# Zbroya – Deep Learning Application for Firearms Logistics and Management

**DT228/TU856**
**BSc in Computer Science**

**Alec Keane**

**C19326126**

**Supervisor**

**Aneel Rahim**

School of Computer Science

Technological University, Dublin

**31/03/2023**

# Abstract

In this project, elements of Deep Learning have been combined will be used in the goal of identifying firearms on a live video feed. This will be performed on a Flutter Application on an Android phone. When weapons are identified the user will be able to add them to their database, forming an inventory over time. The goal of this being to aid the logistics and management of the Armed Forces of the Ukrainian Military. Through the creation of a more effective logging system, squadrons will have to spend less time on inventory management and be able to formulate requests for supplies easier.

# Declaration

I hereby declare that the work described in this dissertation is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

**Signed:**

Alec Keane

____________________________

**Student Name:**

Alec Keane

**Date:**

16/10/2022

## Acknowledgements

I would like to give a special thanks to everyone who has aided me throughout the development of this project. This includes all my friends who have given me advice along the way, along with my family who have always supported me and provided advice for my application.

I wish to give thanks as well to Aneel Rahim for guiding me through the steps of this project and giving advice on different stages of this project. I would also like to thank Dr. Susan McKeever for her advice and insight on Deep Learning and suggestions provided. I would also like to give final thanks to Jonathan McCarthy for all of his advice on my ideas and for approving of the idea, alongside his work on our final year module which has provided much insight for this work.

## Table of Contents

## Table of Figures

# 1. Introduction

## 1.1. Project Background

The war in Ukraine is one of, if not the largest war to happen in the 21st Century, being one of the first Peer to Peer conflicts of our time. The effect of this war has been felt in all corners of the world. The Ukrainian War is on track to be one of modern history's bloodiest conflicts and it shows few signs of slowing down. The purpose of this project will be to assist in logistical and inventory management for the Armed Forces of Ukraine.

Each conflict and sudden retreat on the battlefields in Ukraine result in masses of weapons being left behind. Whilst vehicles are sent to the backlines for repairs and repurposed. Squadrons of soldiers are left to manage rifles and other equipment that they acquire. As such inventory management and logistics for supplies for this equipment has become more and more important.

All the equipment that soldiers have recovered needs to be logged and kept track of. At times identifying the difference between firearms can be very difficult. Along with this, logging a large quantity of items can take time. This is the problem the project aims to resolve. Decreasing the time taken to log and increasing the accuracy of firearms identification.

## 1.2. Project Description

This project is an AI implementation on an Android Flutter application. The AI has been trained on a custom-made Dataset for this specific task. It will perform object detection. The Implementation for the AI Google AutoML pre trained cloud-based AI.

This application consists of a Camera screen for live scanning to detect Firearms that appear on the screen. Using a simple UI design, click on the highlighted object and add it to a database. The database for the storage of firearms is  a locally stored database using Hive and Secure Storage NoSQL databases to increase security by not storing data on the cloud where it could potentially, in a significant breach, be accessed. The choice of Local Storage means that the worst-case scenario is that a single Squad has its database exposed if the group phone is lost and access is gained into the app.

The functionality of the app will be as follows. A squad acquires many weapons and needs to identify and log each one. They use the AI trained by Google AutoML's cloud services, with each gun laid out. The phone's camera will allow the AI to detect and highlight each firearm it can see. Then the user will click on each highlighted weapon to add each one to the database. Here in the database, the Squad can view every weapon they have in their possession. This gives the squad the knowledge they need quickly to know what spare parts and ammunition they may need to request from their command. Squads are also able to view and control how much ammo they need for different firearms. Along with this they can assign specific weapons to specific soldiers to keep track of who has which firearm.

## 1.3. Project Aims and Objectives

Overall aim and some milestones along the way to achieve the aim.

- Flutter Application
- User Authentication
- Training and Deployment of Custom Trained Pre-Trained AI Model
- Two AI Model Layer, Pre-Trained Models for Object Detection and Classification
- Recognition of specific types of firearms, such as the AK-74
- Gathering, Cleaning and Preparation of a Firearms Dataset
- Potential recognition of Firearm Attachments
- Deployment
- Cross-Platform

## 1.4. Project Scope

This project is intended to be a cross-platform logistics and inventory management tool to save soldiers of the Armed Forces of Ukraine time and increase their efficiency. It is designed to assist in identifying and logging various firearms, a task which can be very helpful to newer soldiers who may not be capable of identifying each one yet. This project is focused on weapons only. It will not be used to identify vehicles, missiles, ammunition, or other military equipment.

Overall, the primary areas of development will be the applications Front End and Cloud Services

### i. Front End

The front end of this application will be written in Flutter. Flutter is an open-source UI software kit made by Google and based on Dart. The front end will consist of multiple sections.

**Login & Registration**

The app will first open on a screen where they will be prompted to sign into the account they created with the app or register. They will be able to create an account and register security questions if they ever forget their password. If the user is to create an account and forget the details they can answer the questions to get their password. However, on login and in that screen if the user fails three times in a row, for security reasons the data of Hive and Secure Storage will be deleted to prevent a breach of information.

**Inventory & Logistics**

Once logged in the suer will be met with the Squad inventory page. This is the first of four pages. Being Squad Inventory, General Inventory, Camera, and Logistics. On the Squad inventory page a user can see all of the weapons registered with specific soldiers and if they wish they can create a weapon and assign it to a soldier.

The general inventory page is very similar but here a user can view all of the weapons and their quantities and create new custom weapons. This is linked to the Logistics page which will take in the quantity of weapons, each one having a default magazine and round count for the gun. Using these it will calculate how many bullets are needed per gun when issued to a soldier, and how many bullets would be needed for every weapon to be deployed.

**Camera & Live Detection**

The third page will be the camera screen. Here the user is met with a scanning button that when pressed will paint detected firearms. Each firearm can be added to a list with a simple click of a button. A notification telling the user that they have been added. When the user stops scanning, if they have detected weapons, they will displayed them on a page to confirm them before they are added to the inventory.

**Database**

As for the database, Hive and Secure Storage are going to be used to locally store data on the phones themselves. Both being databases that can be secured with AES-256 bit encryption. Hive will not open or activate until a user has signed on to prevent attempted tampering and to keep all of the data unreachable. Secure Storage will be used to manage users and when Hive will be allowed to activate upon a registered user signing in. It will also be used to manage the state of the application, as if a user fails multiple times in a row, it will delete all of the User data in Hive and Secure Storage.

### ii. Cloud Services

For the deployment of our Deep Learning Model, we will be using Google AutoML's deployment features to upload locally a custom-made model and dataset in Tensorflow. Firebase was considered but for security purposes it has been cut out of the project as it has been decided best to store all military data locally on the phone in case of a breach of google servers. At worst case only one phone could be compromised in this method if the enemy retrieved it. Google AutoML will allow us to deploy a fully custom trained model to the Android itself in the App and scan for items using Google ML Kit Object Detection.

## 1.5. Thesis Roadmap

Chapter 2 will be discussing the research and development of this project, along with what papers and existing solutions exist in a similar domain to this project. Much of the research in this chapter will be used to develop upon the project and prototype. Chapter 3 will cover the methodologies used in the research and development of this project and how the task was approached. Chapter 4 covers the work that was created for the application and the development process behind it. Chapter 5 will cover the testing and evaluation of the application. Finally, Chapter 6 will discuss the issues encountered in the development of the project. It will also cover future risks and the projection of future work and what will be done for project.

# 2. Literature Review

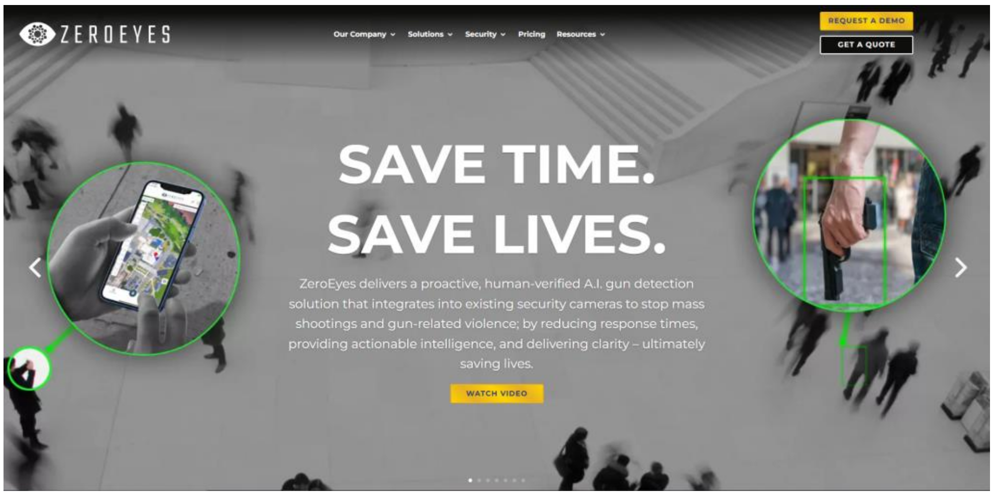
## 2.1. Introduction

This chapter covers all the background research conducted for this project. It also covers what existing or similar solutions already exist to the problem this project is looking to resolve. All technologies and considered solutions will also be detailed in this chapter.

## 2.2. Alternative Existing Solutions to Your Problem

### i. ZeroEyes

For this project, there is currently a handful of similar research papers. Alongside this, multiple products on the market perform a similar task of Gun detection. The primary market product in this domain is ZeroEyes. This is a Gun Detection program implemented onto security cameras to prevent mass shootings. How it does this is by recognising firearms on a live feed and alerting security staff to their presence so they may make the best choices they can to prevent an attack. Similarly, Viso.ai has produced an AI which can detect weapons in real-time video streams, like ZeroEyes(1).



ZeroEyes delivers a proactive, human-verified A.I. gun detection solution that integrates into existing security cameras to stop mass shootings and gun-related violence; by reducing response times, providing actionable intelligence, and delivering clarity – ultimately saving lives.

*Figure 1 ZeroEyes Website*

### ii. Recognizing Firearms – Tony Wang

As for similar research that has been done in this area, there are multiple projects and papers focusing on the area of detecting a firearm in real time. These papers' primary focus has been on recognizing handguns and identifying them as so. Such as a medium article by Tony Wang, titled "Recognizing Firearms from Images and Videos in Real-Time with Deep Learning and Computer Vision."(2) The purpose of this article is to compare different Deep Learning algorithms in recognizing firearms. It used many different models such as Mask R-CNN, Matterport TensorFlow, TensorFlow Object Detection API, Darknet YOLO and the Facebook Detectron. During their research, they chose to implement the

Darknet YOLO and Matterport Mask R-CNN models as real-world examples, both implemented through code rather than API calls. Both datasets used pre-trained COCO weights and images of firearms, roughly 50 rifle images and 100 handgun images. After testing, it was found that YOLO had a relatively high degree of accuracy and only took 0.1 seconds. Whilst Matterport RCNN had a much higher accuracy, it took roughly 1.5 seconds per image to recognise the weapon.

Overall, it was found that, on average, coded implementations had far more success than API this was due to the ability of manual tuning to improve performance. It was also found that YOLO was far superior to Mask R-CNN in terms of real-time performance. This was because it was multitudes faster and slightly less accurate since Mask R-CNN goes through thousands of iterations, whilst YOLO only does it once.

#### iii. Artificial Intelligence and Deep Learning for Weapon Identification in Security Systems

The most similar research that has been done compared to this project is a paper by Makkena Brahmaiah. “Artificial Intelligence and Deep Learning for Weapon Identification in Security Systems.”(3) This paper reached the accuracy and trained speed of R-CNN and SSD for the identification of multiple variants of firearms. This paper trained its algorithms on various firearms, the most like this project’s research is the AK47.

It was found during this paper that SSD took roughly 12 hours longer to train on the dataset than R-CNN, and it still performed almost 10.8% less accurately. The conclusions of this paper have once again shown that RCNN is not fit for real-world detection, however, despite being accurate it still performs too slowly. Even though SSD was 73.8% accurate, it would still be a better choice for real-time detection than R-CNN, which had 84.6% accuracy.

## 2.3. Technologies you’ve researched

#### i. Flutter, Kotlin or Java?

After much consideration, all options are feasible choices for this project. However, for this project, Flutter will be the language of choice; why is that?

**Java**

Java is a veteran language for Android development. It has been around for over two decades and was the standard for Android. It is a scalable object-oriented language that many developers already knew. This made it near the perfect language for Android development at the time of the launch of the Android OS in 2008.

This language has been around for years and has a large community built around it. It’s a dynamic, high-level, flexible language for developing applications for Android. So why wasn’t it chosen? In terms of performance and speed, it is relatively slow compared to the modern industry. Alongside this, the default GUI is very dated. Some packages can help with this, but in terms of default GUI, it is outmatched by many other mobile development languages. One other reason for the decision not to use Java is the complexity of the code. Java is a very verbose language. On average, Java code can be cut down by almost 20% by Kotlin(4), only to be also outmatched on performance and the general feel of the app. To provide a modern and sleek app with a codebase that will be easy to maintain and update, Java has not been chosen for this project. However, what about Kotlin?

### Kotlin

Kotlin is the language that has taken over Java's place as the default language of Android(5). Kotlin, on average is around 20% less code than Java(4). Being a naturally less verbose language not only means that your code will be shorter and more readable, making it less likely to have bugs(6). Being the new standard, Kotlin is gaining more support and a larger community than Java, meaning many more recent packages and libraries related to the project's topic may be developed for it. It is a reliable language that performs faster than Java and can be written more efficiently. But why wasn't it chosen?

### Flutter

Whilst Kotlin is by no means an incorrect decision for this project. The choice is still going to be Flutter. Why is that, exactly? Flutter, developed by Google, has become a popular language for app development since its creation in 2017(7). Soaring in popularity due to its ability to quickly establish beautiful multi-platform and multi-operating system apps. Every year it has gained more popularity and expanded its libraries more and more each month. This means dozens of different libraries and packages in different languages can be employed for this app. Being widely versatile, heavily supported and regularly updated make this the perfect choice for the project's goals, as with its quick design time for a simple, sleek UI this will provide more time for training and building an AI to deploy on the application which is the most essential part of the application.

## ii. Libraries for Deep Learning Models

Some of the most popular libraries for Deep Learning are Tensorflow, Keras, and PyTorch. These libraries have seen extensive usage and upgrades in recent years. Many algorithms and pre-trained models have been developed based on these. In research for this project, these three have been selected as they are popular and well-supported upon Flutter.

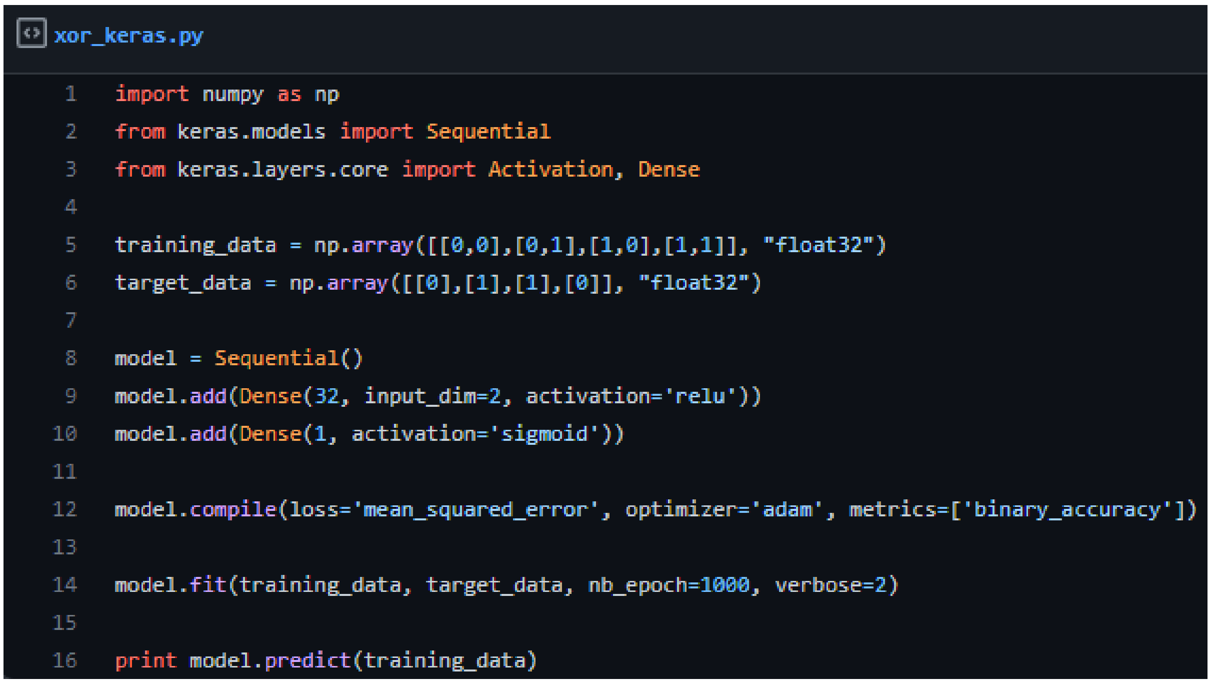
#### PyTorch

PyTorch is a library that is based on Python and the Torch library(8). It is used for Deep Learning and Machine Learning, like Tensorflow and Keras. However, one feature that makes many people prefer it over the other two libraries is that it uses Dynamic Computation Graphs. This means that code can be tested to determine whether it works without requiring a complete implementation. These allow for increased development speed due to the ease of debugging. PyTorch also comes built-in with three pre-trained object detection models(9). Faster R-CNN with a ResNet50 backbone or instead with a MobileNetV3 backbone, and finally, a RetinaNet model with a ResNet50 backbone. This makes it a good candidate for building a mobile application quickly with a pretrained AI that can be trained further with a custom dataset.

**Keras**

Keras is a library that was made in Python(10). It is one of the best choices for fast experimentation for a Deep Learning project due to its ability to define a complex neural net in a few lines. Whilst often paired with TensorFlow it can still operate independently of Tensorflow and vice versa. Despite this, the most efficient method of using Keras in Flutter is combining it with TensorFlow.

