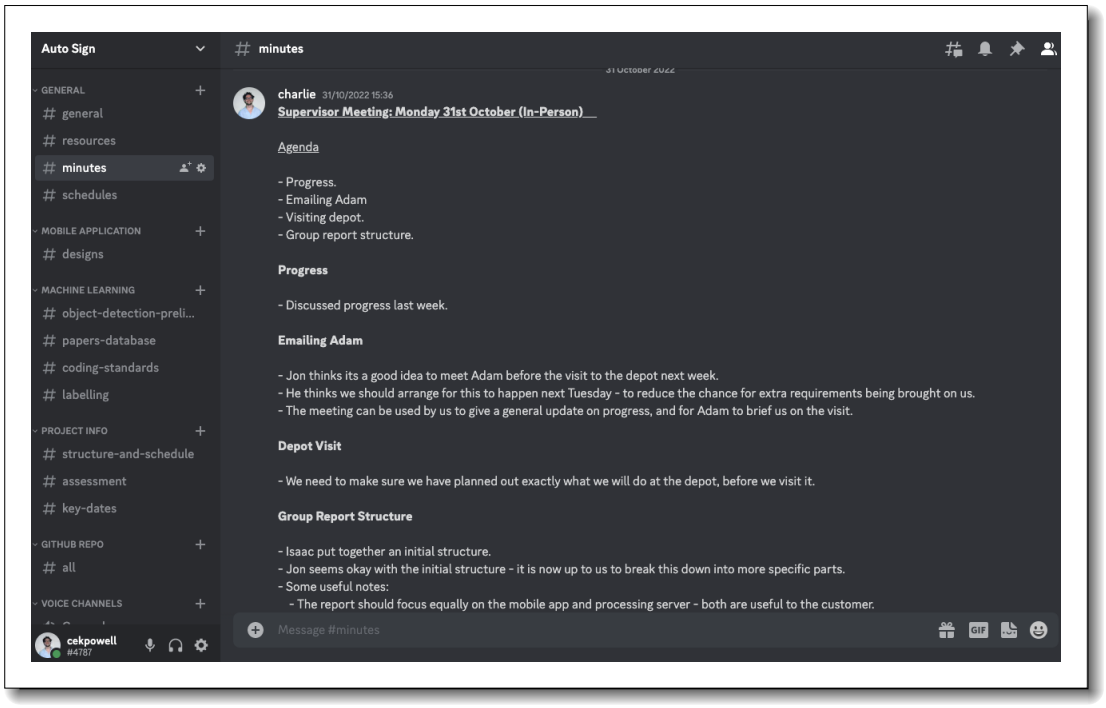


Fig. 5.3. A screenshot taken from the group's discord server showing how meeting logs were recorded

5.3.4 Client Relations

Significant precedence was placed on client relations throughout the project's duration. During the formulation, the group worked closely with the clients to produce a project specification that was both satisfactory, and achievable for the group given the available time and skill sets. For the remainder of the project, the group maintained regular contact with the clients via email, informing them of progress and future plans, and consulting them on major design decisions. Examples of such communication are showcased in **Figure 5.4**. Progress meetings were also held following key project milestones to showcase the proposed system and make adjustments according to received feedback. The positive relations with the clients are also illustrated by the feedback received in **6.2.2**.

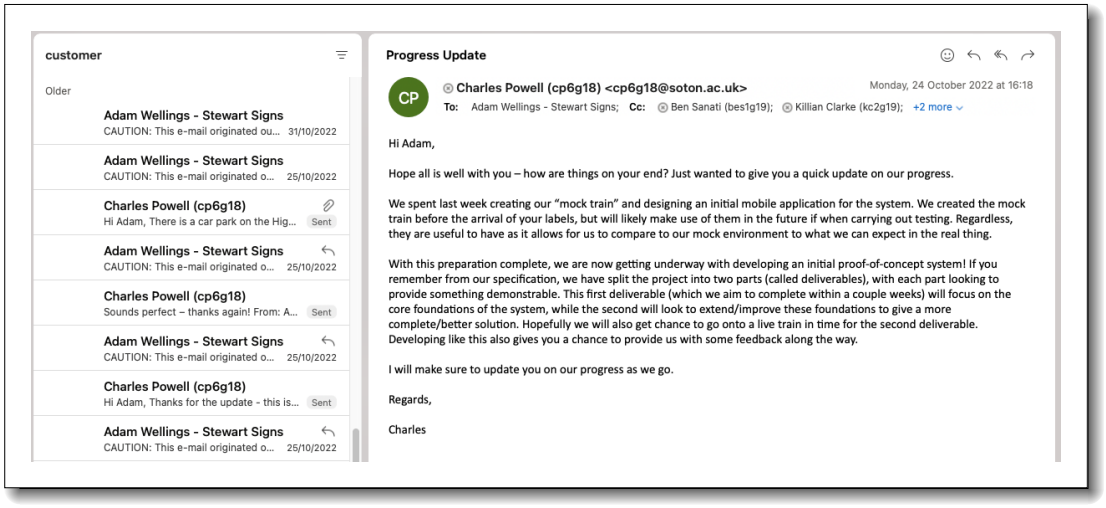


Fig. 5.4. An example communication between the group and project client

5.4 Tools

Table 5.4 describes and provides justification for the project management tools that have been used by the group throughout the project.

Table 5.4: Project Management Tools

Tools		
Name	**Description**	**Justification**
Discord	A messaging application is used as the communication medium between the group members.	Supports lightweight and instant sharing of information in a wide range of formats (e.g., text, files, images, etc), and the creation of channels to group communication based on the subject.
Google Drive	A cloud storage platform used to store and share project-related files amongst group members.	A free-to-use service that is widely accessible and supports a wide range of file formats and sizes.
GitHub	An online Git repository hosting platform used to remotely store/back-up project code-base.	Offers a wide variety of configuration features. Well-established with comprehensive documentation and a large support community.
Overleaf	A collaborative, cloud-based LaTeX editor used to produce and maintain the project's documentation.	Well-established service with a large support community. Supports effective collaboration and management of project documentation.
Mendeley	A reference management tool used to manage research literature and generate a project bibliography.	Supports the storing, organisation and annotation of literature. Integrates with online libraries and automatically generates a bibliography.
TeamGantt	An online Gantt chart tool was used to create and maintain the project's Gantt charts.	A free-to-use platform without impeeding limitations. Supports detailed tasks and time management across an extended duration.

5.5 Risk Assessment

Table 5.5 details the risk assessment conducted on the project. For each technical or non-technical risk that was identified, the following information is given:

- A description of the risk.
- The probability of the risk occurring **P** (from 1, low, to 5, high).
- The severity, **S**, if the risk was to occur(from 1, low to 5, high).
- The risk exposure, **E**, given by **P** $\times$ **S**.
- The steps that have been taken to prevent the risk from occurring, and will be taken to mitigate its effect.

Table 5.5: Risk Assessment

Technical Risks				
Description	**P**	**S**	**E**	**Prevention/Mitigation**
Underestimation of project complexity	3	3	9	Seek appraisal from the project supervisor and customer on the proposed final design. Ensure a strong understanding of all project aspects before beginning implementation and report any issues to the supervisor. Hold regular progress meetings between the group and supervisor to monitor progress and make customer-approved adjustments where required.
Difficulties understanding development tools	3	2	6	Only make use of development tools that at least one group member is proficient in, are well documented, and that have alternatives available in the event they become infeasible.
Issues with personal machine/loss of work	1	5	5	Ensure all project work is regularly backed-up to cloud services or local storage.
Issues with third-party services	2	4	8	Ensure that alternative work can be completed whilst the issue persists and that an alternative is available in the event of complete failure.
Issues with system integration	5	3	15	Make use of tools that group members are proficient in, are well documented and support integration. Ensure adequate time is allocated for integration, and that regular communication occurs between group members.
Difficulties visiting live train(s)	5	3	15	Create a mock train environment using printed pictures of signs to emulate the problem environment. Use this mock environment to develop and test the system until regular visits to a live train can be arranged.
Issues annotating dataset	4	3	12	Make use of well-documented tools to support the annotation of the dataset and outsource this annotation where possible.
Non-Technical Risks				
Description	**P**	**S**	**E**	**Prevention/Mitigation**
Failing to uphold project schedule	3	4	12	Ensure effective, thorough and realistic planning that accounts for setbacks. Maintain regular communication with the supervisor to discuss and track and maintain progress towards this plan. Raise issues with a supervisor as and when they occur and discuss any adjustments that should be made to project requirements.

The supervisor/customer is unwell or unavailable	2	4	8	Maintain regular communication with the supervisor throughout the project duration for continuous feedback/discussion on completed/remaining work. This, therefore, minimises the effect of a period of time without such feedback/discussion.
Team-working issues between group members	3	4	12	Ensure a strong and active communication channel is established within the group, and raise issues immediately with the project supervisor.
Change of project specification/requirements by customer	1	5	5	Ensure that, following the project's formulation, meetings with the customer address progress rather than feedback, and that the supervisor is present to serve as a mediator.
Dissatisfied customer	1	5	5	Seek approval from the customer during the project's formulation and make adjustments where required.
University closure/disruption due to Covid-19	3	2	6	Ensure that the project can be completed from a remote location if required and that communication with the supervisor/customer will not be affected.
Illness/Emergency/ Unforeseen circumstances	1	5	5	Consult with the supervisor to assess the impact of the circumstances and discuss any action that must be taken.