Code example of an XOR algorithm in Keras vs Tensorflow

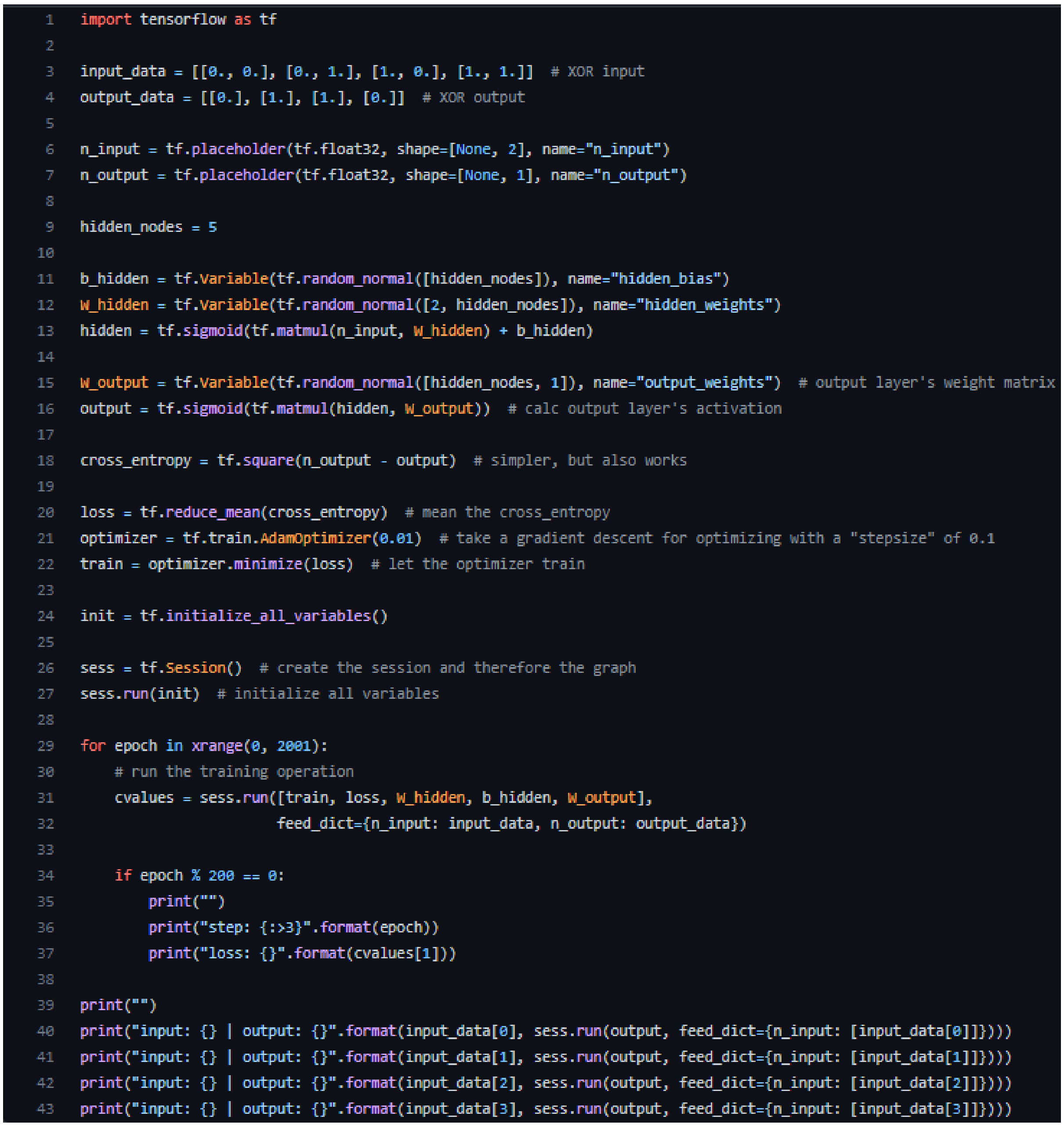


xor_keras.py

```python
import numpy as np
from keras.models import Sequential
from keras.layers.core import Activation, Dense

training_data = np.array([[0,0],[0,1],[1,0],[1,1]], "float32")
target_data = np.array([[0],[1],[1],[0]], "float32")

model = Sequential()
model.add(Dense(32, input_dim=2, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='mean_squared_error', optimizer='adam', metrics=['binary_accuracy'])

model.fit(training_data, target_data, nb_epoch=1000, verbose=2)

print model.predict(training_data)
```

*Figure 2 Keras XOR Algorithm*

https://gist.github.com/cburgdorf/e2fb46e5ad61ed7b9a29029c5cc30134



```python
import tensorflow as tf

input_data = [[0., 0.], [0., 1.], [1., 0.], [1., 1.]]  # XOR input
output_data = [[0.], [1.], [1.], [0.]]  # XOR output

n_input = tf.placeholder(tf.float32, shape=[None, 2], name="n_input")
n_output = tf.placeholder(tf.float32, shape=[None, 1], name="n_output")

hidden_nodes = 5

b_hidden = tf.Variable(tf.random_normal([hidden_nodes]), name="hidden_bias")
W_hidden = tf.Variable(tf.random_normal([2, hidden_nodes]), name="hidden_weights")
hidden = tf.sigmoid(tf.matmul(n_input, W_hidden) + b_hidden)

W_output = tf.Variable(tf.random_normal([hidden_nodes, 1]), name="output_weights")  # output layer's weight matrix
output = tf.sigmoid(tf.matmul(hidden, W_output))  # calc output layer's activation

cross_entropy = tf.square(n_output - output)  # simpler, but also works

loss = tf.reduce_mean(cross_entropy)  # mean the cross_entropy
optimizer = tf.train.AdamOptimizer(0.01)  # take a gradient descent for optimizing with a "stepsize" of 0.1
train = optimizer.minimize(loss)  # let the optimizer train

init = tf.initialize_all_variables()

sess = tf.Session()  # create the session and therefore the graph
sess.run(init)  # initialize all variables

for epoch in xrange(0, 2001):
    # run the training operation
    cvalues = sess.run([train, loss, W_hidden, b_hidden, W_output],
                       feed_dict={n_input: input_data, n_output: output_data})

    if epoch % 200 == 0:
        print("")
        print("step: {:>3}".format(epoch))
        print("loss: {}".format(cvalues[1]))

print("")
print("input: {} | output: {}".format(input_data[0], sess.run(output, feed_dict={n_input: [input_data[0]]})))
print("input: {} | output: {}".format(input_data[1], sess.run(output, feed_dict={n_input: [input_data[1]]})))
print("input: {} | output: {}".format(input_data[2], sess.run(output, feed_dict={n_input: [input_data[2]]})))
print("input: {} | output: {}".format(input_data[3], sess.run(output, feed_dict={n_input: [input_data[3]]})))
```

*Figure 3 TensorFlow XOR Algorithm*

https://gist.github.com/cburgdorf/e2fb46e5ad61ed7b9a29029c5cc30134

**TensorFlow**

Our final potential choice is the TensorFlow library. Supported by Firebase and developed by Google(11), TensorFlow can be applied in an extensive range of languages such as Python, JavaScript, C++ and more. Tensorflow being supported has led Firebase to develop a online deployment service in the cloud for developers to deploy Deep Learning algorithms onto their projects without needing to deploy locally. This will aid in increasing the projects overall efficiency and usability from users as all the complex functionalities will be offloaded to the cloud. Alongside this TensorFlow Lite is also supported on Firebase(12). This means Keras models can be created and deployed on the cloud as well for the application.

**Conclusion**

Overall, it has been decided that none of the three libraires will truly be used. all three choices of libraries are viable and correct choices in themselves. However, it has been decided that Google AutoML is the best choice of method for a custom model for object detection. Being easy to learn and incredibly powerful with the ability for a user to upload thousands of images and leave the AI to train itself on the data. This reduces the chances of complications in the project's development, increases the speed of development, and keeps the technology stack Google oriented.

### iii. Model for Deep Learning

Besides the choice of Language, Framework, Library, there remains one very important decision. The choice of which model will be used in this project. There are many existing models and even more variations of each model. But what matters is which model will perform the best in the task of Live Object Detection. This comes down to a multitude of factors, speed, accuracy, deployment method, implementation difficulty. Along with this the most optimal models will be using a form CNN (Convolutional Neural Networks). This is because when scanning objects in an environment, features need to be extracted and mapped, and CNN is suited for this task. Many models have a backbone which is a pretrained network. Many of these models in our use case utilize CNN algorithms(13). This is because CNN is a neural network that is used in object recognition as it learns patterns from datasets to recognize and classify objects in images.

To create these algorithms in an exportable format for use in a flutter application Google Collab could be used. This is a free cloud-based tool that allows you to write and train algorithms in the style of a Jupyter Notebook on Google Cloud servers(14). The drawback to this is it requires deep knowledge of above-mentioned libraries and languages such as TensorFlow, Keras, Etc. This opens your model up to having errors if less experienced in this. but allows for far greater specification of models and their purpose.

Another choice considered for this projects purpose is Google ML Kit Library and AutoML Model. Both are part of the Google Stack and as such are heavily documented, tested, and will fit into a flutter application with few complications. ML Kit is a library that can be implemented as a package in Flutter(15), it contains a multitude of models but the one for this project will be object detection. AutoML is another model, and this would be implemented as a classification model training on a custom dataset.

They are veteran products that have seen extensive use and training, and AutoML is able to be trained on Google Cloud's Vertex AI platform(16). Meaning that training will be faster and more efficient than if it were to be trained on a PC not made for Deep Learning. Google ML Kit Object Detection is a model made for Object Detection, able to extract objects from images. It is also capable of using a custom classifier which is where AutoML will be deployed to classify extracted objects.

Below we will discuss the different types of suitable models for the project.

**YOLO (You Only Look Once)**

YOLO is one of the more popular models for Real-Time Object Detection. It's well known for its high speed and a decent level of accuracy for a live object detection model, which are often known to have difficulty with accuracy—scoring a Mean Average Precision of 57.9% on COCO test-dev(17).

The task of Object Detection is being considered for the purpose of this project due to its speed and level of accuracy. However, the final choice of model will only be decided upon after testing has been performed on other models as well. YOLOv3, YOLO-PP, YOLOv4, YOLOv5, and more will also be considered for this task. All are forms of YOLO; however, they utilise different backbones to achieve the same goal.

**Mask R-CNN**

Mask R-CNN, an extension of faster R-CNN is a form of Region-Based Convolutional Neural Network model(18). It's primarily used in object detection and is known to be relatively fast. However, it is less frequently used in real-time object detection due to it being known to be slightly slower than other models like YOLO and SSD, which were purpose-built for real-time detection(18).

It is slower compared to other object detection models because it takes in far fewer frames per second and runs many more iterations over the images it receives to determine what it believes it is seeing using feature extraction. This leads to a more significant increase in mAP (Mean Average Precision), but it comes at the cost of speed.

**SSD (Single Shot Detection)**

SSD is another form of Model that is being considered for this project. SSD is often considered the most optimal model regarding speed, performance, and resources. SSD works by breaking down images into grids and extracting features from the image through that(19). Whilst processing the image, it will also affect the zoom and aspect ratio to detect objects of different sizes. This is also up for consideration due to its relatively high speed and accuracy.

**Google AutoML & Google ML Kit**

Google ML Kit is one of the most popular machine-learning packages for Flutter(15). Being updated regularly unlike the TensorFlow Lite package which hasn't been updated in over a year. This package includes many different models for usage but the one for this project is Object Detection. This implements a powerful Object Detection model with a basic classifier easily on the Camera of a Flutter Application. Being used by so many means that the process of implementation is heavily documented, with an example app to show and guide new developers on how to implement it.

Using the ML Kit for our object detection it allows us to implement a custom Classification Model of our choice to classify extracted objects from images. Here we could use many different models, MobileNet, and EfficientNet, however, our choice will be AutoML. AutoML is a powerful Model on Google Cloud. It is thoroughly documented, easily implemented, allows for training on custom datasets, and is often used in cooperation with ML Kit as both are part of the Google Stack.

Alongside this 300$ is provided on Google Cloud for testing and training it. Being on the cloud as well means faster training as it will be trained on Google's servers and technology. This is a better choice instead of a single computer not made for Machine Learning. Making our process of training and implementation more efficient.

**Conclusion**

In conclusion, Google ML Kit and Vertex AI is the choice of Deep Learning algorithms for this project. This is due to the fact they were made for the google stack, meaning they are easily implemented with Flutter. As well as this Google Collab requires a proficient level of knowledge in TensorFlow and other languages which would have taken a large amount of time and possibly introduce complications and errors into the project. Meanwhile Vertex AI is a codeless platform where data can be uploaded, and models can be easily created and managed more efficiently.

As well ML Kit is chosen as it is well-documented, powerful and one of the most popular Deep Learning choices for Flutter applications. And Vertex AI not only provides 300$ for training purposes. But it is also cloud-based which means more efficient training, and it offers the service of training models for the specific purpose of being deployed off the cloud and locally onto an edge device such as an Android. The specialization and efficiency of ML Kit combined with Vertex AI and their customizability to this project's specific purpose make them the most optimal choice for this project.

### iii. Database Framework

For this project's application, there was need for a local database that can store the data retrieved from scanning our weapons, such as how many weapons there are and which variants of weapons. In the future, more may be added, but in terms of priority, these are a must. The database must be local and having it on the cloud proves a security risk and poses logistical issues with the lack of Network availability in parts of Ukraine, which means that soldiers on the front line may be unable to access their database in an area with no signal.  Due to this, Hive, a NoSQL plugin for Flutter was chosen as the database for this application.

Hive was chosen because NoSQL allows for a flexible data model(20), meaning that we are not limited to what has been declared in the schema. This allows changes to be made more accessible and future updates to include other equipment will be far less complex. Hive also takes pride in its encryption. Stating it as one of the main points of the plugin, all your data will be stored behind AES-256-bit encryption. Meaning your data is secure even if your phone falls into someone else's hands. Along with this, Hive is also one of the most popular database plugins for flutter, with a large community and a large amount of documentation.

Alongside Hive a second database was chosen specifically for security concerns. A database called "Secure Storage." This is used in place of preferred preferences and allows users to persist data. The reason this was chosen is to manage the users entirely. So from the front end Hive which contains a user's firearms will not open and expose itself to a potential attacker who may aim to abuse an unforeseen loophole to access the data within. In this method only secure storage will be available and it will require passing checks with Secure Storage before Hive is allowed to activate and open a box for a user to use.

### iv. Dataset Creation

In terms of finding a dataset for this project it was a difficult task. After searching for a long period of time only one dataset could be identified during the research phase. This dataset was made by INTERPOL(21) and contained thousands of firearms. However, to access this you had to be a member of a verified policing authority, this meant I could not access it. During the research as well, a NATO(22) paper was found which also outlined the difficulty in finding such a dataset.

Due to this the only option was to create a dataset for this projects purpose. This meant web scraping for thousands of images, cleaning these images, and then preparing them with labels for training on the Vertex AI Platform by hand. The current dataset that will be used for training is just a prototype and contains 2,645 images of specific firearms that were cleaned and labelled by hand. There are 30 individual labels in this dataset such as "AK-74" and "Malyuk." All images being sourced through web scraping google images using Python.

Gathering weapon data after the Interim report period became more difficult as Google put in place new web scraping restrictions which took down the libraries being used to gather data. As such not as much data was gathered after the interim. However, a dataset of regular items was gathered of over

1000+ images. This has been implemented on the AI and will be known as the “Non-Firearm” Data. This will aid the AI in identifying what is and is not a firearm.

#### V. Conclusion

In conclusion, for this project, a Flutter application has been determined as the most optimal choice of mobile application Framework for this use case. The Hive and Secure Storage plugins for a NoSQL non-relational database will also be used. As for the AI, Google AutoML model has been used and trained on a custom dataset and deployed locally due to security and network concerns. This however removes use of Firebase. Which means Firebase Test Lab cannot test the application’s performance and find bugs. As such manual testing will be required.

The choice of Flutter overall means most technologies in the Google Stack can now be implemented and taken advantage of to their fullest. Meaning there’s a vast amount of technology and documentation available for usages such as cloud storage and google authentication.

## 2.4. Other Research you’ve done

Research in the domain of Identifying firearms is very restricted. Even in terms of datasets, there are none that have been made for the case of identifying specific firearms. The primary dataset and use case regarding AI and Firearms are the detection of them for CCTV cameras for the purpose of security and prevention of active shooters.

Below is a breakdown of two papers which aided the development of this project.

#### i. A Gun Detection Dataset and Searching for Embedded Device Solutions

This paper(23) was made around the basis of helping prevent gun violence which is a rising issue across the world. The paper focuses on the idea of deploying a gun detection algorithm on cameras to perform processing and detection of footage for real-time detection of firearms. This paper is the paper that provided the idea of using Models such as YOLO and SSD in this project. During the research period of the paper, they compare many different forms of CNN-based Models to find which is the most accurate and efficient for users. This paper’s primary work also involves identifying false positives which encouraged finding negative data datasets for this project as well.

#### ii. Firearm Detection in Social Media

This paper(22) made by two researchers from the Swedish Defence Research Agency also provided valuable insights into firearm detection. This paper’s goal is to search across social media for individuals who flaunt their weapons online. To complete this, they needed to create an AI object detection model capable of recognizing firearms. This report proved valuable as they state that they were unable to find any sufficient forms of datasets online which inspired the creation of a custom one from online resources and similar datasets that did contain some data worth usage. It also provided useful details on the training and creation of a custom model, as it repeatedly mentioned the importance of pretraining models on data that is not what the final iteration of the model is meant to be looking for. This is because it improves the accuracy of your model if it knows of other things and decreases false positives.

## 2.5. Existing Final Year Projects

### i. Development of an Automatic Detection Solution for Solar Panel use in the State of Ireland Utilising Machine Learning and Image Recognition Techniques

The focus of this project was on the area of renewable energy, specifically in aiding the deployment of solar panels. The way it intended to do this was through using Deep Learning, specifically the Faster R-CNN model pre-trained on the COCO dataset. The plan was to use spatial data to assess different locations such as large plots of fields across Ireland to determine whether they were fit for the deployment of Solar Panels. The system was deployed and tested on a plot of land that was 0.34 km$^2$.

It was tested using aerial imagery from 170m elevation from the ground level in the local terrain using Google Earth Pro. It was found that in the live test there was an overall recall rate of 47.62% and a precision rate of 60.6%. Whilst these are good results for this test, the overall mAP was found to only be 29.89% which is rather low. Despite this, it was shown this can be accounted for by changing the pixel size. It was also shown if it were to be deployed and used on appropriate hardware the tool could gather data on a larger scale.

### i. Euro Coin Classification Using Image Processing & Machine Learning.

The purpose of this project was to classify euro coins in images through the usage of machine learning and image processing techniques. The primary machine learning techniques used in this were k-NN (K Nearest Neighbours) and Naïve Bayes. Both are two very common and simple, yet effective machine learning algorithms. During testing, it was found that Naïve Bayes was more accurate. They combined these machine learning techniques with the use of image processing, performing techniques such as Edge Detection and Transformations to assist the algorithms in their goal of classifying each coin. This paper is primarily focussed on research that did not have any true deployment and did not grant many insights into this project's work; however, it did lend the idea of augmenting datasets with machine learning techniques to potentially improve models' accuracy.

## 2.6. Conclusions

Throughout the research of this project examples were found which perform similar tasks or serve a similar purpose to this project. The research of these articles, papers and projects have provided great insight. It has been decided that the Google Stack will be primarily used in this project. The choices include, Flutter, Hive, Secure Storage, Vertex AI Platform, AutoML, and Google ML Kit. The reason for this is that these are robust products with detailed documentation that will have few if any compatibility issues with a Flutter application. As well as this Vertex AI platform has been chosen over Google Collab. This is because writing custom models in TensorFlow could implement unforeseen issues and will consume a far larger amount of time and testing. Comparatively Vertex AI has all the functionality needed ready and specific training set up for single-label classification and deployment on phones.

Finally, a custom-made dataset has also be used as there are no existing datasets that access can be gained to that have the data in the format that needed it. This dataset has been constructed through the usage of Python to web scrape. This means a large amount of real weapon images are going to be used to train the AI but also through synthetic data from video games. Some firearms found while web scraping were from games. Whilst cleaning some that use realistically modelled weapons were chosen to not be removed to help the AI and expand the dataset. Synthetic data is a common choice for projects now as many games have very realistic models. For example, Escape From Tarkov(24) specifically could be used to gather additional data on weapons. Providing additional context and angles for the models to train on.

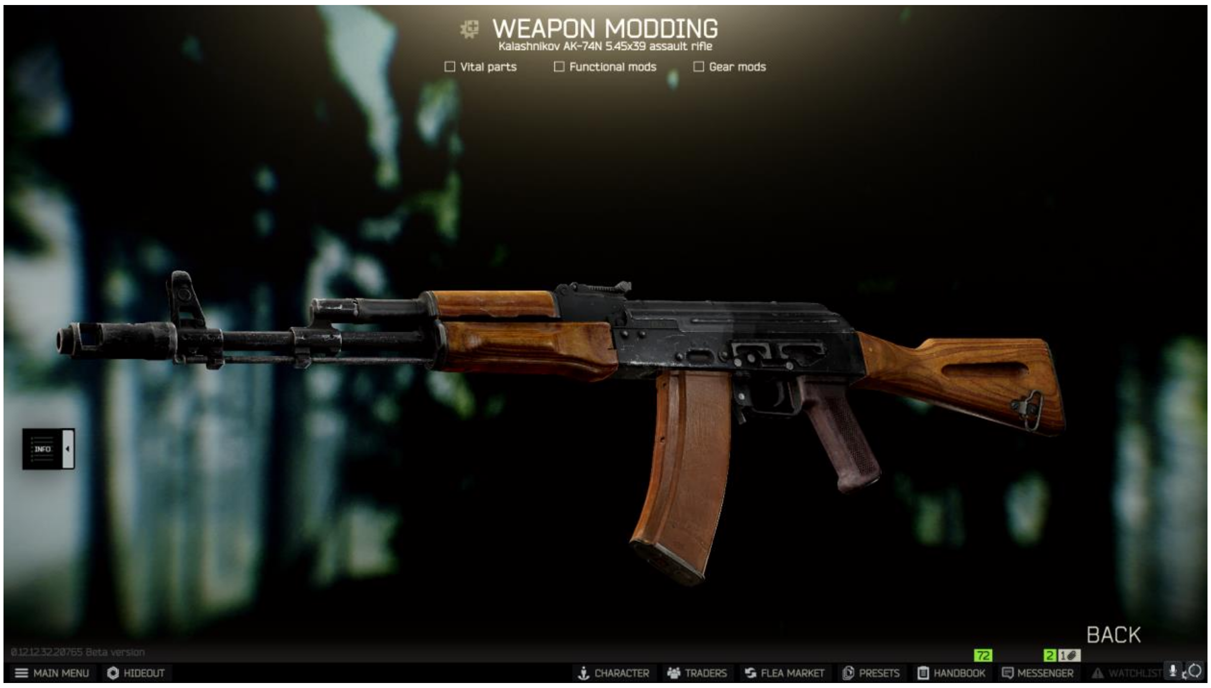


*Figure 4 AK-74N From Escape From Tarkov*

https://www.escapefromtarkov.com/

As for what was findings were made from researching papers.

- This paper(23) was very insightful for what Models might be good to use in detection, alongside this they also provided an open-source dataset of 51k gun images that they used to train their own system. This dataset could prove useful in sourcing images in the future for the dataset in creation for this project. It also proved useful in knowledge of how others have approached the task of detecting weapons.
- The insights that were gained from this paper(22) overall were the process of how to train a model, and the fact that a custom dataset was likely needed for this project's own use case as it seeks to go further than just detect but to identify. It also provided the idea of using an Object Detection API from Tensorflow, however, in research it was found this will often be too inaccurate and operate too slowly for the goals of the project to be met.

# 3. Application Design

## 3.1. Introduction

This chapter will cover the methodologies employed in this project and the descriptions of the software and architecture used within the development of the prototype of this project. These will be explained through the usage of descriptions and diagrams.

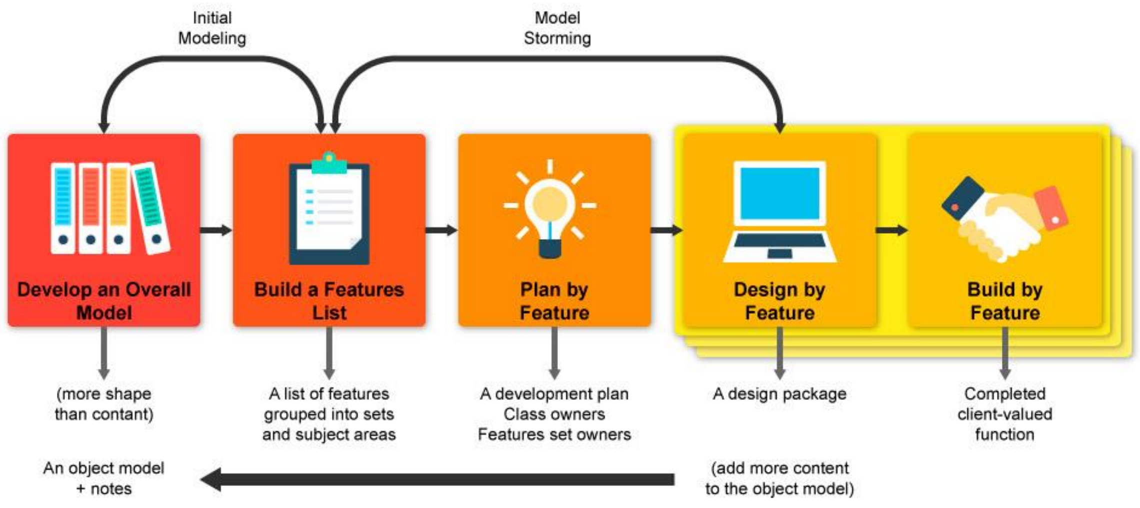
## 3.2. Software Methodology

The purpose of using a software methodology is to create a structure of how you will develop a system. It involves setting requirements and setting a plan of how you will achieve these requirements within a certain time frame. They are used to produce an effective cycle of production that covers planning, coding, testing, and iteration to deliver on the assigned projects goals.

For this specific project Agile approaches were primarily considered. This is as they are suitable to small teams which makes them easy to scale down. As well they focus on iteration and flexibility to create a quality product. Feature Driven Development, Extreme Programming, and Kanban are all examples of agile based methodologies.

### i. Feature Driven Development (FDD).

FDD is a methodology with a focus on iteration and working constantly towards the goal of delivering software that meets the customers needs(25). It is broken down into five phases. Developing an overall rough view of the system, constructing a list of the features required, planning based on the required features, designing by feature, and building by feature. Often used in teams this process breaks apart a software into its components. Tasking the team to focus on each piece in order of priority. Tasking the team to work on each feature with the domain experts in charge of design solutions. Breaking down large tasks into components in this manner allows for safer and more efficient development of large-scale solutions and allowing for easier debugging.



Initial Modeling
Model Storming
Develop an Overall Model
Build a Features List
Plan by Feature
Design by Feature
Build by Feature
(more shape than contant)
A list of features grouped into sets and subject areas
A development plan Class owners Features set owners
A design package
Completed client-valued function
An object model + notes
(add more content to the object model)

*Figure 5 FDD Diagram*

https://lvivity.com/7-things-about-feature-driven-development

### ii. Extreme Programming. (XP)

XP is another agile methodology for small teams that need to focus on dynamically changing requirements for the software in development(26). It has a large focus on customer feedback which is why the requirements must be dynamic as they may change often. This methodology takes a heavy emphasis on programming, recommending pair programming for bug finding and problem solving. This methodology also takes strong approach on testing, requiring a large amount of unit testing to ensure quality is maintained with the speed of development. The downside is the heavy reliance on a customer to give feedback regularly and it's focus on coding over designing can lead to unforeseen issues which is why pair programming is recommended.

### iii. Kanban

Kanban is a methodology like scrum, in that teams will visualize their work to maximize their efficiency(26). Setting up a board to split work between team members and to track the progress of each piece under development by visualizing the work to be done. This optimizes work and ensures that each thing every member of the team takes on is a meaningful piece of work that will aid the final goal. It is a methodology known and liked for its flexibility and requirement of continuously delivering on work in each sprint which is split into multiple week periods. It is a highly efficient system in most cases, however, due to its nature if one piece has its progress disrupted on the board the whole team needs to slow down or readjust to fix the issue(27).

### iv. Scientific Method

As this is a Deep Learning project a certain level of research is required. The research focus will be upon the models used in this project. This project aims to use two layers of models to accomplish the goal of detecting and identifying weapons. One model for Object Detection, this model has been selected as Google ML Kits Object Detection package for Flutter. And one model for classification which will be AutoML.

To accomplish the task of researching and developing a scientific method was considered a good decision to use. In this case the CRISP-DM lifecycle has been chosen to be used in a hybrid form with another agile method. This method consists of six phases(28). Business Understanding, what we require for our project. Data understanding, what data we need for the project's goals. Data preparation, the gathering and cleaning of data. Modelling, the decision of what modelling strategy to use. Evaluation, which model best suits the business objectives, and finally deployment(29).

### v. Conclusion

It was decided that for this project a combination of a scaled down version of Feature Driven Development and CRISP-DM will be used as the methodology. For this project a rough outline of the current conception has been created. Along with this a list of the features that are believed to be needed. An overall list of features decided upon during the requirements and design process. Alongside this CRISP-DM's structure has been followed for the data collection, preparation, and training process. Using web scraping through Python and training via Vertex AI's Platform for this. Finally for the application a portion of the overall features were chosen to be developed. In specific the list of features for the final application are detailed below.

| **Login UI** | **Camera** | **Hive/Secure Storage** | **Squad/General Inventory** | **Logistics page** | **Pin Page** | **Translations** | **PDF Export** |
|---|---|---|---|---|---|---|---|
| Login Page | Object Detector | User Details handling | Adding custom weapons functionality | Display ammunition requirements | Force user Pin login | | |
| Sign Up Page | Painting | Store detected weapons | Assigning Soldiers weapons | Allow firearm editing | | | |
| Forgot Password Page | Scanning | Custom weapons class | | | | | |
| | Confirmation | | | | | | |

(Red marks future work.)

## 3.3. Overview of System

This application has been developed for the Android series of phones. Using Hive and Secure Storage for database management with a NoSQL database. Firebase Authentication was going to be used for login if there's internet, however, this was cut out in the final product. This was done to security concerns and the fact soldiers may not have access to internet at all times. As such Hive and Secure Storage will be used to manage a phones user locally. Dart will be used for managing methods and data in the back end. The Google AutoML model and Flutter for the front end. The former system architecture can be seen below in the appendix, displayed here is the current system architecture.

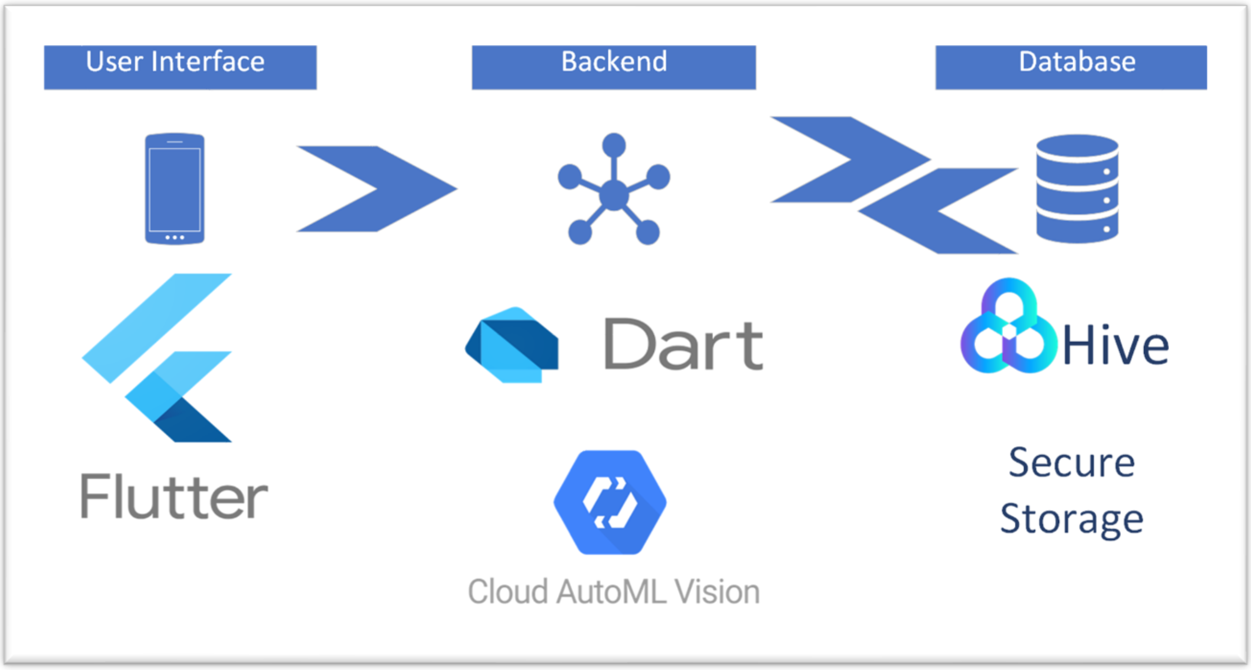


*Figure 6 System Overview*

Now in this application users will create accounts and sign in using Secure Storage. This will manage all users and if they are cleared through it will allow Hive to turn on and open a box for a user to use to scan and add weapons. Past this when the user logs in they will be faced with the Squad Inventory view. This page was chosen over the camera screen to open on for performance reasons. On the page a user can see all of the information of who in the squad uses what weapon. They can also add additional weapons if they chose to. On the second page they will find General Inventory which is very similar, here can will see all of the weapons in storage and be able to add more. As well as this they can edit every weapon and change details about them. This page is closely linked with the Logistics page, which displays all of the weapons and how many rounds per gun will be needed. Along with the total ammunition requirements of each batch of firearms and how many magazines each weapon needs. This provides information for the user regarding logistics when requesting ammunition resupplies. Finally, they will be met with the camera page. Here they can begin scanning which will paint detected weapons in a red box. If they are then added using a button the user will be notified. When the user stops scanning if they have scanned weapons a page will display the weapons and ask the user to confirm the weapons they see. If they do they will added to Hive and displayed on the General Inventory page.

Also of note, in the event of a phone being captured by an unauthorized person, the users account was originally planned to be linked firebase and an admin could delete it. Now however a PIN is demanded on each login if the user is signed in. If they fail this the user is signed out and brought to the login page. Here they can request a password back or try to login. If the user fails either of these three times however, all of their data will be deleted to prevent an unauthorized user breaching the data.

The former case diagram can be seen below in the appendix, here is the current applications case diagram.

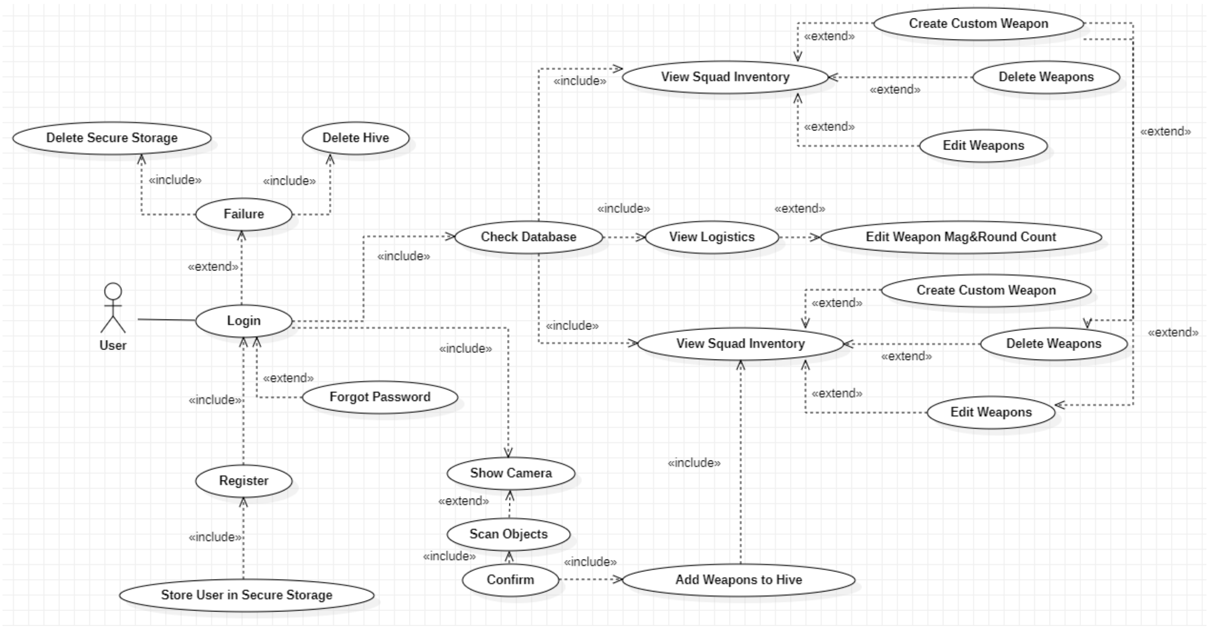


*Figure 7 System Use Case Diagram*

To add onto this with additional context a sequence diagram is also provided below. Detailing the process of the users' actions along with the back-end checks and showing how situations are handled. It also displays in a clear manner how the models interact with each other to find objects and classify them. Then from there displaying a modal to the user of what it found for them to add them to the database. Provided is the current system diagram of the current application. The Previous version is in the Appendix below.