SECTION 6

Project Feedback

6.1 Process

To aid in the evaluation, feedback was sought from potential users and the client. Two potential users and two representatives from the client were each given a guide and a questionnaire to complete. The guide provided an overview of the project, including its goals, and a description of **AutoSign**, including video demonstrations of key features. The survey given to potential users addressed their perceptions of the system in the context of their work life, while the survey given to clients addressed their perceptions of the system relative to the agreed specification, and their experience with the project and its team. This research was conducted in accordance with ERGO II, with participants shown an information sheet prior to their contribution, and documenting their consent within the questionnaire.

6.2 Results

6.2.1 Potential Users

6.2.1.1 Overall Impressions

The overall impression expressed by both potential users was incredibly positive. In particular, repeated reference was made to the simplicity, convenience, and ease-of-use provided by the system, with one participant noting how **AutoSign** could be "*used by everyone in the industry*" as it "*makes it easier for everyone*". Additionally, both participants made reference to how the system reduces the knowledge required by the worker.

6.2.1.2 Preference Over Existing Methods

Both participants expressed a strong preference for **AutoSign** over the current methods, responding with "*strongly agree*" when asked if they think **AutoSign** would improve the current processes, and if they would prefer to use it over existing methods. Participants mainly justified this in terms of the system's ease of use and speed, but also noted the improved organisation and accessibility that **AutoSign** would provide. Additionally, one participant made reference to how the system alone, without any automation, would still be a very useful tool.

6.2.1.3 Additional Features

The participants noted a desire for the inclusion of technical drawings to aid the inspection and fitting process, as well as having these technical drawings linked to the sign-purchasing system. Note, however, that the lack of additional features suggested by the participants is not indicative of a complete solution.

6.2.1.4 Issues with Mandatory Use

No concerns were raised by the participants with regards to mandatory use, with both answering "*strongly agree*" in response to the statement "*I have no concerns with an application like AutoSign being mandatory for my work*". Furthermore, the users expressed a welcoming attitude to new technologies in their workplace, especially those which make their job easier.

6.2.2 Project Clients

6.2.2.1 Overall Impressions

The overall impressions of both clients were incredibly positive towards both the developed system and their experience with the project and team. Most notably, references were made throughout to their desire to develop the project further and how this project has given them the confidence to believe such work is possible.

6.2.2.2 Perceptions towards Final System

Both participants stated that they were "*very satisfied*" with **AutoSign** and that the system achieves what was desired as a proof-of-concept. Additionally, when reflecting on the original specification, both clients noted how the system meets their expectations and provides all previously agreed functionality. When considering future areas for development, the clients highlighted the improvement of **AutoSign**'s detection capabilities, and integration with pre-existing architecture, such as the Stewart Signs online store for the purchasing of signs.

6.2.2.3 Perceptions towards Project and Team Experience

The clients stated that they were "*very satisfied*" with the overall project experience and that their interaction with the team had been "*excellent*". In particular, the clients noted how the project "*opens an opportunity to a real-world tool*", and provides a "*foundation for [Stewart Signs] to plan the next steps*". With regards to project management, both clients expressed positive feelings towards the group's apparent management of the project, and their interactions with the group, though did note a desire for additional project milestones and reviews.

SECTION 7

Evaluation

7.1 Critical Evaluation

7.1.1 Requirement Analysis

Provided here is an analysis of the requirements that were placed on the system within **2 - System Design**, which considers their satisfaction and provides justification.

7.1.1.1 Application Requirements

Table 7.1: Application Requirements Analysis

ID	Title	Priority	Status	Justification
R1	Inspection Capture	MUST	FULLY MET	3.2.1.8 & 3.2.2.6 & 4.2.3.2 & 3.5.1 & 4.6.1
R2	Inspection Upload	MUST	FULLY MET	3.2.1.8 & 3.2.2.6 & 3.5.1 & 4.6.1
R3	Inspection Viewing	MUST	FULLY MET	3.2.1.9 & 4.2.2.10 & 3.5.1 & 4.6.1
R4	Remediation Logging	SHOULD	FULLY MET	4.2.2.7 & 4.2.3.4 & 4.6.1
R5	Remediation Upload	SHOULD	FULLY MET	4.2.2.7 & 4.2.3.4 & 4.6.1
R6	Remediation Viewing	SHOULD	FULLY MET	4.2.2.9 & 4.6.1
R7	Vehicle Status	SHOULD	FULLY MET	4.2.2.6 & 4.6.1
R8	Purchasing Signs	SHOULD	FULLY MET	4.2.2.7 & 4.2.2.8 & 4.2.3.3 4.6.1
R9	Adding Train Vehicles	COULD	NOT MET	Prioritisation of features, time constraints and client value.
R10	Inspection Capture Assistance	COULD	NOT MET	Prioritisation of features, time constraints and client value.

7.1.1.2 Processing Server Requirements

Table 7.2: Processing Server Requirements Analysis

ID	Title	Priority	Status	Justification
R11	Missing Sign Detection	MUST	FULLY MET	3.3 & 4.3 & 3.4 & 4.5 & 3.5.2 & 4.6.2

R12	Inspection Result	MUST	FULLY MET	3.4.4 & 4.5.6 & 3.5.2 & 4.6.2
R13	Damaged Sign Detection	SHOULD	FULLY MET	4.4 & 4.5 & 4.6.2
R14	Video Processing	COULD	FULLY MET	4.5.5 & 4.6.2
R15	Incorrectly Positioned Sign Detection	WONT	NOT MET	Prioritisation of features, time constraints and client value.

7.1.1.3 Cloud Server Requirements

Table 7.3: Cloud Server Requirements Analysis

ID	Title	Priority	Status	Justification
R16	Database	MUST	FULLY MET	3.3 & 4.1
R17	File Storage	MUST	FULLY MET	3.4 & 4.2
R18	Communication	MUST	FULLY MET	3.1.4 & 4.1.2

7.1.2 Project Goal Analysis

Provided here is an analysis of the project's goals, which reflects on the completion, achievements, implications and limitations of each.

7.1.2.1 PG1: Developing The System

Table 7.4: Project Goal Analysis: PG1

ID	Project Goal	Status
PG1	Develop a proof-of-concept system that supports the inspection of a train vehicle via collected photos/videos, the creation of a digital report detailing missing and damaged signs, the purchasing of replacement signs, the logging of remediations, and the viewing of a vehicle's current status and history of inspections and remediations.	FULLY MET
	Achievements	
A1	Designed and developed a mobile application using modern and relevant technologies that supports the inspection, purchasing and remediation of a train vehicle's signs, and viewing of related information.	

A2	Developed a number of custom datasets, according to leading research, to assist in the development of problem-specific machine learning algorithms.
A3	Developed techniques for detecting missing and damaged signs within train vehicles that perform incredibly well when tested.
A4	Developed a system architecture that links components together to provide a functioning solution.
	Implications
I1	This project's work has provided a strong foundation for the future development of the approach, which is able to utilize the established components and methodologies.
	Limitations
L1	The problem area was constrained in many aspects to make the development of the proof-of-concept system achievable with the available skill sets and time period.

7.1.2.2 PG2: Evaluating System Performance

Table 7.5: Project Goal Analysis: PG2

ID	Project Goal	Status
PG2	Evaluate the performance of the developed system to detect the presence and type of non-conformance within a train vehicle, in terms of both time and accuracy.	**MOSTLY MET**
	Achievements	
A5	The machine learning techniques implemented throughout the project have been extensively tested and evaluated, and have demonstrated excellent performance.	
A6	The performance of the mobile application has been successfully evaluated according to the scenario cases defined during its design.	
A7	The entire system has been tested in a live train, and mock environments, where it successfully demonstrated an ability to detect missing and damaged signs. This testing also evaluated the integration of the system's various components.	
	Implications	
I2	The success of the application, machine learning algorithms, and **AutoSign** in general has shown the approach to be viable, and worthy of further development.	
	Limitations	
L2	The nature of the project and rail industry limited the amount of field testing that could be performed within the duration of the project.	

7.1.2.3 PG3: Demonstrating Commercial Feasibility

Table 7.6: Project Goal Analysis: PG3

ID	Project Goal	Status
PG3	Demonstrate the feasibility of the system as a commercial product.	**FULLY MET**

Achievements	
A8	The ability of the group to develop a functioning and successful proof-of-concept system within the available time has demonstrated the feasibility of such a system as a commercial and professional product.
Implications	
I3	This project's work provides justification for further development to be carried out to transition the system from a proof-of-concept to a complete solution.
Limitations	
L3	Notable issues that must be considered prior to commercial use, such as the scalability of the solution and usability concerns, have not been addressed by this project.

7.1.2.4 PG4: Considering User Perception

Table 7.7: Project Goal Analysis: PG4

ID	Project Goal	Status
PG4	Consider the perception of potential users towards the developed system.	**PARTIALLY MET**
Achievements		
A9	Feedback gathered from a small number of potential users has shown that perceptions towards **AutoSign** are incredibly positive and that users believe it would be an improvement on the current methods.	
Implications		
I4	The positive feedback received provides justification to develop the approach beyond a proof-of-concept, and into a commercial and professional product.	
Limitations		
L4	The number of potential users surveyed was small and was not based on real-world use of the system.	

7.2 Comparative Evaluation

The following list provides a comparative analysis of **AutoSign** with the existing approach, according to a number of criteria.