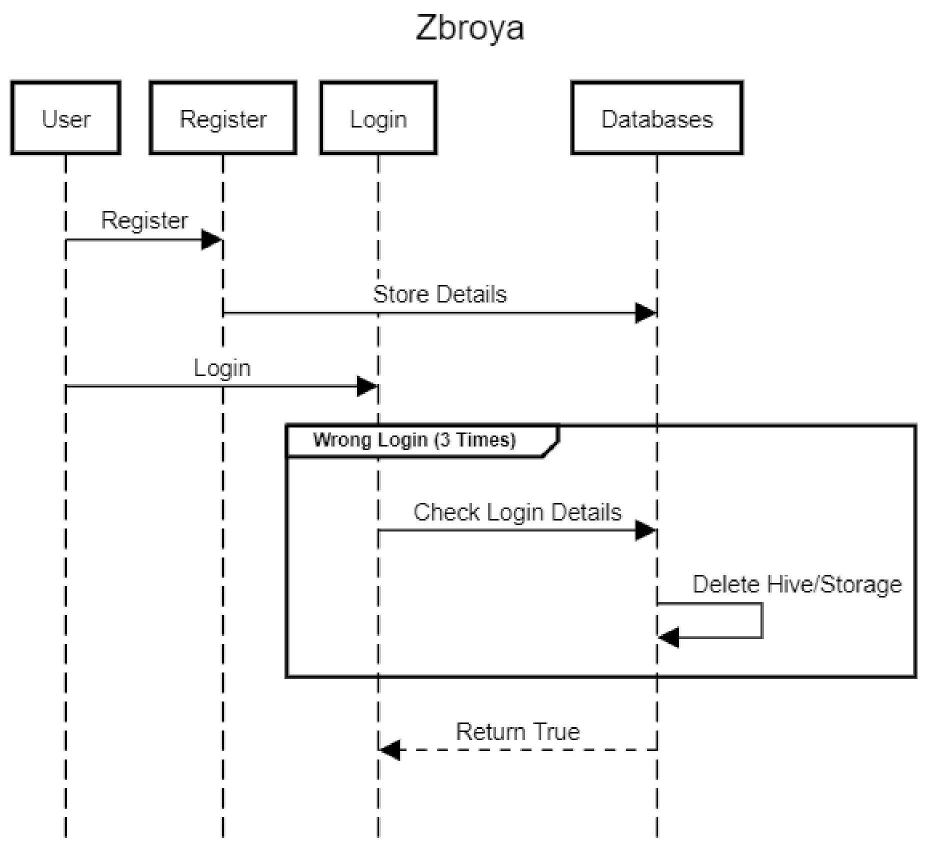


*Figure 8 Login Sequence Diagram*

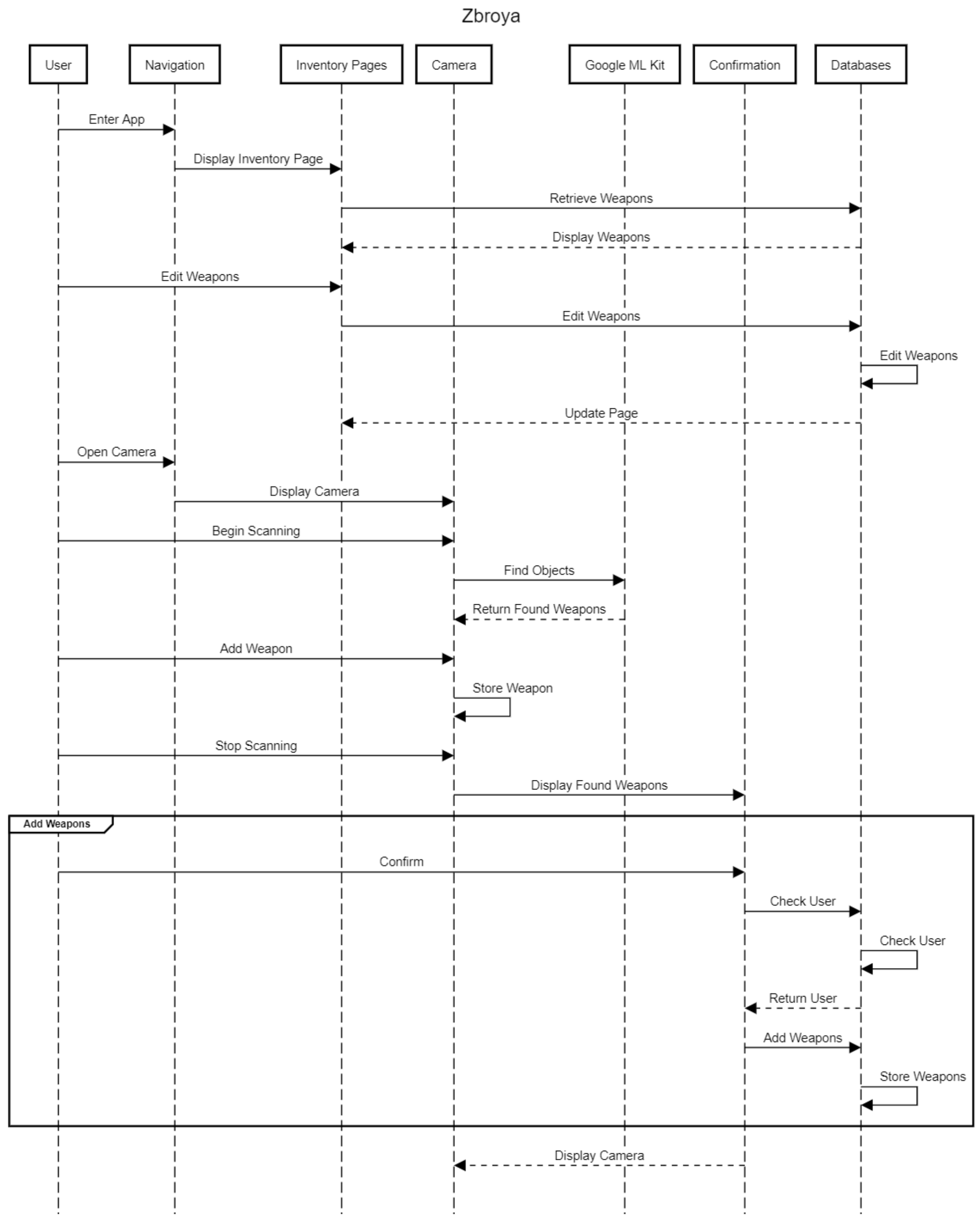


*Figure 9 User Inventory and Scanning Sequence Diagram*

Below is a class diagram. This diagram is used to display the relationships between different objects in the system and the methods that they provide. Class diagrams are useful in giving a full overview of the system and how it will function as you know what functions come from where. It is useful as a guide when implanting systems. Below is the finished applications class diagram. In the appendix is the old version.

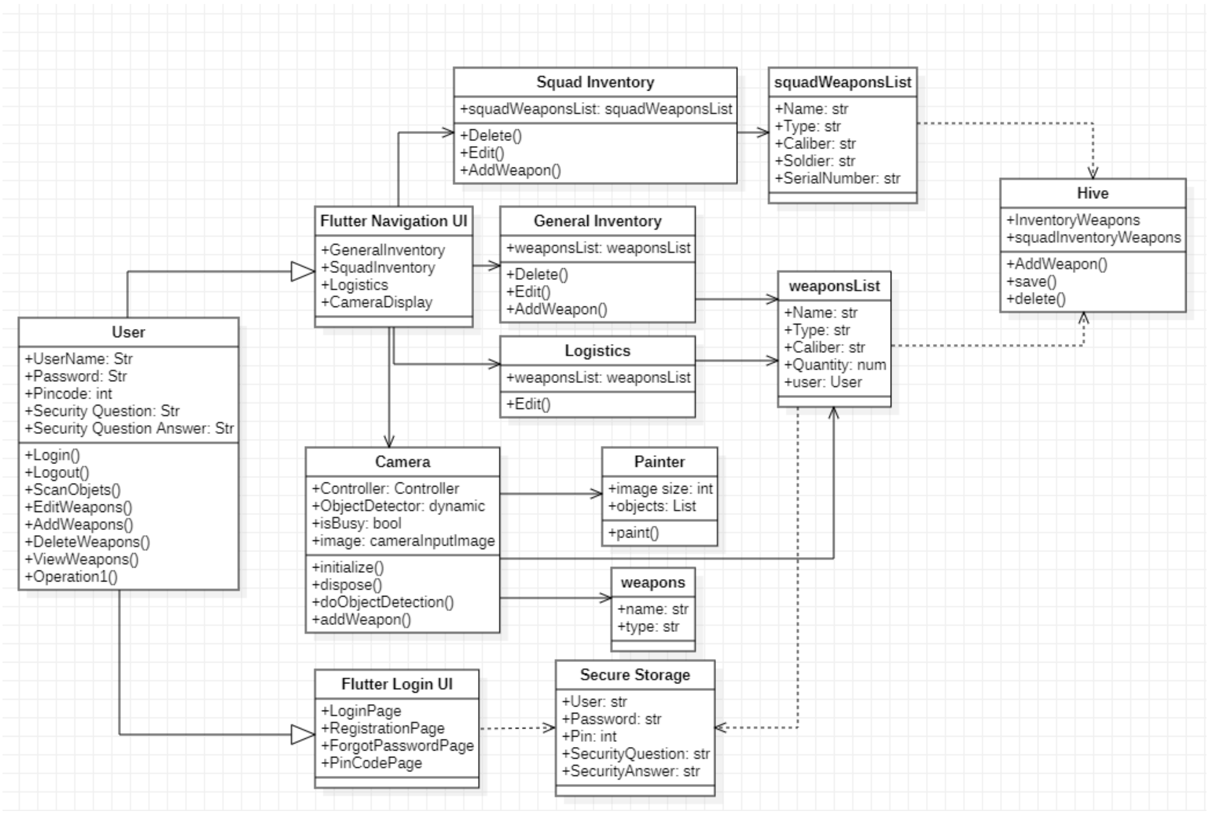


*Figure 10 System Class Diagram*

Finally, we have the database diagrams, these are useful for developers when developing and implementing databases. For this projects purpose a NoSQL database via the Hive package has been decided as the best decision due to the need for data to be dynamic. Due to previous misconceptions, there is only have two tables in the prototype database shown in the appendix. There is the user table and the weapon table. The below diagram is the latest applications database diagram structure which is more advanced as there is now two databases and two different weapons stored in the database.

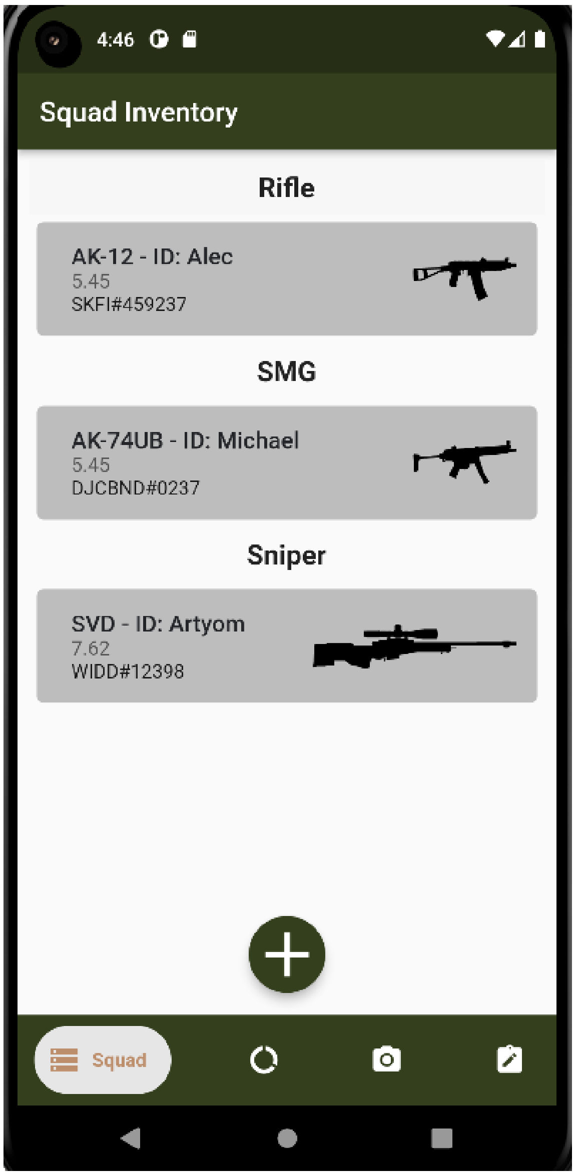
| User(Secure Storage) | weaponsList(Hive) | squadWeaponsList(Hive) |
|---|---|---|
| {<br>“Username”: String,<br>“Password”: String,<br>“Pin”: int,<br>“Security Question”: String,<br>“Security Answer”: String<br>} | {“weapon”:{<br>“Name”: String,<br>“Type”: String,<br>“Caliber”: String,<br>“Quantity”: num,<br>“User”: User(),<br>“RoundCount”: int,<br>“MagCount”: int<br>},} | {“weapons”:{<br>“Name”: String,<br>“Type”: String,<br>“Caliber”: String,<br>“Soldier”: String,<br>“SerialNumber”: String<br>},} |

*Figure 11 Database Diagram Structure*

## 3.4. Prototype Mock-ups Vs Application

In the appendix are a series of images of the mockups which were made while the prototype was in development. At this time the only page that existed on the application was the camera page. Since beginning implementation design changes have occurred for a multitude of reasons. Ranging from logic and states needing to passed easier, to certain widgets only working in a certain manner, and purely aesthetic choices to make the application look better.

Below are the finished images of the final application running on an emulator, to show how the pages have changed from start to finish of development and the stylistic and layout changes that have been made, along with the new pages that have been created such as  the page for logistics. In the appendix you will find the old mock-ups.


4:46

Squad Inventory

Rifle

AK-12 - ID: Alec
5.45
SKFI#459237

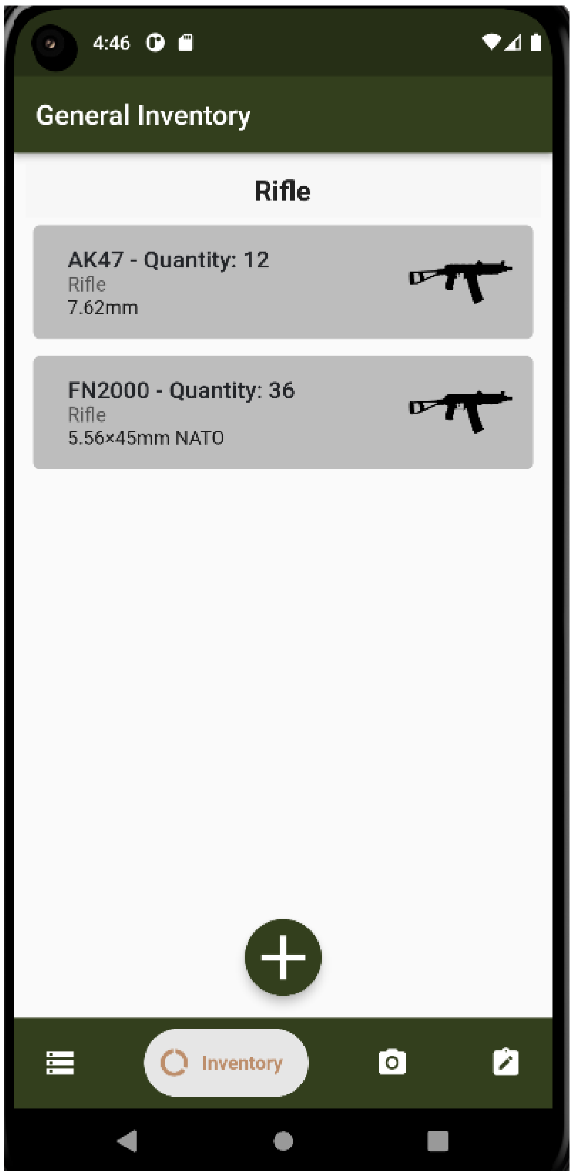
SMG

AK-74UB - ID: Michael
5.45
DJCBND#0237

Sniper

SVD - ID: Artyom
7.62
WIDD#12398

*Figure 13 Squad Inventory*


4:46

General Inventory

Rifle

AK47 - Quantity: 12
Rifle
7.62mm

FN2000 - Quantity: 36
Rifle
5.56×45mm NATO

*Figure 12 General Inventory*

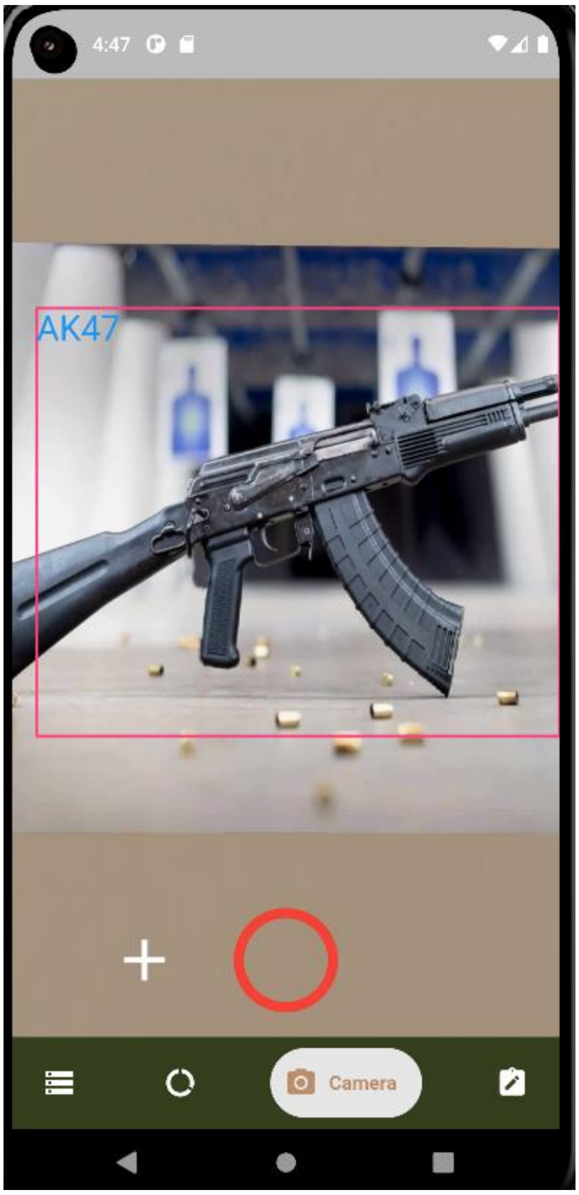
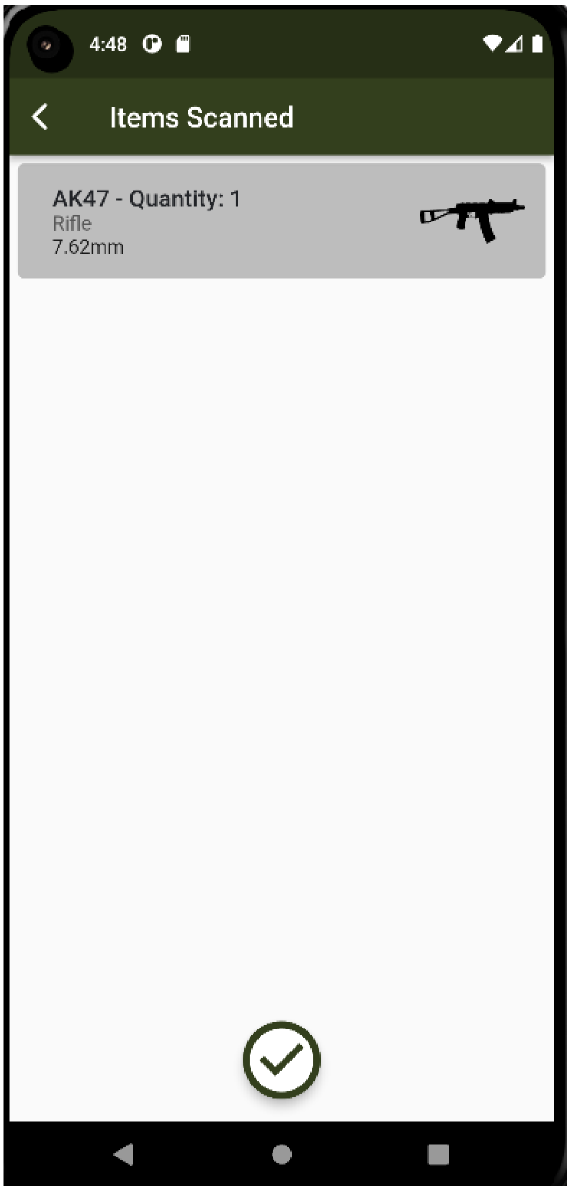
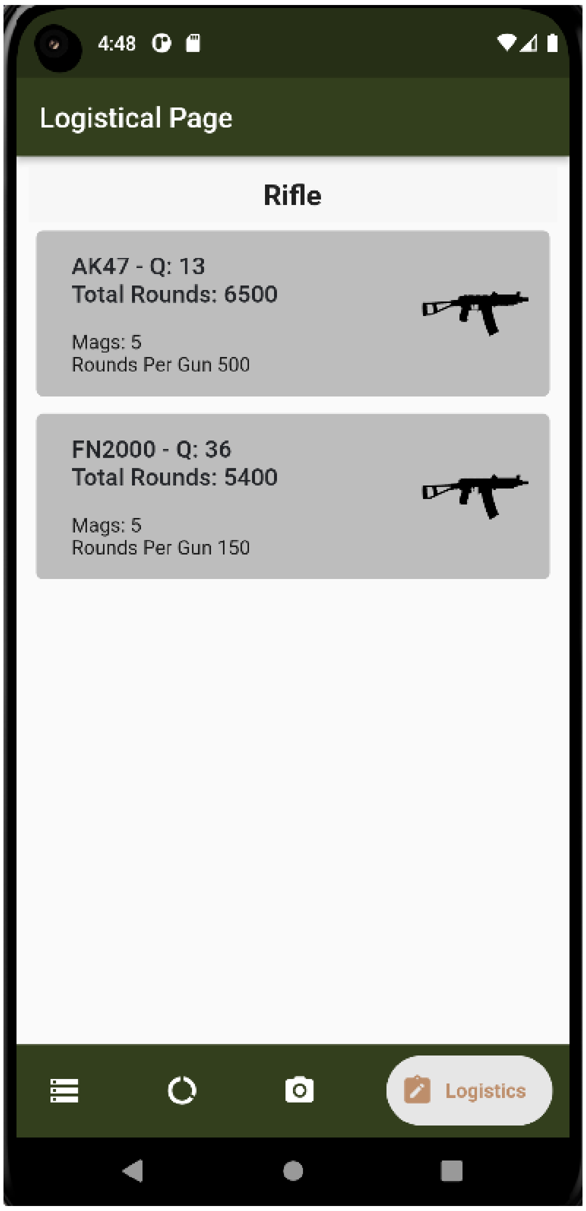


*Figure 14 Camera Scanning*



AK47 - Quantity: 1
Rifle
7.62mm

*Figure 15 Confirmation of Scanned Weapons*

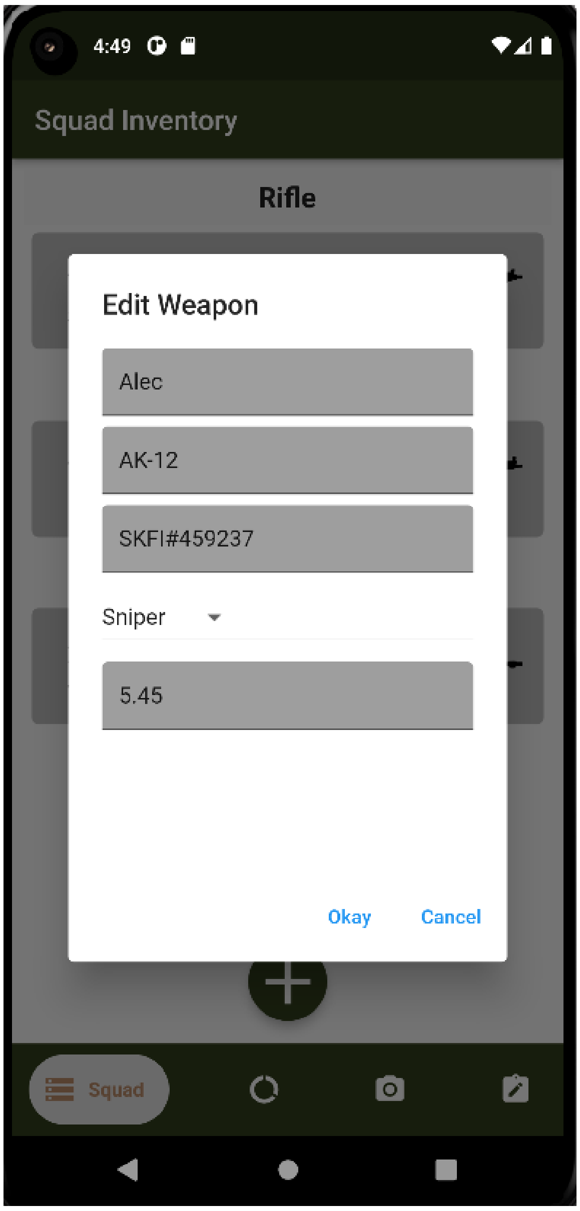


Logistical Page

Rifle

AK47 - Q: 13
Total Rounds: 6500
Mags: 5
Rounds Per Gun 500

FN2000 - Q: 36
Total Rounds: 5400
Mags: 5
Rounds Per Gun 150



Squad Inventory

Rifle

Edit Weapon

Alec

AK-12

SKFI#459237

Sniper

5.45

*Figure 16 Logistics Page*

*Figure 17 Edit Weapon Modal*

## 3.5 Conclusions

In conclusion, the chosen methodology has been Feature Driven Development combined with CRISP-DM. These methodologies have helped drive along progress of this project by aiming the work needed for the project towards the most vital and important pieces first before others, and allowing for swift upgrading of the Google AutoML model. These methodologies combined with the rough inspiration given by the UI mock-ups have helped propel this application from a prototype to an almost fully finished product.

# 4. Application Development

## 4.1. Introduction

This chapter will discuss the process of development of the final application for this report. Breaking down the development process into the different sections of the front-end to the back-end and how the AI model was created for the system.

## 4.2. Overview

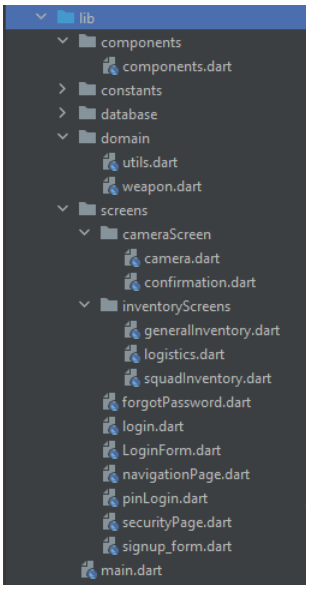
The application developed for this project is a Flutter Cross-Platform application with a locally installed custom AI Model for Object Detection and another model for object-detection and extraction called Google ML Kit.

This application is entirely run locally on the phone and is offline. This was a hard requirement due to security and network concerns, due to the nature of the information that will be stored in the database and the locations of which this application will be employed. Being in a dangerous environment where the phone could be compromised and likely on a battlefield meant that this had to be a secure application that could also be used where no internet connection was available.

Being Flutter the applications front-end code and business logic code are heavily linked together in the front-end. They can be separated into what is called BLOC architecture, however, this would have taken too much time to be completed by the project deadline. As such it was attempted to the best degree possible to keep a somewhat clean architecture with some forms of code separation, for increasing readability. The front and back-end code will be discussed under the codebase section. Along with this the application uses two databases. One for users and the other protected by the first will store that users data. Finally there is a local AI deployed known as Google ML Kit which is performing object detection and extraction, with a second Model as another object detector for classification custom made form Google Vertex AI platform, known as AutoML.

### 4.3.1 Codebase-Login

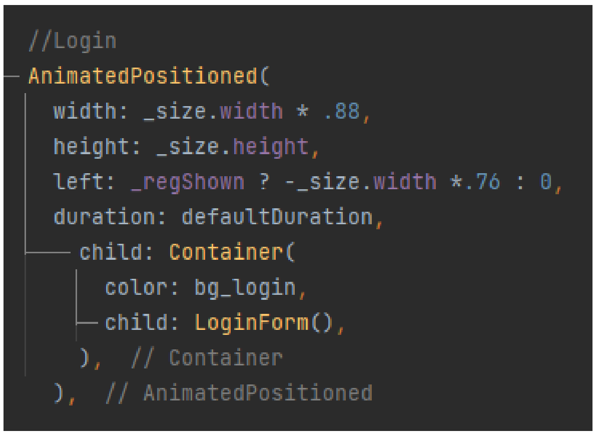
The codebase of the application has been written in Flutter. As such the front-end and back-end are heavily coupled together in the standard fashion of flutter when not using BLOC Architecture. This application is comprised of approximately 13 separate screens and is entirely local. The application being local made certain features such as sign up and login more complex. Most applications use firebase but this application cannot. As such additional form validations and security features had to implemented due to the application being entirely offline. Meaning all validation, security checks and functions and registration had to be performed locally. This section will detail specifically how the Login functionality of this application was handled.



```
lib
  components
    components.dart
  constants
  database
  domain
    utils.dart
    weapon.dart
  screens
    cameraScreen
      camera.dart
      confirmation.dart
    inventoryScreens
      generalInventory.dart
      logistics.dart
      squadInventory.dart
    forgotPassword.dart
    login.dart
    LoginForm.dart
    navigationPage.dart
    pinLogin.dart
    securityPage.dart
    signup_form.dart
  main.dart
```

*Figure 18 Application Structure*

Two of the most important screens are Login.dart and NaviagtionPage.dart. These pages handle the two separate halves of the app. The login.dart file contains the animation login and registration pages as animatedPostions with children designated to the signup_form.dart file and the LoginForm.dart files. Both being displayed on the page allows for the unique login animations to be handled when navigating between the pages.

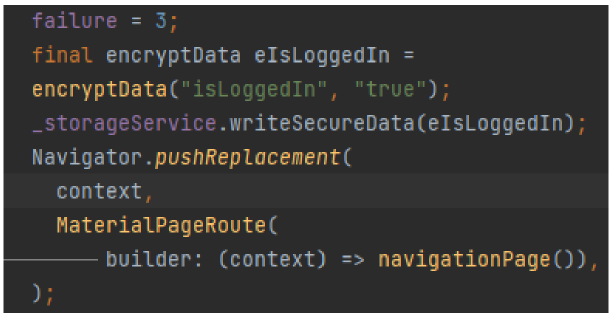


```
//Login
AnimatedPositioned(
  width: _size.width * .88,
  height: _size.height,
  left: _regShown ? -_size.width *.76 : 0,
  duration: defaultDuration,
  child: Container(
    color: bg_login,
    child: LoginForm(),
  ),  // Container
),  // AnimatedPositioned
```

*Figure 19 Animated Position for Login Animation*

These pages employ the SecureStorage database to handle registering a user to the database and then allowing that user to sign in, which will then open Hive “boxes” to allow for the user to begin to add and manipulate data in Hive. Hive is only opened when a user logs in as a security feature to protect it and decouple it entirely from the login screens until a user has signed into the application.

The login section also includes the “isLoggedIn” variable which will be set to true when a user logs into the application. This will allow a user to persist in the application as logged in and instead of prompting a login from them everytime they open the app, it will ask for a PIN code instead to allow the user to login. As well as all of this, if a new account is created in an attempt to overwrite details, for safety reasons the whole Secure Storage and Hive databases are deleted. This is because this application is only meant to have a single user.



```
failure = 3;
final encryptData eIsLoggedIn =
encryptData("isLoggedIn", "true");
_storageService.writeSecureData(eIsLoggedIn);
Navigator.pushReplacement(
  context,
  MaterialPageRoute(
    builder: (context) => navigationPage()),
);
```

*Figure 20 Code for Pushing a new Page*

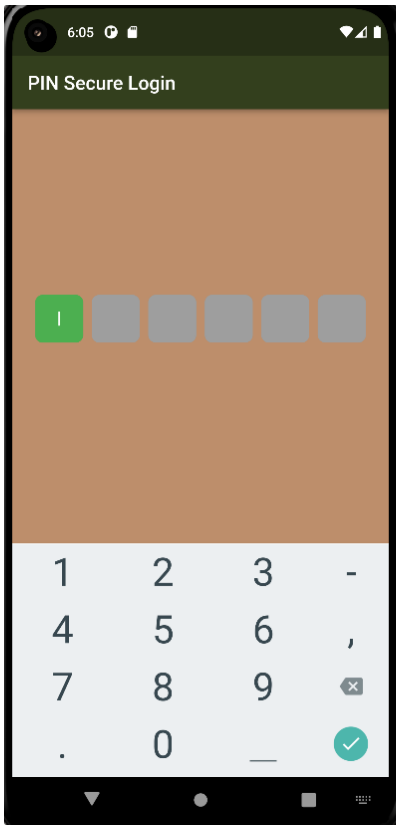
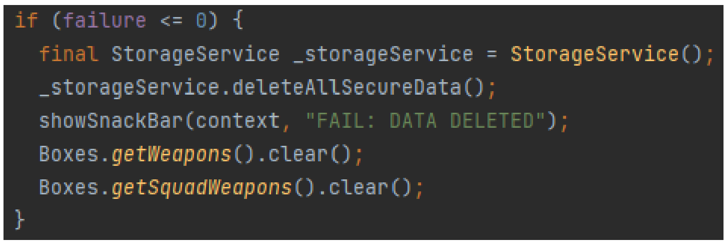


*Figure 21 PIN Login Page*

As well as handling the login, there is additional security failsafe employed on the application to prevent any unauthorized users from handling the data contained on the application. If the user fails their PIN three times in a row they are logged out, however, if the user also fails to login 3 separate times or to answer the security questions that they set then all the data in both databases will be purged to protect it from being breached. This was done as the primary use case of this security would be if the phone was already lost and unretrievable in which case the data needs to be wiped before data can be stolen.



```
if (failure <= 0) {
  final StorageService _storageService = StorageService();
  _storageService.deleteAllSecureData();
  showSnackBar(context, "FAIL: DATA DELETED");
  Boxes.getWeapons().clear();
  Boxes.getSquadWeapons().clear();
}
```

*Figure 22 Delete Database Code*

## 4.3.2 Codebase-Inventory

After having completed registration and login. The user will be brought to the Squad Inventory page, they are brought here instead of the camera page for performance reasons. As activating a camera controller can be intense on older phones.

On this page they will be met with the weapons that are in use by members of their squad and which members are actively using each weapon, along with the serial number of the weapon in the event it is ever lost. They will also have a floating action button which will allow them to add a new custom weapon themselves.

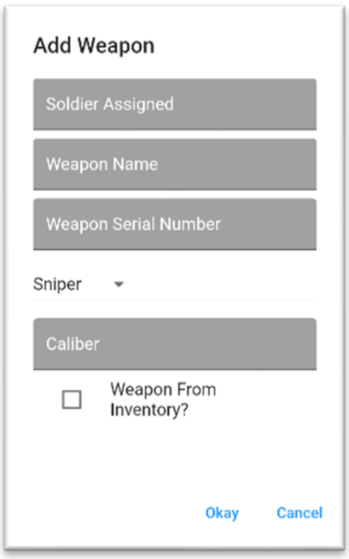
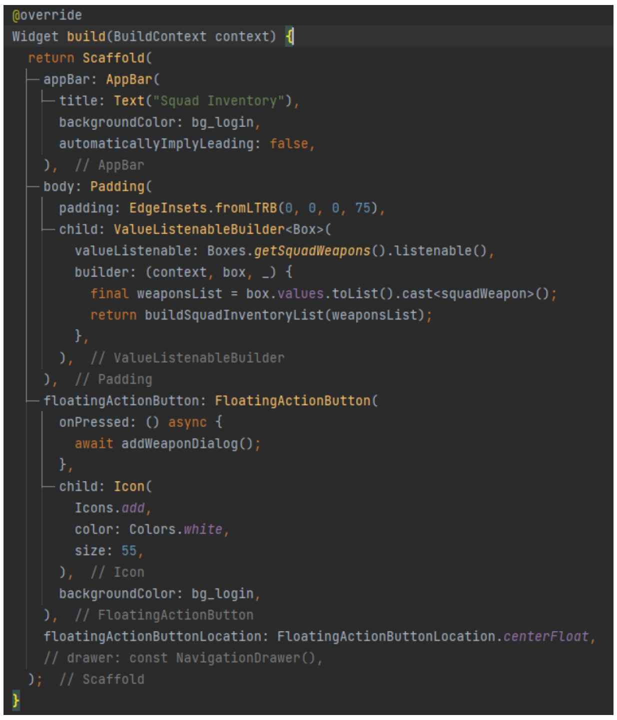


*Figure 23 Add Weapon Modal*

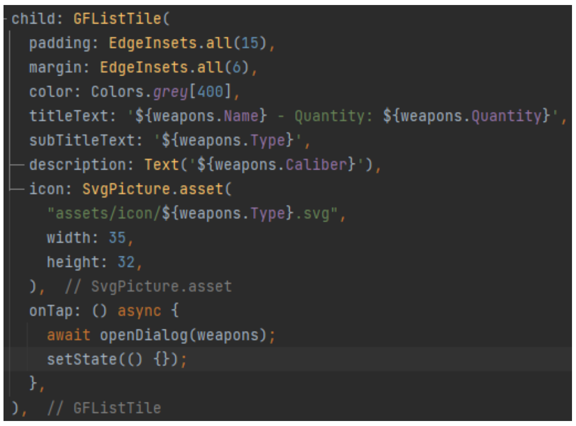
This page along with all of the other inventory screens will be using GFListTiles to display the data found on the page, contained inside of a scaffold. This will populate tiles going down the page for each weapon that has been added to the Hive Database.



```dart
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text("Squad Inventory"),
      backgroundColor: bg_login,
      automaticallyImplyLeading: false,
    ),  // AppBar
    body: Padding(
      padding: EdgeInsets.fromLTRB(0, 0, 0, 75),
      child: ValueListenableBuilder<Box>(
        valueListenable: Boxes.getSquadWeapons().listenable(),
        builder: (context, box, _) {
          final weaponsList = box.values.toList().cast<squadWeapon>();
          return buildSquadInventoryList(weaponsList);
        },
      ),  // ValueListenableBuilder
    ),  // Padding
    floatingActionButton: FloatingActionButton(
      onPressed: () async {
        await addWeaponDialog();
      },
      child: Icon(
        Icons.add,
        color: Colors.white,
        size: 55,
      ),  // Icon
      backgroundColor: bg_login,
    ),  // FloatingActionButton
    floatingActionButtonLocation: FloatingActionButtonLocation.centerFloat,
    // drawer: const NavigationDrawer(),
  );  // Scaffold
}
```