- **Inspection Time**: *Average time taken to perform an inspection.*
  - The time taken to perform an inspection with **AutoSign** is quicker than a manual approach. This is because it takes less time to record a video/capture an image of a group of signs, in comparison to analysing each sign independently.
- **Reliability**: *The precision and accuracy of the system for detecting non-conformances.*
  - **AutoSign** provides more definitive and consistent precision and accuracy in comparison to manual methods. Manual methods, however, do provide more specificity with regard to non-conformance details.
- **Minimum Skill Required**: *The level of skill/knowledge required to complete the method.*

  - A worker is able to perform an inspection using **AutoSign** without any prior knowledge of train signs, or their regulations, in comparison to manual methods which require in-depth knowledge of regulations and sign locations.

- **Data Availability**: *Refers to the accessibility to the data relating to a particular train vehicle.*

  - **AutoSign** stores its data within a cloud server, making it highly accessible as only an internet connection is required. In comparison, the manual methods require the individual to be in the same location as the data, or for it to be sent to them.

- **Data Safety**: *Refers to the likelihood of data loss/theft.*

  - **AutoSign** stores its data within Firebase, a BaaS provider backed by Google. Firebase offers excellent security, making loss of data/breaches highly unlikely. In a manual approach, documents can easily be misplaced and lost.

- **Train Carriage Variety**: *Refers to the variety of train vehicles that inspections can be performed on.*

  - To carry out an inspection on a given vehicle, an inspector is required to have an understanding of the vehicle's signs as well as their locations and governing standards. In **AutoSign**, the system must be trained on a dataset of appropriate signs for the vehicle, and a gold-standard definition of the vehicle defined in the cloud server.

This discussion illustrates that **AutoSign** provides a better alternative to the current methods. This stems from the many advantages a digital system has over an analogue counterpart, as well as the evidence that a machine learning-based approach to detecting non-conforming signs is not only viable but also has the potential to vastly outperform the current, manual approach.

Additionally, while **AutoSign** does not easily support additional train carriages, this can be attributed to its nature as a proof-of-concept system. It is reasonable to suggest that if development were to continue, a more structured and efficient workflow could be defined to resolve this issue.

7.3 Reflective Evaluation

7.3.1 Project Successes

- **System Development**: This project designed and developed a successful proof-of-concept to address a real-world problem.

- **Commercial Feasibility**: The ability of this project to develop a successful proof-of-concept has demonstrated the commercial feasibility of the product, and provided justification for future work.

- **Positive Feedback**: The highly positive feedback received by the sample of potential users and clients illustrates the success of the project's work.

- **Effective Time Management**: The group demonstrated excellent time management and planning throughout the project to deliver the optimal system for the client within a constrained time period.

- **Personal Development**: Through the course of the project, group members have been able to develop a range of technical and non-technical skills.

7.3.2 Project Shortcomings

- **System Testing**: Due to the nature of the project, and the rail industry, the group were not able to perform extensive testing of **AutoSign** on a live train.
- **User Feedback**: The group were only able to gain limited feedback from a small set of potential users on **AutoSign**.

SECTION 8

Client Recommendations

Provided here are the team's recommendations for the client regarding the transition from a proof-of-concept into a complete solution. These aspects are divided into development and testing, and aim to alleviate constraints placed on this project, and address aspects outside of its scope.

8.1 Development

8.1.1 Initialization & Deployment Workflow

For wide-scale adoption and use, it is necessary that the system can be initialized and deployed across any given train vehicle efficiently. Such initialization requires the collection of data regarding both the appearance and location of signs within the vehicle. This is for the purposes of training the required machine learning models and defining the ground truth, respectively. It is therefore essential that standardised workflows are defined to support the efficient retrieval of this data. While such collection is expensive in terms of both time and resources, **Section - 8.1.5** explores a potential approach to alleviate these concerns.

8.1.2 Improved Processor

The primary function of the system is to process inspection footage and detect non-conformances. This project's work focused on the detection of missing and damaged signs, with a small selection of damage types being considered. To be deployed commercially, the processor must provide results that match those produced by a human inspector. As such, the processor requires extension to support the detection of a wider range of non-conformances, such as incorrectly positioned signs, and provide more detailed reports, such as a greater selection of damage types. Achieving this functionality relies on more expansive data collection that was not possible during this project due to time and facility access constraints.

8.1.3 Improved Interfaces

The application developed within this project contains limited features which seek to demonstrate the system's overall feasibility and value to to the client. To be a deployed commercially, additional features, such as a physical mapping of vehicle status, technical drawings/documentation and linking to the client's existing systems must be implemented. Additional user testing would also assist in the identification of required features, as discussed in **Section - 8.2.2**.

Additionally, alternative and/or supplementary interfaces could be considered that provide alternative functionality, such as an administrative application that supports greater management capabilities.

8.1.4 Additional Data Collection

As demonstrated in this project, the deployment of machine learning techniques requires the collection of problem-specific data. While this project gathered a limited amount of data to develop a proof-of-concept, a more representative set of data must be collected to provide more generalisation and improve the robustness of the models trained on the dataset.

8.1.5 Process Digitisation

This project sought to digitise the existing methods of train sign inspection, purchasing and remediation with the use of machine learning techniques. While both these aspects could continue development in tangent, a worthy consideration is the focus on purely the digitisation of the existing methods, with machine learning integration occurring at a later stage.

In such a system, the inspection process would be updated to allow for the user to manually log the status of the vehicle based on their own observations, and capture supporting footage. This approach not only allows for the system to be deployed while the time-consuming development of the required machine learning techniques is carried out, but also automates the process of data collection.

8.2 Testing

8.2.1 Additional Field Testing

While the approach produced positive results in field testing, its use was limited, and was not properly deployed on a live train. Before commercial deployment, extensive field testing within the real world environment must be carried out to better assess the systems performance, and address issues not identified within this project.

8.2.2 Additional User Testing

Despite receiving positive feedback from a small set of potential users, perceptions towards the approach have not been effectively assessed. By carrying out more extensive testing, user perceptions can be analysed in more detail, allowing the system to be adjusted based on user requirements. Such work would improve the chances of user adoption, which is vital for commercial success.

SECTION 9

Conclusion

To address the inefficiencies regarding the current methods of manual train sign inspection, purchasing and remediation, this project sought to develop a proof-of-concept system that serves as a semi-automated improvement upon the existing approach.

Based on consultation with the client, the group designed and developed **AutoSign** - a system that supports the inspection of a train vehicle, the identification of missing and/or damaged signs, the purchasing of replacement signs, and the logging of remediations. In particular, **AutoSign** implements a three-component architecture, consisting of a mobile application to be used by workers, a processing server that makes use of custom datasets and algorithms to process inspection footage, and a cloud server to host system data.

In testing, **AutoSign**'s application was shown to function according to its requirements, while its processing techniques were shown to be highly effective and accurate. When used in live trains and mock environments, **AutoSign** also demonstrated an ability to function as a single, cohesive system, and was able to correctly identify real-world non-conformances. These findings, and given the time available to develop **AutoSign**, demonstrate the feasibility of the approach in the commercial world and provide justification for its development beyond a proof-of-concept.

Furthermore, **AutoSign** received positive feedback from potential users, who noted how such a system would improve their workflow and their preference for it over existing methods. This analysis is supported by the project's comparative evaluation of **AutoSign**, which shows it to out-perform current approaches against a number of different criteria.

Based on this project's findings, its final discussion presents a list of recommendations to the client regarding the transition of the proof-of-concept into a complete solution. This discussion aims to alleviate the constraints placed upon this project and address aspects outside of its reach.

While the scope for future development is large, this project's work remains significant for the client, and the rail industry. The conclusion of this project is the successful development of a proof-of-concept system that addresses a novel problem and demonstrates the capacity for a solution. This project has successfully shown that a system such as **AutoSign** is a feasible solution to the problems it first presented, and identified what is required to complete the transition from a proof-of-concept to a commercial product.

Bibliography

[1] Laith Alzubaidi, Jinglan Zhang, Amjad J Humaidi, Ayad Al-Dujaili, Ye Duan, Omran Al-Shamma, José Santamar\'\ia, Mohammed A Fadhel, Muthana Al-Amidie, and Laith Farhan. "Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions". In: *Journal of big Data* 8.1 (2021), pp. 1–74.

[2] Hangbo Bao, Li Dong, and Furu Wei. "Beit: Bert pre-training of image transformers". In: *arXiv preprint arXiv:2106.08254* (2021).

[3] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. "Learning long-term dependencies with gradient descent is difficult". In: *IEEE transactions on neural networks* 5.2 (1994), pp. 157–166.

[4] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. "End-to-end object detection with transformers". In: *European conference on computer vision*. 2020, pp. 213–229.

[5] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. "Empirical evaluation of gated recurrent neural networks on sequence modeling". In: *arXiv preprint arXiv:1412.3555* (2014).

[6] Antonio Criminisi, Ian Reid, and Andrew Zisserman. "A plane measuring device". In: *Image and Vision Computing* 17.8 (1999), pp. 625–634.

[7] David Kriegman. *Homography Estimation*. 2007. URL: https://cseweb.ucsd.edu//classes/wi07/cse252a/homography_estimation/homography_estimation.pdf.

[8] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. "Imagenet: A large-scale hierarchical image database". In: *2009 IEEE conference on computer vision and pattern recognition*. 2009, pp. 248–255.