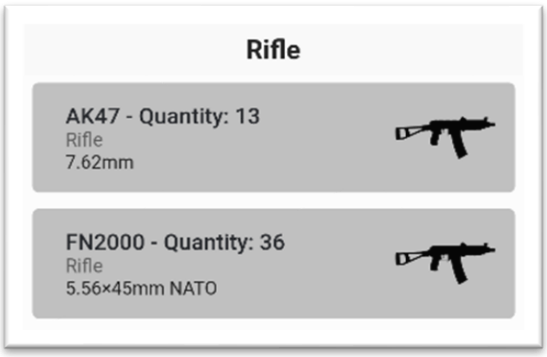
*Figure 24 Squad Weapon Build Method*

The general inventory page will be very similar to this page, however, instead of having the serial number and the active user, the weapon tiles will contain the quantity of weapons so that the soldier in charge of logistics can check how many of a specific firearm they have in storage.



```dart
child: GFListTile(
  padding: EdgeInsets.all(15),
  margin: EdgeInsets.all(6),
  color: Colors.grey[400],
  titleText: '${weapons.Name} - Quantity: ${weapons.Quantity}',
  subTitleText: '${weapons.Type}',
  description: Text('${weapons.Caliber}'),
  icon: SvgPicture.asset(
    "assets/icon/${weapons.Type}.svg",
    width: 35,
    height: 32,
  ),  // SvgPicture.asset
  onTap: () async {
    await openDialog(weapons);
    setState(() {});
  },
),  // GFListTile
```

*Figure 25 Weapon Tiles on Inventory Screens*



Rifle

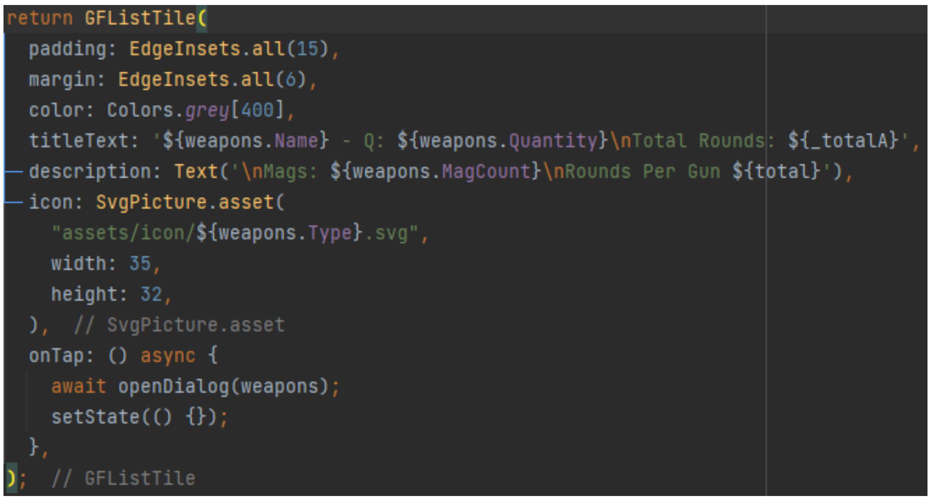
AK47 - Quantity: 13
Rifle
7.62mm

FN2000 - Quantity: 36
Rifle
5.56×45mm NATO

*Figure 26 Tile Examples*

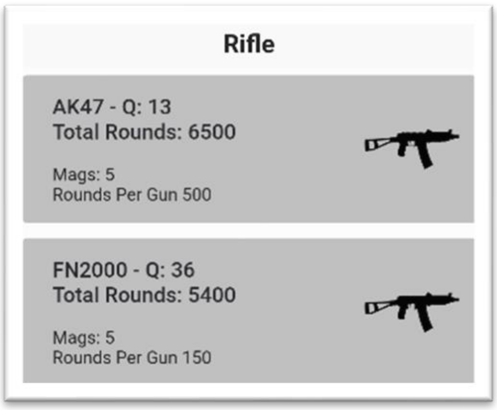
Each weapon can be clicked on and edited entirely. Along with this new weapons can be defined once again by using a floating action button found at the bottom of the page, giving users freedom to add what they like as the camera is constrained by how many weapons it is able to label.

Finally of the inventory screens we have the simplest but one of the most important pages which is the logistics page. This page is a new addition onto the prototype and will display to the user specifically how many rounds each weapon requires per soldier. Along with this it will display how many rounds would be needed of that weapons type if they were to want to deploy every firearm that they have in the inventory at that time. The data found here can be very helpful to officers organizing logistics and is able to be edited to fit the needs of soldiers and how much ammo they want each soldier to carry.



```
return GFListTile(
  padding: EdgeInsets.all(15),
  margin: EdgeInsets.all(6),
  color: Colors.grey[400],
  titleText: '${weapons.Name} - Q: ${weapons.Quantity}\nTotal Rounds: ${_totalA}',
  description: Text('\nMags: ${weapons.MagCount}\nRounds Per Gun ${total}'),
  icon: SvgPicture.asset(
    "assets/icon/${weapons.Type}.svg",
    width: 35,
    height: 32,
  ), // SvgPicture.asset
  onTap: () async {
    await openDialog(weapons);
    setState(() {});
  },
); // GFListTile
```

*Figure 27 Logistics ListTile Example*



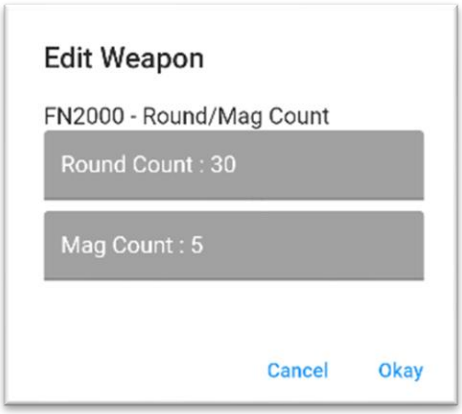
Rifle

AK47 - Q: 13
Total Rounds: 6500

Mags: 5
Rounds Per Gun 500

FN2000 - Q: 36
Total Rounds: 5400

Mags: 5
Rounds Per Gun 150



Edit Weapon

FN2000 - Round/Mag Count

Round Count : 30

Mag Count : 5

*Figure 28 Logistics Tile Example*

*Figure 29 Edit Logistics Tile Modal*

### 4.3.3 Codebase-Camera

The final page to discuss is the camera page. This is the primary page in the application as this is where most of the logic will happen along with all of the scanning for new weapons. When opening the page the camera controller will initialize the camera preview and show the user the camera view, along with a camera icon. When pressed this icon will begin the scanning process which will begin painting each weapon that is detected in the camera's preview. Along with this it will display a second icon. This icon is used to add new weapons to a list on the page, this list will be held onto for the confirmation page which come later.

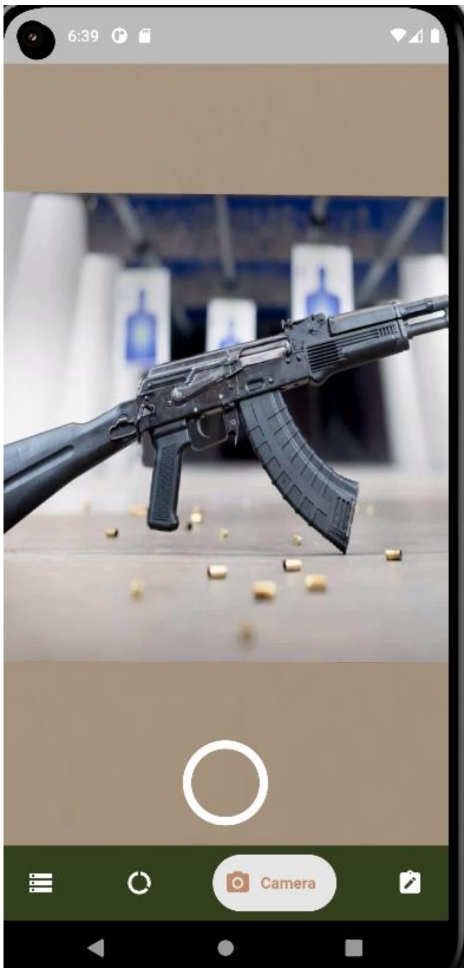


*Figure 30 Camera Not Scanning*

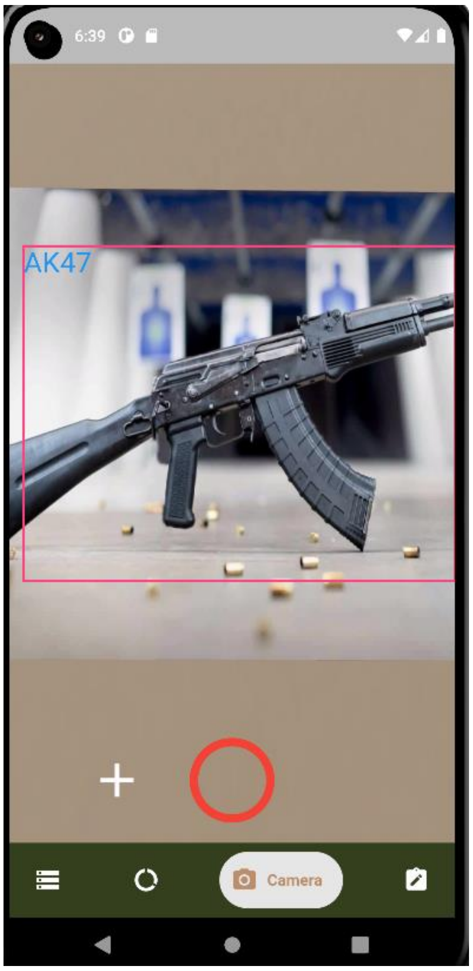
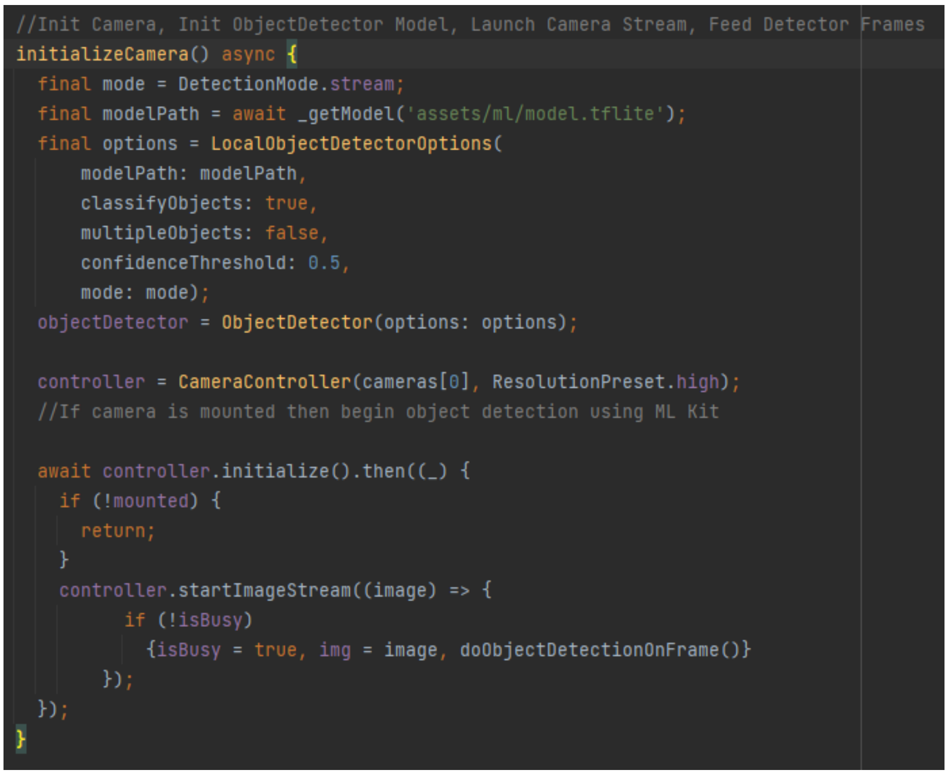
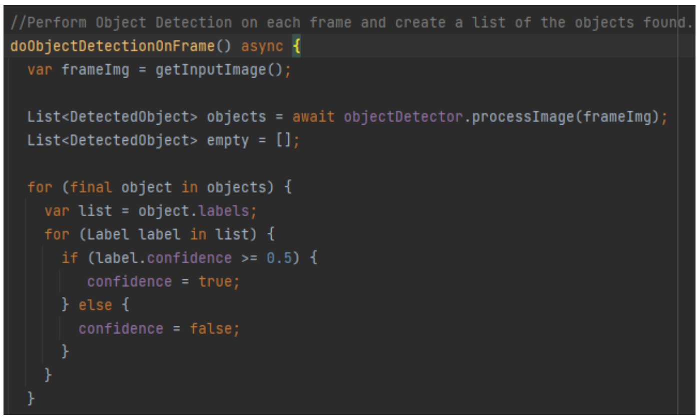


*Figure 31 Camera Scanning*



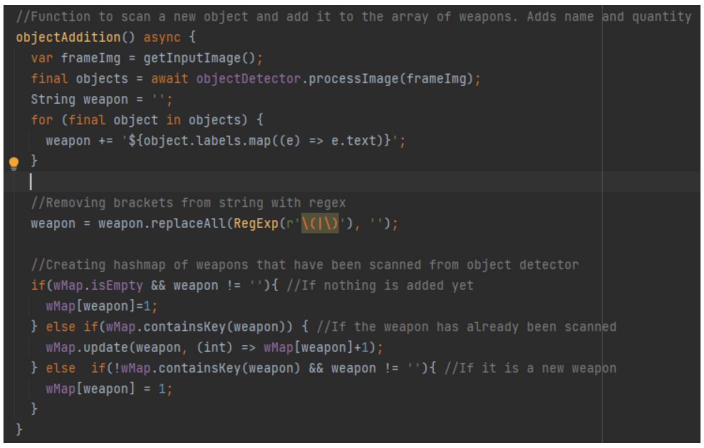
```
//Init Camera, Init ObjectDetector Model, Launch Camera Stream, Feed Detector Frames
initializeCamera() async {
  final mode = DetectionMode.stream;
  final modelPath = await _getModel('assets/ml/model.tflite');
  final options = LocalObjectDetectorOptions(
      modelPath: modelPath,
      classifyObjects: true,
      multipleObjects: false,
      confidenceThreshold: 0.5,
      mode: mode);
  objectDetector = ObjectDetector(options: options);

  controller = CameraController(cameras[0], ResolutionPreset.high);
  //If camera is mounted then begin object detection using ML Kit

  await controller.initialize().then((_) {
    if (!mounted) {
      return;
    }
    controller.startImageStream((image) => {
          if (!isBusy)
            {isBusy = true, img = image, doObjectDetectionOnFrame()}
        });
  });
}
```

*Figure 32 Camera Initialization Code*



```
//Perform Object Detection on each frame and create a list of the objects found.
doObjectDetectionOnFrame() async {
  var frameImg = getInputImage();

  List<DetectedObject> objects = await objectDetector.processImage(frameImg);
  List<DetectedObject> empty = [];

  for (final object in objects) {
    var list = object.labels;
    for (Label label in list) {
      if (label.confidence >= 0.5) {
         confidence = true;
      } else {
        confidence = false;
      }
    }
  }
```

*Figure 33 Object Detection Code*

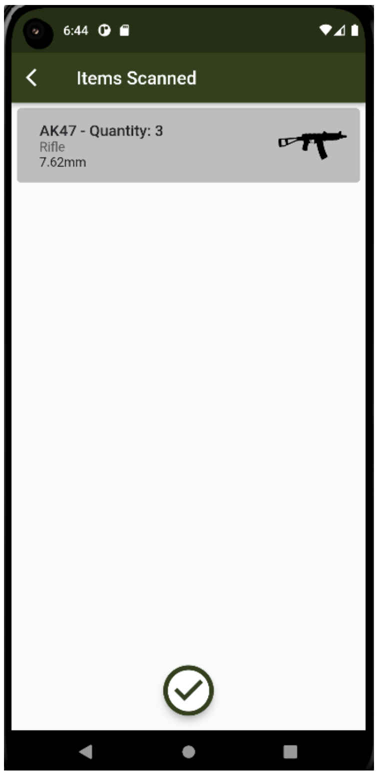


```
//Function to scan a new object and add it to the array of weapons. Adds name and quantity
objectAddition() async {
  var frameImg = getInputImage();
  final objects = await objectDetector.processImage(frameImg);
  String weapon = '';
  for (final object in objects) {
    weapon += '${object.labels.map((e) => e.text)}';
  }

  //Removing brackets from string with regex
  weapon = weapon.replaceAll(RegExp(r'\(|\)'), '');

  //Creating hashmap of weapons that have been scanned from object detector
  if(wMap.isEmpty && weapon != ''){ //If nothing is added yet
    wMap[weapon]=1;
  } else if(wMap.containsKey(weapon)) { //If the weapon has already been scanned
    wMap.update(weapon, (int) => wMap[weapon]+1);
  } else  if(!wMap.containsKey(weapon) && weapon != ''){ //If it is a new weapon
    wMap[weapon] = 1;
  }
}
```

*Figure 34 Object Addition to List Code*

Here can be seen the camera active and scanning, along with code snippets, displaying how the camera and object detector are initialized. Along with the functionality of object detection being performed and the function for the user to add weapons to the list of found weapons when they press the button.

After having populated the list with some firearms, the user will stop scanning. When they do this if the list contains data then a new page will be pushed. This page is the confirmation screen which will show users what they have scanned and ask them to confirm that it is correct. If it is they can click the check to confirm which will add the weapons to the Hive database and return the user to the camera.



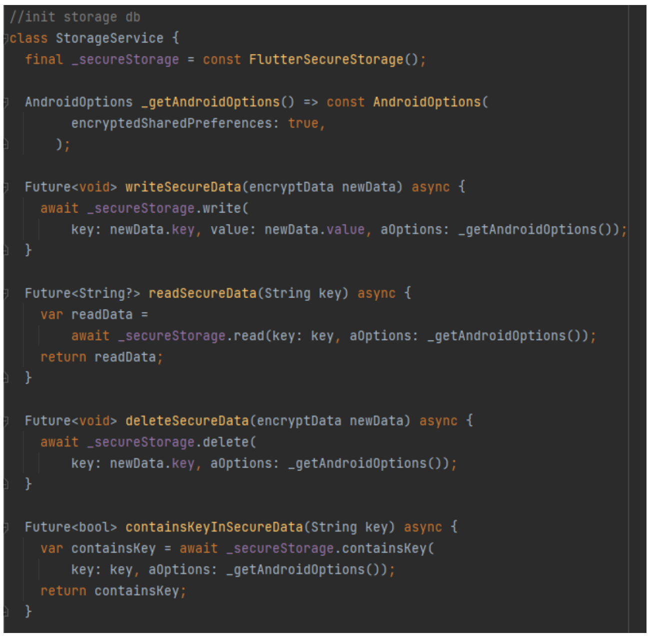
AK47 - Quantity: 3
Rifle
7.62mm

Figure 35 Scanned Items Confirmation Screen

### 4.4.1 Databases – Secure Storage(30)

The first of the two databases used in this project, Secure Storage is a plugin that is to be used in place of shared preferences for persistent data such as a user being logged in. Shared preferences is not recommended as it is entirely unsecure, whilst Secure Storage is encrypted via AES-256 Bit encryption, securing User details behind a layer of security. Originally the login was going to be handled by Firebase, however, during the development of the project it was decided that the app needed to be entirely local and disconnected from the internet. This was because the application stores sensitive data and will often be used where there is no available internet on the frontlines of war.

As such the NoSQL database Secure Storage is employed to secure and encrypt the user details of an individual who has installed the app on their phone, as well as this it will remember if they had logged in or not and verify all of their details from the login or pin screen when they login. This database doubles as security as the only way Hive will activate and begin to run is if the user can successfully login to their account, otherwise it will remain unreachable and decoupled. Along with this if a user turns out to be an unauthorized user who is not meant to be on the phone, there are security features in place to delete all data in both Hive and Secure Storage. Below is an example of some of the Secure Storages functionality.



```
//init storage db
class StorageService {
  final _secureStorage = const FlutterSecureStorage();

  AndroidOptions _getAndroidOptions() => const AndroidOptions(
        encryptedSharedPreferences: true,
      );

  Future<void> writeSecureData(encryptData newData) async {
    await _secureStorage.write(
        key: newData.key, value: newData.value, aOptions: _getAndroidOptions());
  }

  Future<String?> readSecureData(String key) async {
    var readData =
        await _secureStorage.read(key: key, aOptions: _getAndroidOptions());
    return readData;
  }

  Future<void> deleteSecureData(encryptData newData) async {
    await _secureStorage.delete(
        key: newData.key, aOptions: _getAndroidOptions());
  }

  Future<bool> containsKeyInSecureData(String key) async {
    var containsKey = await _secureStorage.containsKey(
        key: key, aOptions: _getAndroidOptions());
    return containsKey;
  }
```

*Figure 36 Secure Storage Code*

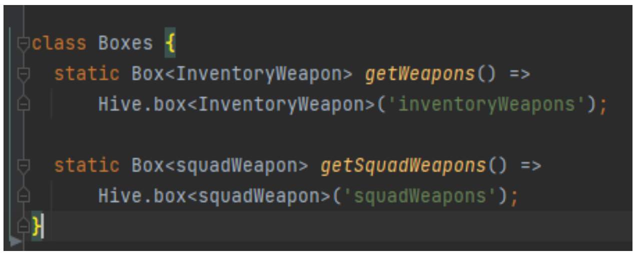
### 4.4.2 Databases – Hive(20)

The second database used in this application is the Hive database. This Database is also a NoSQL database encrypted with AES-256 Bit encryption to secure all of the sensitive data stored by the user. This database in particular is used to store all of the data of the weapons of a squadron or company, as well as the weapons that are in use and who is using them.

This is the database that will display to the pages what weapons are stored, as well as this it is used to perform the logistical analysis to calculate how many bullets are needed for every weapon in the inventory.

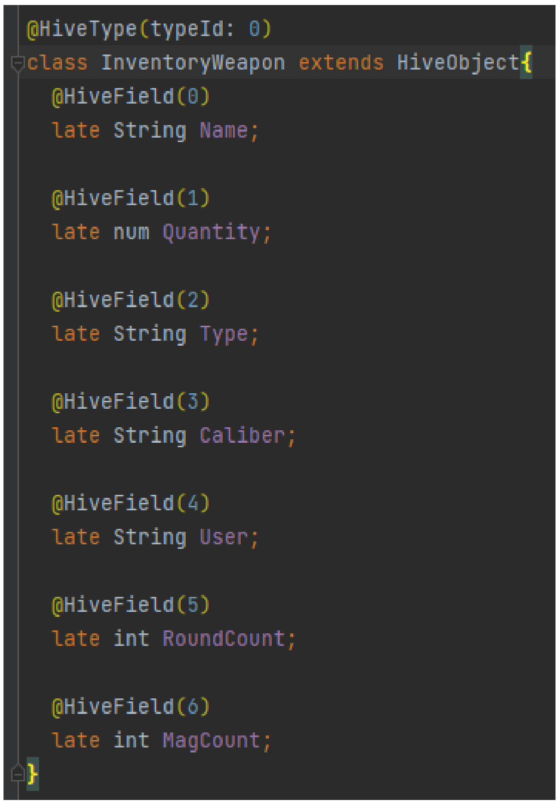
Hive operates through the use of boxes which act as containers which will store specific data, for me it is inventoryWeapons and squadWeapons that are held within these boxes. These boxes are called on and opened to retrieve the data contained within them and disposed of to close access to them once they are finished being used.

Below are examples of the boxes and an implementation of the inventoryWeapons class.



```
class Boxes {
  static Box<InventoryWeapon> getWeapons() =>
      Hive.box<InventoryWeapon>('inventoryWeapons');

  static Box<squadWeapon> getSquadWeapons() =>
      Hive.box<squadWeapon>('squadWeapons');
}
```

*Figure 37 Hive Boxes*



```
@HiveType(typeId: 0)
class InventoryWeapon extends HiveObject{
  @HiveField(0)
  late String Name;

  @HiveField(1)
  late num Quantity;

  @HiveField(2)
  late String Type;

  @HiveField(3)
  late String Caliber;

  @HiveField(4)
  late String User;

  @HiveField(5)
  late int RoundCount;

  @HiveField(6)
  late int MagCount;
}
```

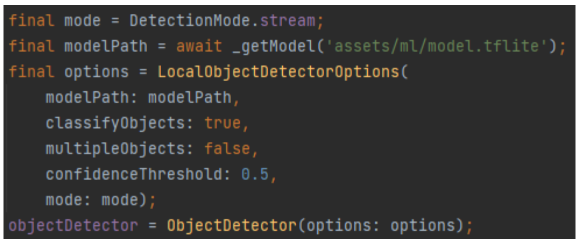
*Figure 38 Hive InventoryWeapon Class*

### 4.5.1 AI Model – Google ML Kit(15)

Google ML Kit was the choice of local object detector to deploy on the application. This is because the Google ML Kit package on pub dev is frequently updated and one of the largest and most supported packages there is. Offering many services, however, the only one of need for this project is Object Detection.

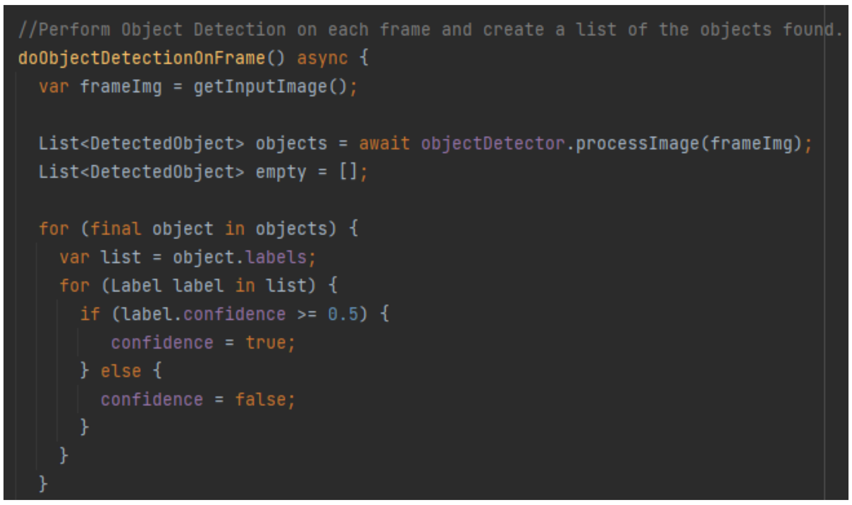
This package was deployed on the camera feature and is enabled when the user decides to begin scanning. Using the function doObjectDetection, this will begin the object detector and begin to call upon the custom-made model that was created for this project. Also calling upon a painter which will highlight the objects that the ML Kit extracts, and it will then highlight them with the label provided by the AutoML Model.

Below can be found a screenshot of the object detection method and the painting method for the application. Along with it's implementation method. This method has been changed from the prototype. As before in the prototype a weapon could be highlighted multiple times despite only 1 gun being in the frame. This was because the AI could recognize parts of the gun and the whole gun. Now it has been limited to only detect a single object in the camera feed to prevent issues like that from happening. Being allowed to filter the items it recognizes would be helpful as it is clear which parts of the gun the base model of the Object Detector recognizes. However, Google at this moment in time to do not allow for filtering(15).



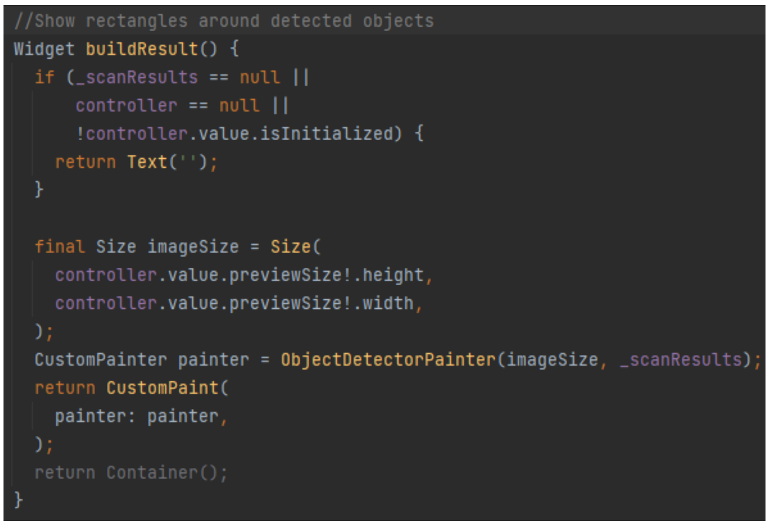
```
final mode = DetectionMode.stream;
final modelPath = await _getModel('assets/ml/model.tflite');
final options = LocalObjectDetectorOptions(
    modelPath: modelPath,
    classifyObjects: true,
    multipleObjects: false,
    confidenceThreshold: 0.5,
    mode: mode);
objectDetector = ObjectDetector(options: options);
```

*Figure 39 Object Detector Options*



```
//Perform Object Detection on each frame and create a list of the objects found.
doObjectDetectionOnFrame() async {
  var frameImg = getInputImage();

  List<DetectedObject> objects = await objectDetector.processImage(frameImg);
  List<DetectedObject> empty = [];

  for (final object in objects) {
    var list = object.labels;
    for (Label label in list) {
      if (label.confidence >= 0.5) {
         confidence = true;
      } else {
        confidence = false;
      }
    }
  }
```

*Figure 40 doObjectDetection Code*



```
//Show rectangles around detected objects
Widget buildResult() {
  if (_scanResults == null ||
      controller == null ||
      !controller.value.isInitialized) {
    return Text('');
  }

  final Size imageSize = Size(
    controller.value.previewSize!.height,
    controller.value.previewSize!.width,
  );
  CustomPainter painter = ObjectDetectorPainter(imageSize, _scanResults);
  return CustomPaint(
    painter: painter,
  );
  return Container();
}
```

*Figure 41 Object Painting Code*

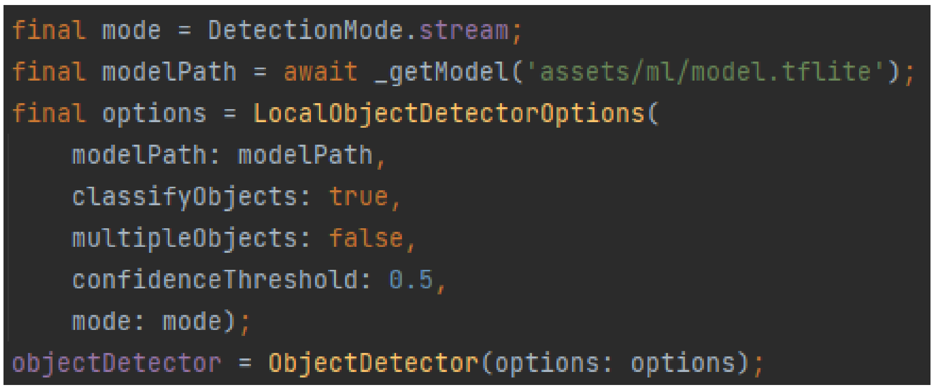
### 4.5.2 AI Model – Google AutoML Model(16)

Google AutoML is the choice of Model for this application as it was deemed superior to creating a model in Google Collab due to the time constraints of the project deadline. Google Collab would require full understanding of tensorflow and how to create a deep learning AI that can take input images and classify them.

As such instead Google AutoML on the Google Vertex AI platform was chosen. As this allows for the creation of a locally deployable Object Detection Model. As well as this training of this AI Model to the specific purpose is easier as a user can assign how many node hours they wish for their AI to train and can upload a custom dataset with custom labels to train the AI to recognize specific objects.

The AI used in this project has been trained on 3990 images which were web scraped from google images using Python. After this the AI has been trained on 3 different iterations, along with now a fourth which is deployed on the application. This iteration has refined labels to resolve issues with the difference between an AK47 and AK-74. As well as this the accuracy of the AI has been increased as in previous iterations it could highlight objects and think they were guns. This is because it had not been trained on enough data that was not weapons. As such 1000+ images were uploaded to it as as the "non-firearm" label to help the AI distinguish a firearm from regular objects.

The custom model has been added to the assets field of the project and is called in through the Google ML Kit to replace the basic object detector which can only label a few items. See below for the section where the Object Detector Model is called.



```
final mode = DetectionMode.stream;
final modelPath = await _getModel('assets/ml/model.tflite');
final options = LocalObjectDetectorOptions(
    modelPath: modelPath,
    classifyObjects: true,
    multipleObjects: false,
    confidenceThreshold: 0.5,
    mode: mode);
objectDetector = ObjectDetector(options: options);
```

*Figure 42 Configuring Custom AI Model Code*

## 4.6. Conclusions

In conclusion the application that has been developed is an offline local flutter application, deployed with Google AutoML and ML Kit to detect firearms that can be seen in a camera's view and then added to a Hive database. Along with this security has been implemented through the use of two different databases, one database requiring checks before activating the second database. Both being encrypted with AES-256 Bit Encryption standards to keep user data safe from unauthorized users who may have seized their phone and are combing through it for information.

Firebase has been entirely removed from the application due to security and network concerns and both AI's have been modified since the prototype to improve their performance and user experience alike.

Overall the application is simple to use and provides important logistical information that could be of use to a supply officer requesting ammunition for troops on the frontlines of war. Along with this it has simplified the messy task of manually documenting every weapon obtained on paper to logging every weapon on the soldiers phone. Along with this easing the process of identifying firearms for newer soldiers thanks to the Locally deployed AI.

# 5. Evaluation

## 5.1. Introduction

This chapter is going to encompass all of the testing and user evaluations of the application which was developed for this project. It will include the Manual Testing conducted, AI Testing, and the user feedback along with the changes made to the application due to user feedback.

## 5.2. Manual Testing

In order to test all of the UI and functionality that had been developed for this application it was important to conduct manual testing. This verified that pages and features function as expected. The primary method of manual testing was to write down specific actions that a user would be likely to conduct whilst using the application. Followed by then testing to guarantee that the user could or could not conduct the tests that were created

Here are the tests that were conducted on the application to ensure that a user's experience would not be interrupted by errors or fields allowing incorrect input.

| Login | Squad | General | Logistics | Camera | Confirmation | PIN Page | Logout |
|---|---|---|---|---|---|---|---|
| Users cannot mismatch passwords | Create squad weapon | Create weapon | All general weapons display | Open Camera Preview | Remove weapon | User see's pin when opening app and logged in. | User can logout |
| User PIN must 6 numbers. | Edit squad weapon | Edit weapon | Edit weapon fields | Begin Scanning | Add weapon to Database | User can enter PIN and enter app | User remains logged out when app turns off and on |
| User must fill in Security Questions. | Delete squad weapon | Delete weapon | | Scan guns | | Wrong PIN logs out user | |
| User is navigated to inventory upon creation. | Edit single field | Edit quantity | | Add gun to list | | | |
| Forgot Password works | | | | Navigate to Confirmation | | | |
| | | | | | | | |
| PASS | PASS | PASS | PASS | PASS | PASS | PASS | PASS |

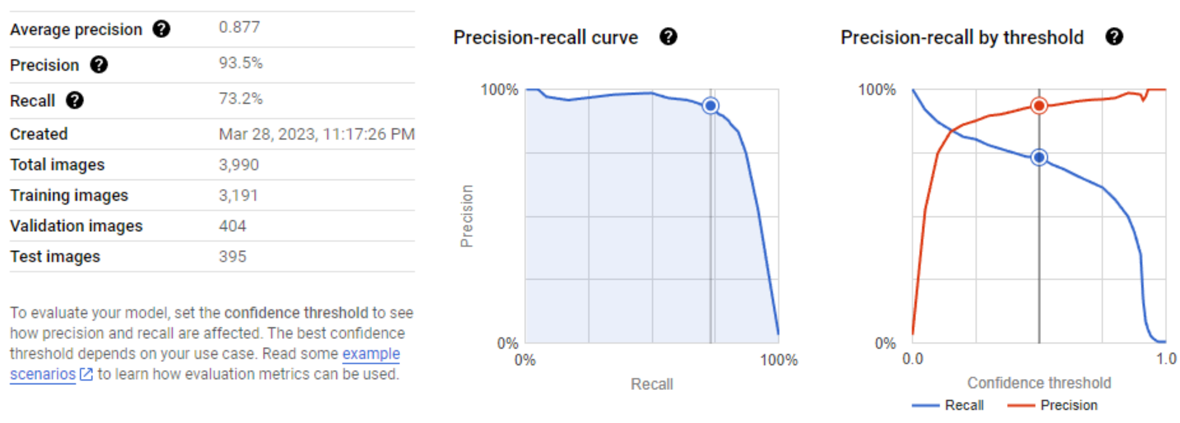
All primary features of the application and likely scenarios were tested to guarantee that a user will not encounter any bugs. Every feature selected was chosen as it was either a primary feature or one that would have a profound impact on the user experience if it was to stop functioning. After the completion of each feature, it would be tested multiple times in a variety of ways to guarantee that they would remain functioning, and if any exceptions were to occur, they would be handled.

## 5.3. AI Testing

Testing of the AI was crucial for the final application. The previous iterations had been accurate but had difficulties with different types of AK model. This was a huge problem as the most common model of firearm in the Ukrainian conflict is variants of the AK models, both the 5.45, and 7.62 variants. As such after multiple iterations of AI with different labels and slightly altered datasets the latest iteration and most accurate was developed. This version changing the labels from the countless variants of the AK models to simply identifying what calibre of AK model the camera was viewing. Allowing a user to later get more specific. This AI uses the AK_545 and AK_762 labels which greatly increased their accuracy. 87.2% and 79.2% respectively.

Along with this the overall accuracy of the AI entirely was increased and a massive issue of former iterations was resolved in this version as well. Previous iterations had been trained on only firearms. As a result when manually testing weapons the camera would be very accurate at identifying firearms when it see's them. However, it could also look at a wall and believe it was a firearm as well, or even a tea cup. As such it was realized that the AI needed to be trained on objects that were not firearms. This led to over 1000 images being fed to the AI in the "non_firearm" label which helped improve the accuracy of the AI greatly. Along with this the AI has for the most part stopped seeing firearms in everyday objects.

As well as this the overall precision and recall have been greatly improved, falling within the aim of 70-90% I had set for the training of this iteration. Having an 87% accuracy overall.