[9] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. "An image is worth 16x16 words: Transformers for image recognition at scale". In: *arXiv preprint arXiv:2010.11929* (2020).

[10] Mark Everingham, Luc Van Gool, Christopher K I Williams, John Winn, and Andrew Zisserman. "The pascal visual object classes (voc) challenge". In: *International journal of computer vision* 88.2 (2010), pp. 303–338.

[11] *fiverr*. URL: https://www.fiverr.com/.

[12] Ross Girshick. "Fast r-cnn". In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1440–1448.

[13] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 580–587.

[14] Google Developers. *Firebase: Choose a data structure*. 2022. URL: https://firebase.google.com/docs/firestore/manage-data/structure-data.

[15] Priya Goyal, Mathilde Caron, Benjamin Lefaudeux, Min Xu, Pengchao Wang, Vivek Pai, Mannat Singh, Vitaliy Liptchinsky, Ishan Misra, Armand Joulin, et al. "Self-supervised pretraining of visual features in the wild". In: *arXiv preprint arXiv:2103.01988* (2021).

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[17] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. "Improving neural networks by preventing co-adaptation of feature detectors". In: *arXiv preprint arXiv:1207.0580* (2012).

[18] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[19] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. "Mobilenets: Efficient convolutional neural networks for mobile vision applications". In: *arXiv preprint arXiv:1704.04861* (2017).

[20] Sergey Ioffe and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *International conference on machine learning*. 2015, pp. 448–456.

[21] Licheng Jiao, Fan Zhang, Fang Liu, Shuyuan Yang, Lingling Li, Zhixi Feng, and Rong Qu. "A survey of deep learning-based object detection". In: *IEEE access* 7 (2019), pp. 128837–128868.

[22] Justin M Johnson and Taghi M Khoshgoftaar. "Survey on deep learning with class imbalance". In: *Journal of Big Data* 6.1 (2019), pp. 1–54.

[23] Asifullah Khan, Anabia Sohail, Umme Zahoora, and Aqsa Saeed Qureshi. "A survey of the recent architectures of deep convolutional neural networks". In: *Artificial intelligence review* 53.8 (2020), pp. 5455–5516.

[24] Alex Krizhevsky, Geoffrey Hinton, et al. "Learning multiple layers of features from tiny images". In: (2009).

[25] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Communications of the ACM* 60.6 (2017), pp. 84–90.

[26] Labelbox. *LabelBox | The Leading AI Data Engine Platform*. URL: https://app.labelbox.com/projects.

[27] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[28] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. "Feature pyramid networks for object detection". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 2117–2125.

[29] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. "Focal loss for dense object detection". In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 2980–2988.

[30] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. "Microsoft coco: Common objects in context". In: *European conference on computer vision*. 2014, pp. 740–755.

[31] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. "Path aggregation network for instance segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 8759–8768.

[32] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. "Swin transformer: Hierarchical vision transformer using shifted windows". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 10012–10022.

[33] Wenhan Luo, Junliang Xing, Anton Milan, Xiaoqin Zhang, Wei Liu, and Tae-Kyun Kim. "Multiple object tracking: A literature review". In: *Artificial Intelligence* 293 (2021), p. 103448.

[34] Kemal Oksuz, Baris Can Cam, Sinan Kalkan, and Emre Akbas. "Imbalance Problems in Object Detection: A Review". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43.10 (2021), pp. 3388–3415. DOI: 10.1109/TPAMI.2020.2981890.

[35] Rafael Padilla, Sergio L Netto, and Eduardo A B da Silva. "A Survey on Performance Metrics for Object-Detection Algorithms". In: *2020 International Conference on Systems, Signals and Image Processing (IWSSIP)*. 2020, pp. 237–242. DOI: 10.1109/IWSSIP48289.2020.9145130.

[36] Luis Perez and Jason Wang. "The effectiveness of data augmentation in image classification using deep learning". In: *arXiv preprint arXiv:1712.04621* (2017).

[37] Waseem Rawat and Zenghui Wang. "Deep convolutional neural networks for image classification: A comprehensive review". In: *Neural computation* 29.9 (2017), pp. 2352–2449.

[38] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. "You only look once: Unified, real-time object detection". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 779–788.

[39] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. "Faster r-cnn: Towards real-time object detection with region proposal networks". In: *Advances in neural information processing systems* 28 (2015).

[40] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. "Learning representations by back-propagating errors". In: *nature* 323.6088 (1986), pp. 533–536.

[41] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. "Learning representations by back-propagating errors". In: *nature* 323.6088 (1986), pp. 533–536.

[42] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. "Mobilenetv2: Inverted residuals and linear bottlenecks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 4510–4520.

[43] J Semple and G Kneebone. "Algebraic Projective Geometry". In: *Oxford University Press, Oxford* (1979).

[44] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (2014).

[45] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. "Going deeper with convolutions". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1–9.

[46] Antonio Torralba and Alexei A Efros. "Unbiased look at dataset bias". In: *CVPR 2011*. 2011, pp. 1521–1528. DOI: 10.1109/CVPR.2011.5995347.

[47] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[48] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. "YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors". In: *arXiv preprint arXiv:2207.02696* (July 2022).

[49] Chien-Yao Wang, Hong-Yuan Mark Liao, and I-Hau Yeh. "Designing Network Design Strategies Through Gradient Path Analysis". In: *arXiv preprint arXiv:2211.04800* (2022).

[50] Xiaolong Wang, Abhinav Shrivastava, and Abhinav Gupta. "A-fast-rcnn: Hard positive generation via adversary for object detection". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 2606–2615.

[51] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. "Simple online and realtime tracking with a deep association metric". In: *2017 IEEE international conference on image processing (ICIP)*. 2017, pp. 3645–3649.

[52] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. "Huggingface's transformers: State-of-the-art natural language processing". In: *arXiv preprint arXiv:1910.03771* (2019).

[53] Kai Xiao, Logan Engstrom, Andrew Ilyas, and Aleksander Madry. "Noise or signal: The role of image backgrounds in object recognition". In: *arXiv preprint arXiv:2006.09994* (2020).

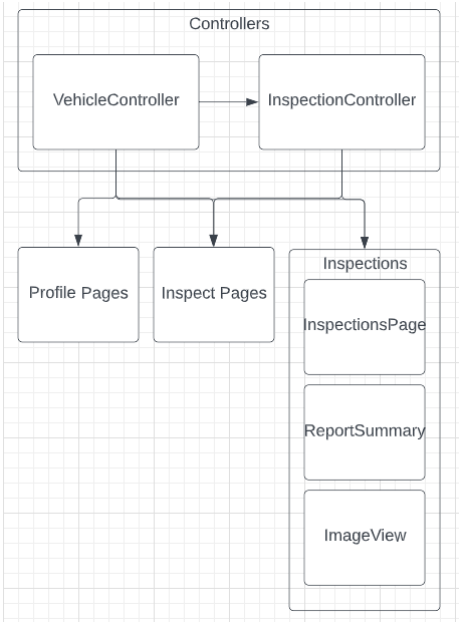
[54] S S A Zaidi, M S Ansari, A Aslam, N Kanwal, M Asghar, and B Lee. "A survey of modern deep learning based object detection models". In: *arXiv preprint arXiv:2104.11892* (2021).

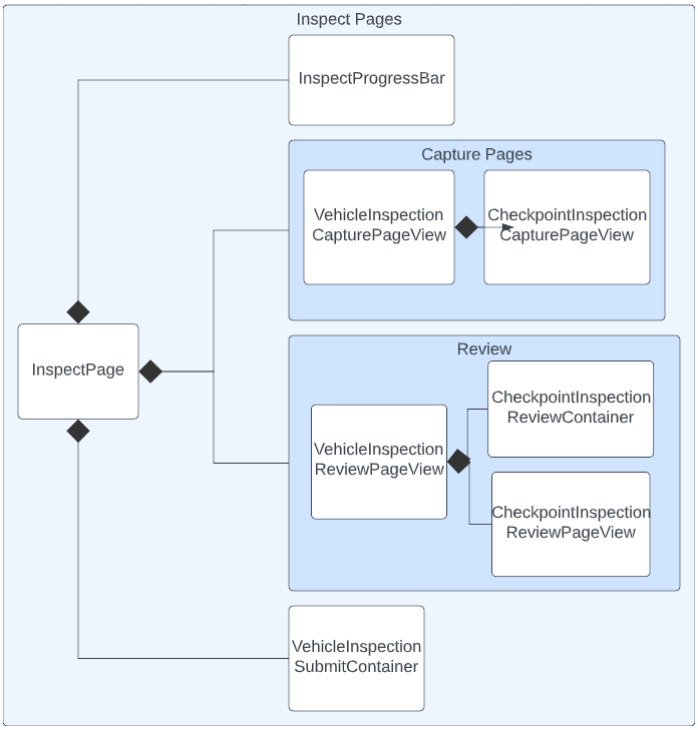
[55] Mu Zhu. "Recall, precision and average precision". In: *Department of Statistics and Actuarial Science, University of Waterloo, Waterloo* 2.30 (2004), p. 6.

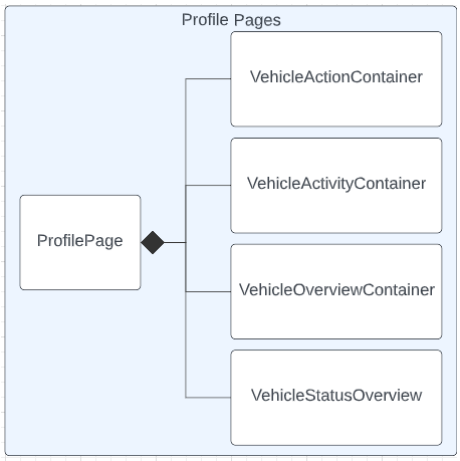
SECTION A

Application

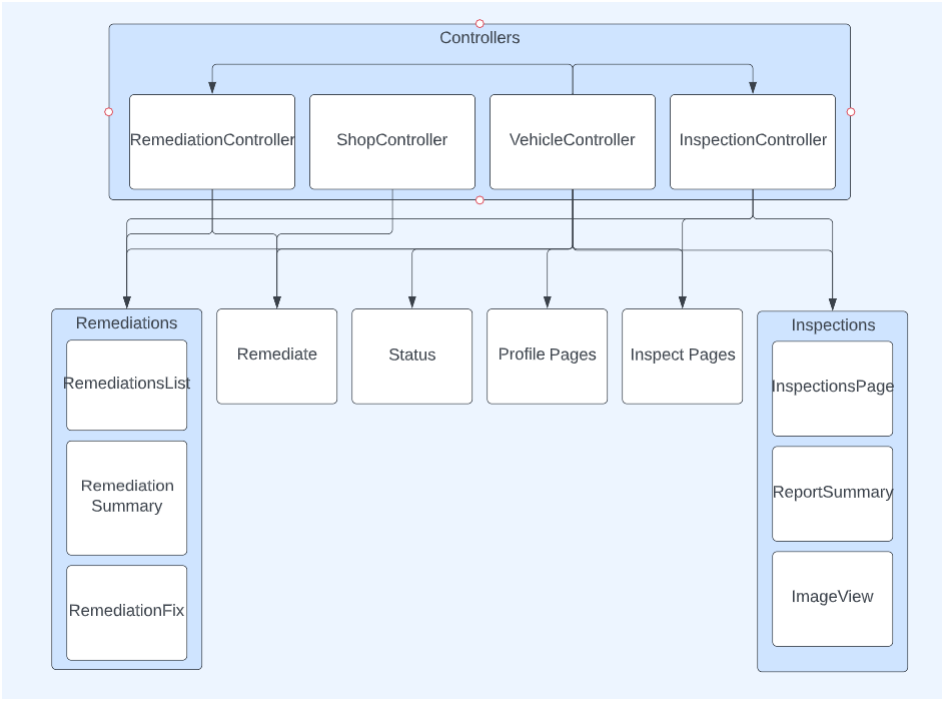
A.1 Deliverable 1 Class Diagrams

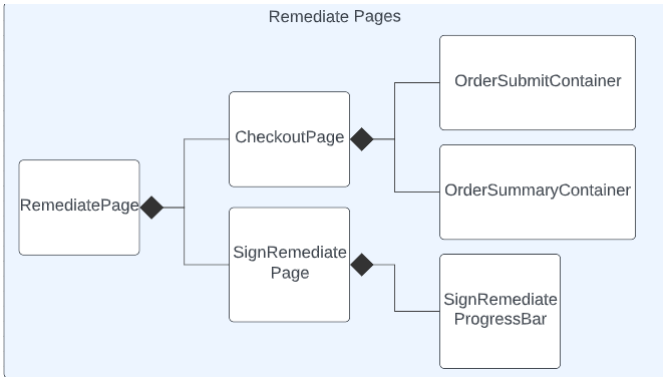


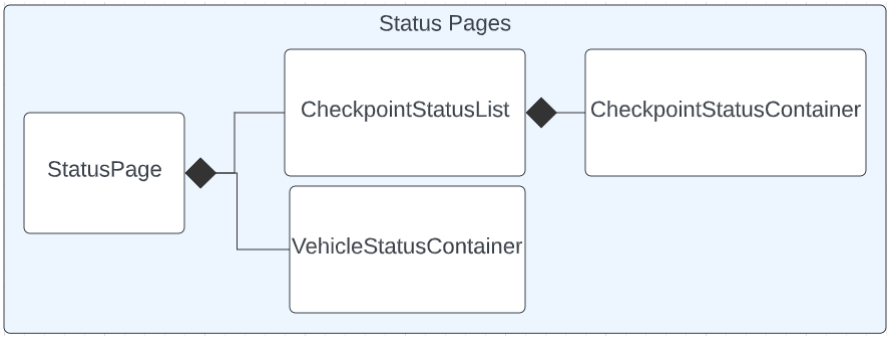




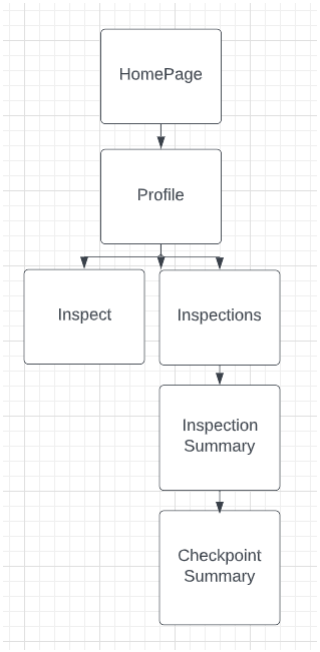
A.2 Deliverable 2 Class Diagrams

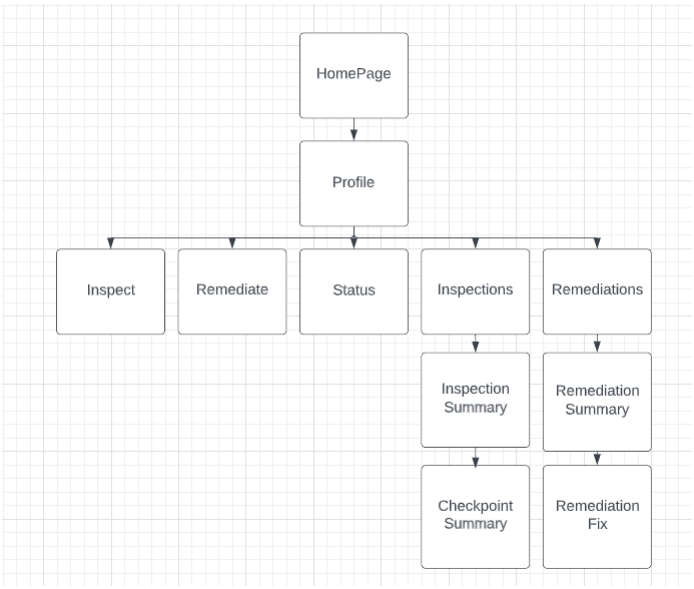






A.3 Routing Diagrams





SECTION B

DATASETS

B.1 DATASET INSTRUCTIONS

Dataset Instructions

Instructions for AutoSign-Mock

Summary

This is the labelling project for the AutoSign mock environment created on 18/10/22. 466 images were taken and uploaded to the dataset. Each of these images requires bounding boxes to be drawn around the signs, with each bounding box being annotated. The general standard for bounding box drawing is to ensure the red boundary around each sign is within the bounding box.

Workflow

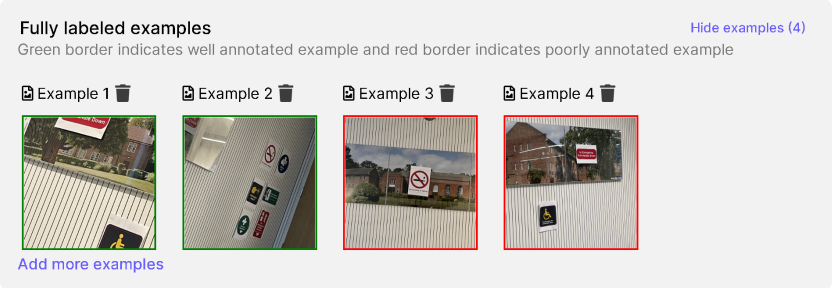
Labelling

- Go to your batch for labelling
- Draw bounding boxes around each sign, with the red boundary defining the edge of the label (see well-annotated examples below)
- Label each BBox drawn
- Ensure every sign in the image is labelled (not just the signs with the most semantic value)

Review

- Review other people's batches and verify/fix them based on whether they conform to the standard set above

Fully Labelled Examples



Label Taxonomy

A total of 11 possible label annotations, including:

- In Emergency Pull Handle Down
- No Smoking or Vaping
- Call for Aid
- Push Button when Lit
- In Emergency Pull to Operate
- First Aid Equipment
- Fire Extinguisher Beneath the Seats
- Emergency Exit Forwards
- Priority Space for Wheelchair Users
- Train to Track Evacuation Instructions
- Please Give Up this Seat for Someone Less able to Stand than You

Dataset Instructions 2

SECTION C

Processing Server

C.1 Model Hyperparameters

C.1.1 Mock Object Detection Model

The finetuned hyperparameters for the YOLOv7 [48] model trained on the mock dataset are shown below.

- `lr0`: 0.01
- `lrf`: 0.2
- `momentum`: 0.937
- `weight_decay`: 0.0005
- `warmup_epochs`: 3.0
- `warmup_momentum`: 0.8
- `warmup_bias_lr`: 0.1
- `box`: 0.05
- `cls`: 0.3
- `cls_pw`: 1.0
- `obj`: 0.7
- `obj_pw`: 1.0
- `iou_t`: 0.2
- `epochs`: 50
- `batch_size`: 16
- `img_size`: (1280, 1280)
- `workers`: 4

C.1.2 Object Detection Model

- `lr0`: 0.01
- `lrf`: 0.2
- `momentum`: 0.937
- `weight_decay`: 0.0005
- `warmup_epochs`: 3.0
- `warmup_momentum`: 0.8
- `warmup_bias_lr`: 0.1
- `box`: 0.05
- `cls`: 0.3
- `cls_pw`: 1.0
- `obj`: 0.7
- `obj_pw`: 1.0
- `iou_t`: 0.2
- `epochs`: 200
- `batch_size`: 16
- `img_size`: (1280, 1280)
- `workers`: 4

C.1.3 Damage Classification Models

C.1.3.1 Fine Classification Model

The finetuned hyperparameters for the fine class classification BEiT [2] model trained on the damaged dataset are shown below.

- `lr`: 5×10^{-6}
- `warmup` = 10
- `weight_decay` = 0.0005
- `batch_size` = 256
- `warmup_ratio` = 0.01
- `grad_accum_steps` = 4
- `num_epochs`=50
- `patch_size`=16
- `img_size`: (224, 224)
- `attention_heads`: 12
- `workers`: 4

C.1.3.2 Coarse Classification Model

The finetuned hyperparameters for the coarse class classification BEiT [2] model trained on the damaged dataset are shown below.

- `lr`: 5×10^{-5}
- `warmup` = 10
- `weight_decay` = 0.0005
- `batch_size` = 256
- `warmup_ratio` = 0.01
- `grad_accum_steps` = 4
- `num_epochs`=50
- `patch_size`=16
- `img_size`: (224, 224)
- `attention_heads`: 12
- `workers`: 4

C.2 Sign-Logic Algorithm

```
procedure SignLogic(filtered_list)
    camera_direction ← CameraMovement(filtered_list)
    unique_signs ← ∅
    for frame in video do
        if frame ≠ 0 then
            for sign in frame do
                current_sign ← filtered_list[frame][sign]
                prior_sign ← filtered_list[frame-1][sign]
                if current_sign not in prior_sign then
                    unique_signs.insert(current_sign)
                else
                    sign_dir ← Direction(current_sign, prior_sign)
                    if sign_dir == camera_direction then
                        unique_signs.insert(filtered_list[frame])
        prior_frame ← filtered_list[frame])
```

C.3 Processing Server Workflow

C.3.1 Deliverable 1

Algorithm C.1 Deliverable 1 Processing Server workflow

```
1: procedure RunServer
2:     while True do
3:         wait(2s)
4:         inspections ← Firestore.getUnprocessedVehiceInspections()
5:
6:         for inspection in inspections do
7:             ProcessVehicleInspection(inspection)
8:
9: procedure ProcessInspection(inspection)
10:     vehicle_conforming ← True
11:     checkpoints ← Firestore.getCheckpoints(inspection)
12:
13:     for checkpoint in checkpoints do
14:         checkpoint_conforming ← ProcessCheckpoint(checkpoint)
15:
16:         if !checkpoint_conforming then
17:             vehicle_conforming ← False
18:
19:     if vehicle_conforming then
20:         inspection.conformanceStatus ← "conforming"
21:     else
22:         inspection.conformanceStatus ← "non-conforming"
23:     Firestore.updateVehicleInspection(inspection)
24:
25: procedure ProcessCheckpoint(checkpoint)
26:     checkpoint_conforming ← True
27:     image ← Storage.getCheckpointImage(checkpoint)
28:     detected_signs ← detectSigns(image)
29:
30:     for sign in checkpoint.signs do
31:         if sign in detected_signs then
32:             sign.conformanceStatus ← "conforming"
33:
34:         else
35:             sign.conformanceStatus ← "non-conforming"
36:             checkpoint_conforming ← False
37:
38:     if checkpoint_conforming then
39:         checkpoint.conformanceStatus ← "conforming"
40:         return True
41:     else
42:         checkpoint.conformanceStatus ← "non-conforming"
43:         return False
```

C.3.2 Deliverable 2

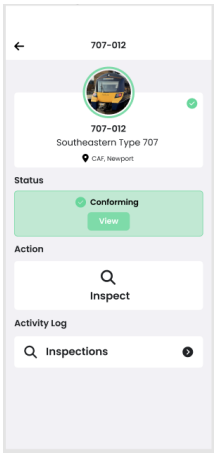
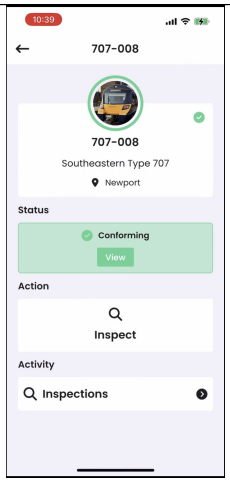
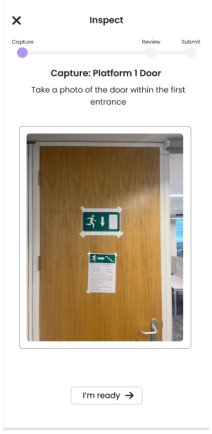
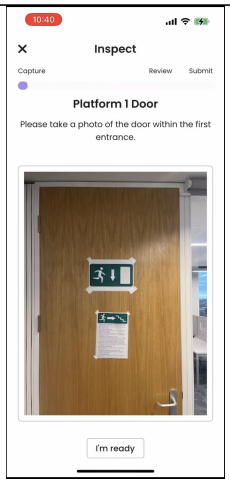
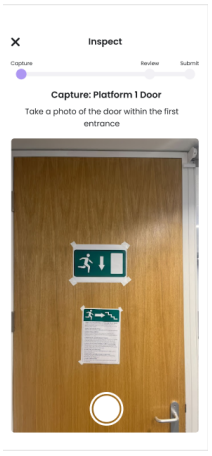
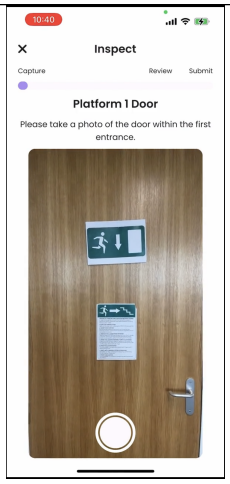
Algorithm C.2 Deliverable 2 Processing Server workflow

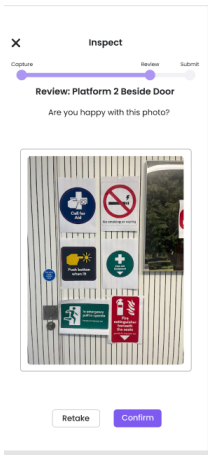
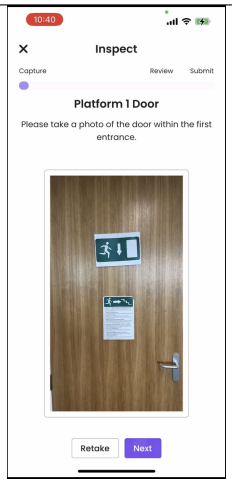
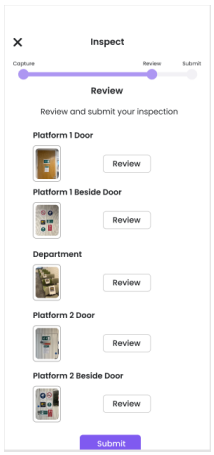
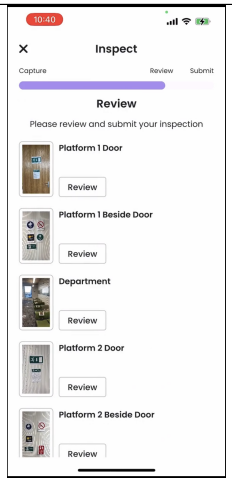
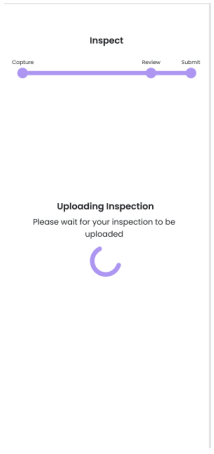
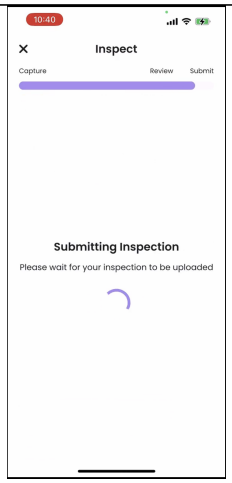
```
1:  procedure ProcessCheckpoint(checkpoint)
2:      checkpoint_conforming ← True
3:
4:      if checkpoint.captureType == "photo" then
5:          image ← Storage.getCheckpointImage(checkpoint)
6:
7:          detected_signs, norm_signs ← detectSignsFromImage(image)
8:
9:      else
10:         video ← Storage.getCheckpointImage(checkpoint)
11:
12:         detected_signs, norm_signs ← detectSignsFromVideo(video)
13:
14:     for sign, norm_sign in checkpoint.signs, norm_signs do
15:         if sign in detected_signs then
16:             damaged ← classifyDamge(norm_sign)
17:             if damaged then
18:                 sign.conformanceStatus ← "non-conforming"
19:
20:                 checkpoint_conforming ← False
21:
22:             else
23:                 sign.conformanceStatus ← "conforming"
24:         else
25:             sign.conformanceStatus ← "non-conforming"
26:
27:             checkpoint_conforming ← False
28:
29:     if checkpoint_conforming then
30:         checkpoint.conformanceStatus ← "conforming"
31:
32:         return True
33:     else
34:         checkpoint.conformanceStatus ← "non-conforming"
35:
36:         return False
```

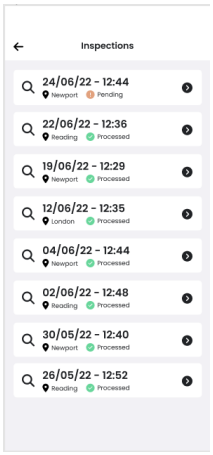
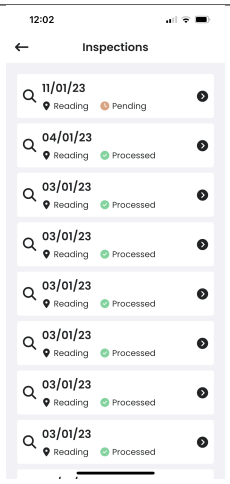
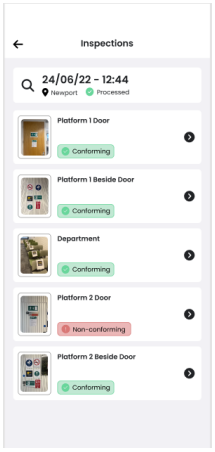
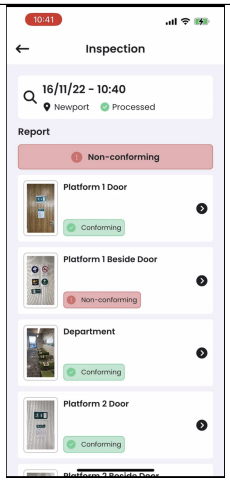
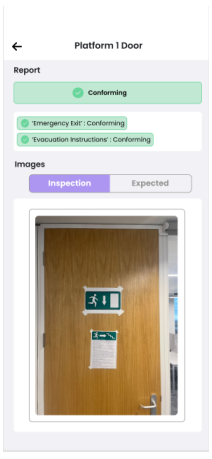
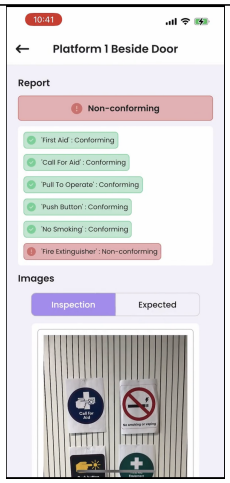
SECTION D

Testing

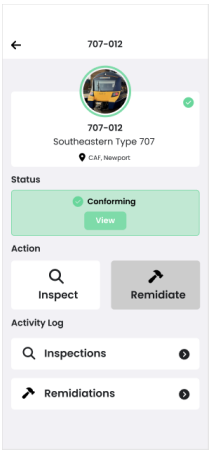
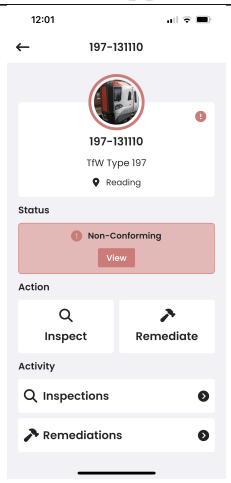
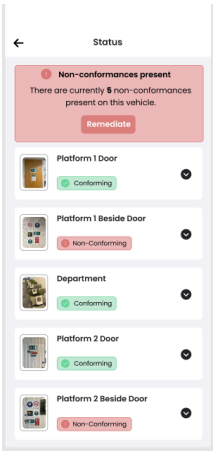
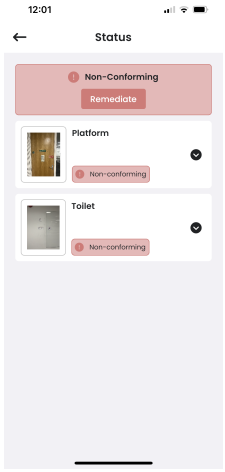
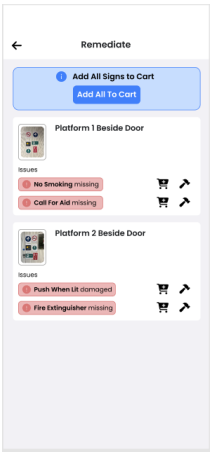
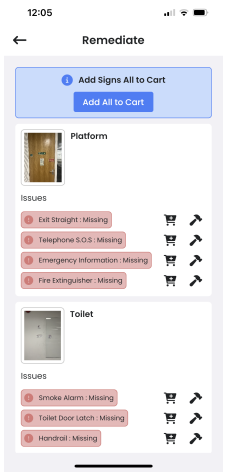
D.1 Deliverable 1 Storyboard Testing

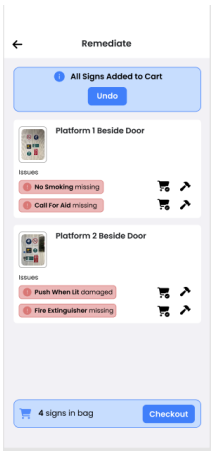
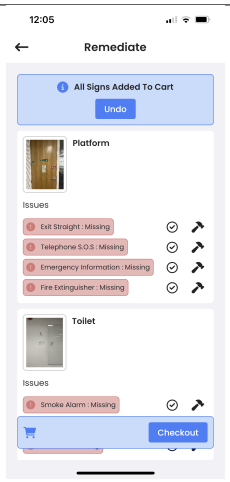
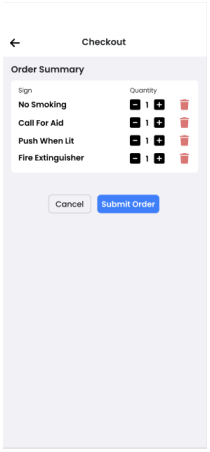
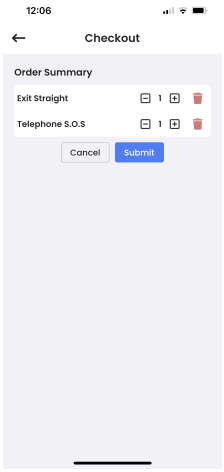
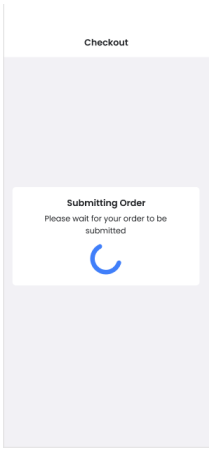
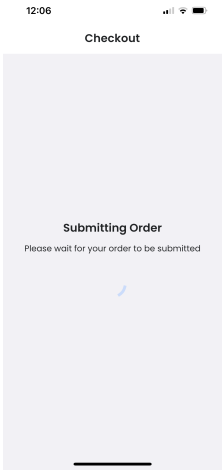
Planned Mock-up	Actual App Window	Notes
		No Notes
		Slight sizing changes
		Slight sizing changes

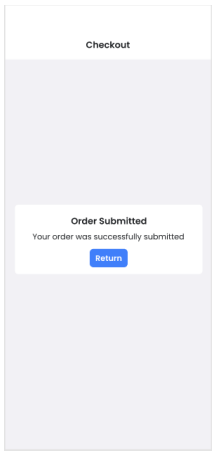
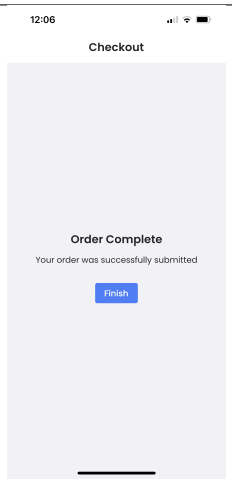
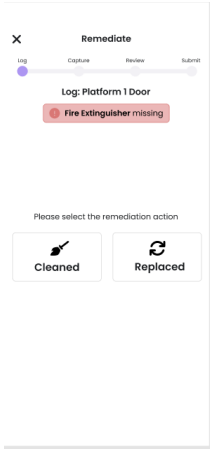
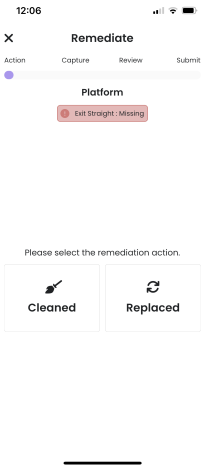
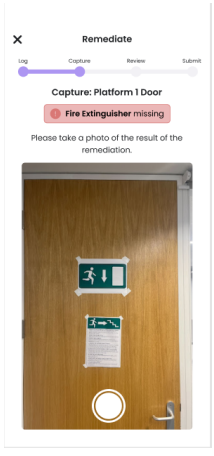
		Slight sizing changes
		Slight layout changes
		Word change

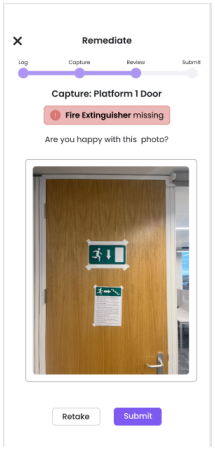
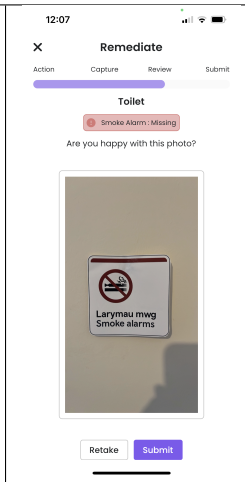
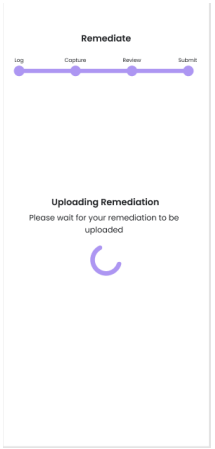
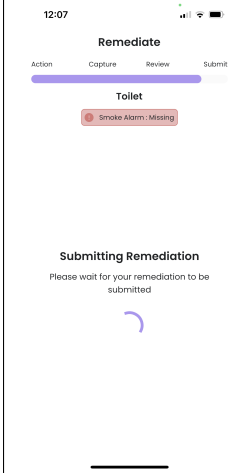
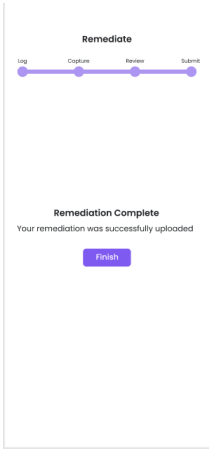
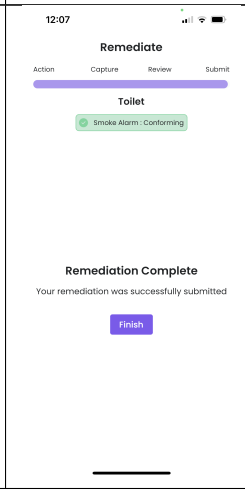
		Removed time from inspection tile and slight spacing changes.
		Additional bar added at the top to show user the conformance status of the whole inspection.
		Looks different because of different signs in the image.

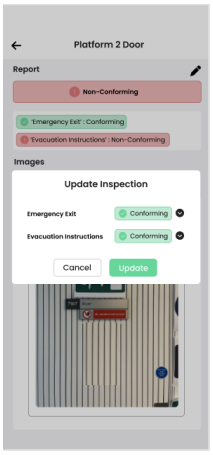
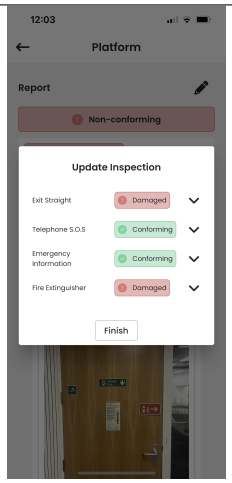
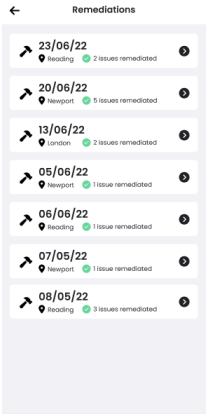
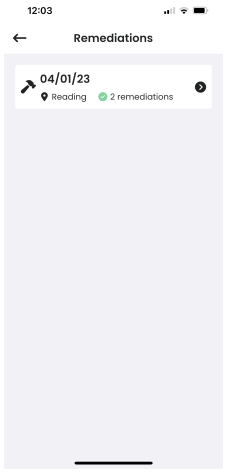
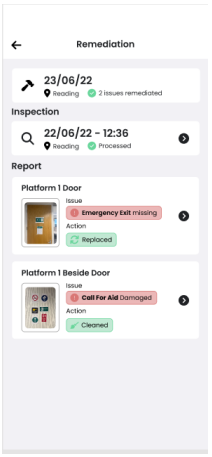
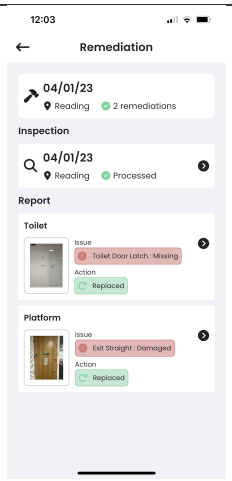
D.2 Deliverable 2: Storyboard Testing

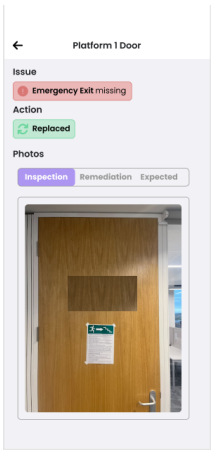
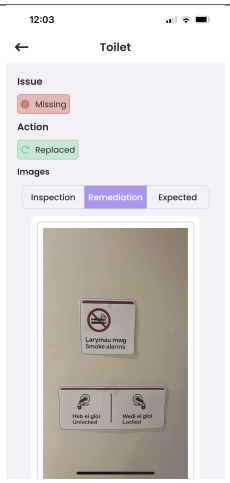
Planned Mock-up	Actual App Window	Notes
		No notes
		Slight simplification in design for convenience in development.
		No notes

		No notes
		Slight UI changes
		No notes

		Slight UI changes
		No notes
		Slight UI changes to remove unneccessary text

		No notes
		No notes
		Slight UI changes to show the updated conformance status now the remediation has been logged

		Slight UI changes for convenience in development
		No notes
		No notes

 Platform 1 Door Issue Emergency Exit missing Action Replaced Photos Inspection Remediation Expected	 12:03 Toilet Issue Missing Action Replaced Images Inspection Remediation Expected	No notes

SECTION E

Project Management

E.1 Project Gantt Chart

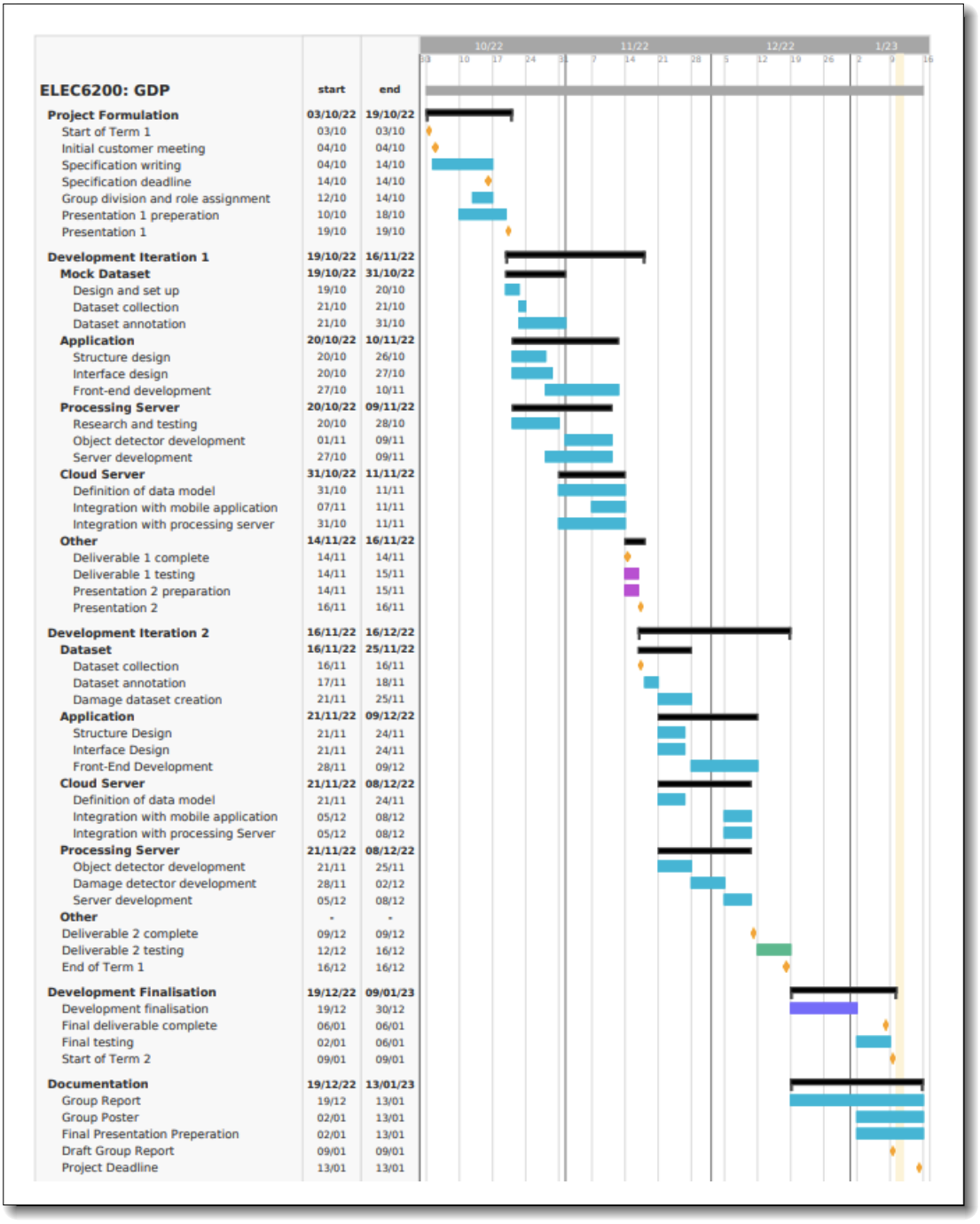


Fig. E.1. Project Gantt Chart