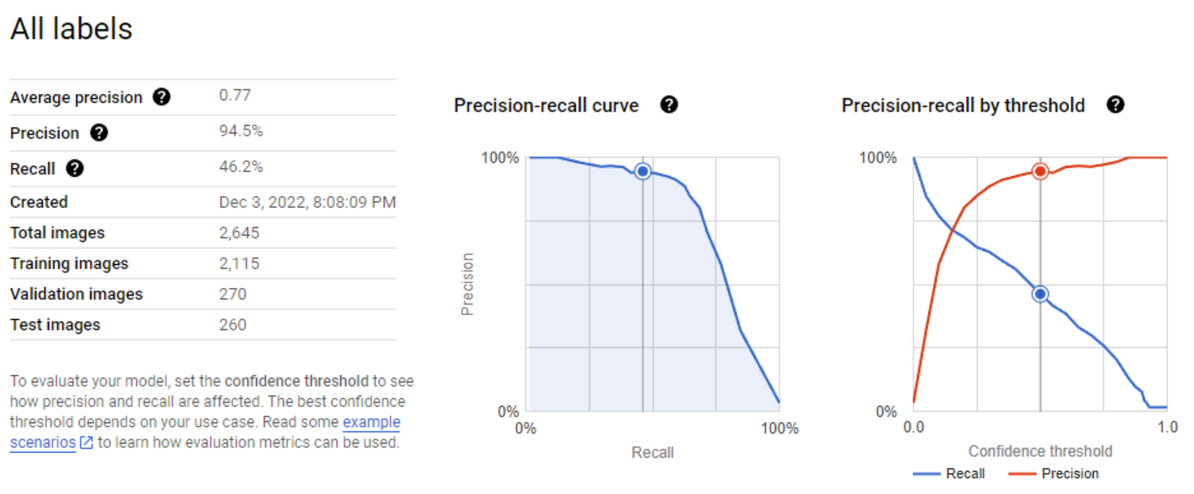


| | |
|---|---|
| Average precision | 0.877 |
| Precision | 93.5% |
| Recall | 73.2% |
| Created | Mar 28, 2023, 11:17:26 PM |
| Total images | 3,990 |
| Training images | 3,191 |
| Validation images | 404 |
| Test images | 395 |

To evaluate your model, set the confidence threshold to see how precision and recall are affected. The best confidence threshold depends on your use case. Read some example scenarios to learn how evaluation metrics can be used.

*Figure 43 New AI Test Results*

These results are a vast improvement over the latest previous iteration which pales in comparison to this dataset primarily in terms of recall, which means there are far less false negatives present. The only differences between them being that this dataset has been provided images that are not weapons, and the AK labels have been adjusted to reflect their calibre instead of their model. This small change resulted in a massive difference in recall and a 10% boost to overall confidence in every label. Below can be seen the previous training results of the latest previous iteration.



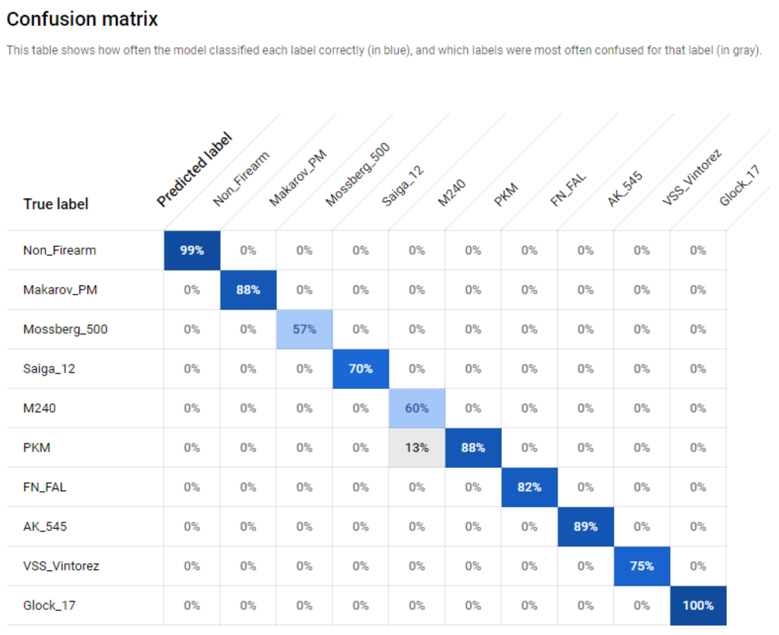
All labels

| | |
|---|---|
| Average precision | 0.77 |
| Precision | 94.5% |
| Recall | 46.2% |
| Created | Dec 3, 2022, 8:08:09 PM |
| Total images | 2,645 |
| Training images | 2,115 |
| Validation images | 270 |
| Test images | 260 |

To evaluate your model, set the confidence threshold to see how precision and recall are affected. The best confidence threshold depends on your use case. Read some example scenarios to learn how evaluation metrics can be used.

*Figure 44 Former AI Test Results*

Also having improved, the confusion matrix has grown far more consistent compared to previous iterations now and shows far less confusion as was seen before. With weapons often being mixed up in minor percentages or having vast issues in recognition when tested.



**Confusion matrix**

This table shows how often the model classified each label correctly (in blue), and which labels were most often confused for that label (in gray).

| True label \ Predicted label | Non_Firearm | Makarov_PM | Mossberg_500 | Saiga_12 | M240 | PKM | FN_FAL | AK_545 | VSS_Vintorez | Glock_17 |
|---|---|---|---|---|---|---|---|---|---|---|
| Non_Firearm | 99% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% |
| Makarov_PM | 0% | 88% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% |
| Mossberg_500 | 0% | 0% | 57% | 0% | 0% | 0% | 0% | 0% | 0% | 0% |
| Saiga_12 | 0% | 0% | 0% | 70% | 0% | 0% | 0% | 0% | 0% | 0% |
| M240 | 0% | 0% | 0% | 0% | 60% | 0% | 0% | 0% | 0% | 0% |
| PKM | 0% | 0% | 0% | 0% | 13% | 88% | 0% | 0% | 0% | 0% |
| FN_FAL | 0% | 0% | 0% | 0% | 0% | 0% | 82% | 0% | 0% | 0% |
| AK_545 | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 89% | 0% | 0% |
| VSS_Vintorez | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 75% | 0% |
| Glock_17 | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 100% |

*Figure 45 AI Confusion Matrix*

Overall the AI has been vastly improved over previous iterations thanks to the addition of the "non_firearm" label which stops the AI highlighting items such as cups as guns. As well as this helping to increase the overall accuracy from 77% to 87% total in the dataset and boosting the recognition of AK models which was the primary concern thanks to the new labelling applied to them.

## 5.4. User Feedback

User feedback was an important and crucial step to the testing of this application. As when developing a product an engineer can have a bias towards the application and assume some features may be obvious in how they function, when to non-technical individuals may in fact struggle to understand what the application wants from a user.

As such user feedback was necessary, especially from the target demographic of Ukrainian Servicemen as if they cannot understand the usage of the application, then the application has already failed in it’s purpose.

For the purpose of gaining user feedback a Ukrainian Soldier by the name of Sergiy was reached out to in the hopes of gaining a tester. Thankfully he agreed and gave very insightful feedback that led to some minor changes being made to the application.

Below will be listed some of the most crucial feedback to changes. Overall Sergiy believed the interface to be good and usable, however, he had some suggestions to make the app more user friendly and understandable.

*“in the first ‘rifle’ tab, it's better to put AKM as an icon, because AKSU is more of a submachine gun than an assault rifle and it is less common in culture, so it is difficult for some people to perceive.”*

His first suggestion was to adjust the Icons which had been used to display the weapons, in specific the Rifles. As a result of the suggestion the AKSU has been changed to the G36C, a more common NATO Assault Rifle.

*“- in the second tab, if the calibre is written, then it is specified completely. For example, 7.62x39. 7.62 mm many different types of this calibre - there are 18 mm (TT pistol), 39 mm (AKM), 51 mm (308. , NATO), 54 mm (Soviet) - this is the length of the sleeve (or cartridge, I don’t remember) and it would be nice to point it out.”*

The second insight provided by Sergiy was to adjust the edit weapons and add weapons modals. This was because the User was allowed to enter in any Calibre they chose. This was done as there are hundreds of calibres. However, he said that they primarily use Post-Soviet and NATO standard calibres, and as such allowing all made less sense than specifying the most common in their exact formats.

As such the weapon dialogues have been edited from before where they would allow any calibre to accepted, as shown below in the first image. After the suggestions a JSON of every common calibre in use with Sergiys advice was created, and now a dropdown is shown instead.

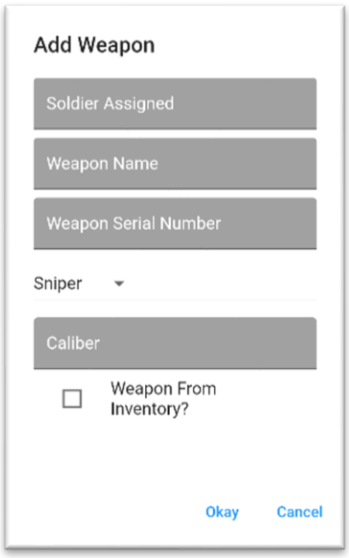


*Figure 46 Old Add Weapon Modal*

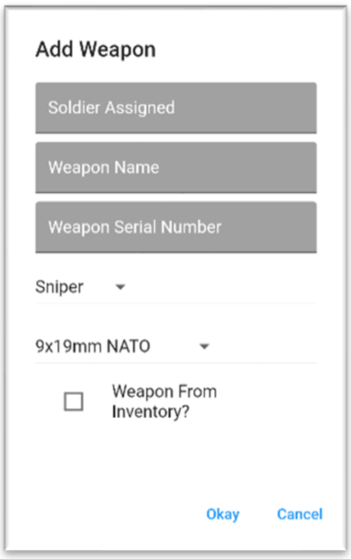


*Figure 47 New Add Weapon Modal*

After making these changes there was no remaining feedback of anything that Sergiy would change. The UI and general function and feel of the app was perfectly adequate. He deemed it to be useful for an individual like a Platoon Chief Sergeant who is responsible for requesting supplies.

*"In general, as a platoon chief sergeant who counts the number of fighters and applies to a company commander for ammunition, this is good".*

## 5.5. Conclusion

Through thorough testing of the application using manual testing and expected user actions, combined with extensive AI testing, and relabelling and adjusting of the dataset. Along with user feedback from an active-duty soldier in the Ukrainian military, this project has produced a fully functional application that is near entirely complete. Future development will be far less intensive due to the constant testing the project has undertaken to ensure no bugs will take down other pages and ruin the user experience.

# 6. Conclusion and Future Work

## 6.1. Introduction

This chapter will cover the conclusions of this project and the application that was developed for the project. As well as the future plans that await this application that were not able to be implemented in time for the projects deadline.

### 6.2.1 Conclusions – AI

Google ML Kit and Google AutoML on the Vertex AI platform were implemented into the application. Chosen over Google collab to minimize risk of complications and for concerns of how long it may take to create and implement an AI created with Google collab.

Google ML Kit was implemented with relative ease after working out multiple issues with its connection to the camera thanks to the large support of it's community and popularity. Along with this a custom Object Detection model was uploaded onto it. Developed using a custom dataset of 3990 firearms and everyday items. The Custom-Dataset had to be hand made due to INTERPOL and NATO restrictions on access to their Datasets of firearms. And outside of these two datasets no other dataset exists on individual firearms categorized as they were in this project's dataset. The AutoML model was developed with over 12 total training hours in the different iterations of it as labels and data in the dataset were changed.

The outcome of this resulted in an AI with a high accuracy that was able to detect firearms with speed and accuracy for the application to then add to its database.

### 6.2.2 Conclusions – Application UI

The UI of this project was developed using the Flutter Framework and many of the latest and most popular packages. These packages allowing for additional features to be implemented and for the user experience to be improved and simplified.

Along with this thanks to the chosen language being Flutter the app is capable of being cross-platform. Meaning IOS and Android users alike will be able to utilize this application on the frontlines of the war in Ukraine. Flutter is also a lightweight framework meaning even older phones will have no issues with running the application, as there was a specific emphasis placed on making the application widely available. Flutter overall allowed for a lightweight application available to many, with a clean User Interface and efficient functionality.

### 6.2.3 Conclusions – Database

Hive and Secure Storage are the chosen databases for this application. Hive allowing for all of the users weapons to be stored and secured with AES-256 Bit Encryption. And Secure Storage storing User account details which are also secured behind AES-256 Bit Encryption. The choice of two databases allowed for additional security to be in place. As Secure Storage will require checks on the User to be completed, and if the details provided are correct then it will allow for the Hive Boxes to be activated and for information to be displayed to the user. This also allowed for Hive to be entirely decoupled and inaccessible from the front end.

The overall final product having fully functional databases to store both User credentials and User information that they gather and create whilst using the application. Entirely secure and encrypted to protect from unauthorized users in the event of the phone being captured.

### 6.2.4 Conclusions – Application Security

In terms of security the Application itself uses AES-256 Bit Encryption to prevent unauthorized users from viewing the applications data easily. As well as this the Hive database containing all of a user's firearms and data is entirely decoupled from the login screen.  So if a user logs out they have the database cut off, and if they are logged in a PIN is demanded or else the user will be forcefully logged out. For additional security and as a last resort in the event of a phone being captured. Should a user fail to login or answer the security questions correctly three times. Then the entire database of both Secure Storage and Hive will be wiped to prevent a data breach.

## 6.3. Future Work

Thankfully the project was able to proceed mostly on time despite some minor hiccups such as bugs and the Python web scraping library for Google-Images becoming abandoned. As such there is not much future work remaining for the application Zbroya. There is only one known bug to be fixed and two new features that were not able to be implemented in time.

### 6.3.1 Object Detector Bug

The Object detector deployed on the application has a bug that despite multiple attempts to fix has remained persistent. The bug is caused when the camera is moved too quickly and shakily such as swinging it back and forth. For an indiscernible reason this causes the object detector Google ML Kit provides to break due to an inputStream error. All attempts to fix this have not worked and it is noticeable when the camera stops painting all together. It required a refresh of the camera to fix, however, it was no often that this occurred and was simple to fix.

### 6.3.2 Translations

Translations were a new feature that was planned to be implemented but due to time constrains they were unable to be implemented. It was going to be a simple toggle in the drawer menu that would allow a user to switch between English and Ukrainian on the application. The translations would have been provided from Ukrainian speaking individuals willing to translate the different strings used in the application. However, due to time constraints this feature was pushed back to a nice-to-have feature and in the end was not implemented.

### 6.3.3 PDF Export

Much like translations, this was a new feature that was not implemented due to a lack of time remaining and more primary features needing more time dedicated towards refactoring them. As such this feature was also sacrificed for later work. It would have allowed a user to create a PDF copy of their logistics page. This would allow for a Platoon Chief to simply forward a commander the munitions required. However, as said due to a lack of time this feature was not implemented.

## 6.4. Conclusion

In conclusion to this project, Zbroya is a near completed application only missing minor features, but delivering on all promised primary functionality and more. The development of this application has been a valuable learning exercise and has offered much experience, especially in the fields of Mobile Development and Deep Learning.

The final product is finished to the point in which it could be deployed fully to the Android and IOS app stores for Ukrainian soldiers to make use of on the frontlines of their war. Filling in for a vital role on the battlefield that can make the difference between the platoon being prepared for an engagement or not.

## Bibliography

1. Security Camera Gun Detection - ZeroEyes [Internet]. [cited 2022 Oct 23]. Available from: https://zeroeyes.com/

2. Tony W. Recognizing Firearms from Images and Videos in Real-Time with Deep Learning and Computer Vision. Medium [Internet]. 2022 Aug 11 [cited 2022 Nov 10]; Available from: https://medium.com/@tont/recognizing-firearms-from-images-and-videos-in-real-time-with-deep-learning-and-computer-vision-661498f45278

3. Makkena B. Artificial Intelligence and Deep Learning for Weapon Identification in Security Systems [Internet]. PACE Institute of Technology and Sciences; 2021 [cited 2022 Nov 7]. Available from: https://iopscience.iop.org/article/10.1088/1742-6596/2089/1/012079/pdf

4. Patrick Bollhoff. Kotlin vs Java [Internet]. 2022 [cited 2022 Oct 22]. Available from: https://kruschecompany.com/kotlin-vs-java/

5. Kristen Carter. How Android App Development Became Kotlin First [Internet]. 2019 [cited 2022 Oct 22]. Available from: https://medium.com/hackernoon/how-android-app-development-became-kotlin-first-c79e493e02fb

6. Frederic Lardinois. Kotlin is now Google's preferred language for Android app development [Internet]. 2019 [cited 2022 Oct 29]. Available from: https://techcrunch.com/2019/05/07/kotlin-is-now-googles-preferred-language-for-android-app-development/

7. Kotlin vs Flutter 2023: Which one is Better [Internet]. 2022 [cited 2022 Oct 24]. Available from: https://fireart.studio/blog/what-about-flutter-vs-kotlin/

8. Kinza Yasar. Pytorch [Internet]. [cited 2022 Oct 23]. Available from: https://www.techtarget.com/searchenterpriseai/definition/PyTorch#:~:text=PyTorch%20is%20an%20open%20source,platforms%20for%20deep%20learning%20research.

9. Pytorch and pre-trained weights [Internet]. [cited 2022 Oct 23]. Available from: https://pytorch.org/vision/stable/models.html#:~:text=TorchVision%20offers%20pre%2Dtrained%20weights,weights%20to%20a%20cache%20directory.

10. About Keras [Internet]. [cited 2022 Oct 24]. Available from: https://keras.io/about/#:~:text=Keras%20is%20a%20deep%20learning,key%20to%20doing%20good%20research.

11. Jack Vaughan. TensorFlow [Internet]. [cited 2022 Oct 24]. Available from: https://www.techtarget.com/searchdatamanagement/definition/TensorFlow#:~:text=TensorFlow%20is%20an%20open%20source,statistical%20and%20predictive%20analytics%20workloads.

12. TfLite for ML Kit [Internet]. [cited 2022 Oct 26]. Available from: https://firebase.google.com/docs/ml-kit/android/use-custom-models#:~:text=ML%20Kit%20can%20use%20TensorFlow,to%20different%20sets%20of%20users.

13. Ayoub Benali Amjoud, Mustapha Amrouch. Convolutional Neural Networks Backbones for Object Detection. In 2020 [cited 2022 Oct 27]. Available from: https://link.springer.com/chapter/10.1007/978-3-030-51935-3_30

14. Jupyter Notebook vs. Google Colab [Internet]. 2022 [cited 2022 Oct 27]. Available from: https://blog.reviewnb.com/jupyter-notebook-google-colab/

15. Pub Dev. Google ML Kit [Internet]. [cited 2022 Dec 3]. Available from: https://pub.dev/packages/google_ml_kit

16. Vertex AI for AutoML users [Internet]. [cited 2022 Oct 29]. Available from: https://cloud.google.com/vertex-ai/docs/start/automl-users

17. YOLO: Real-Time Object Detection [Internet]. [cited 2022 Oct 30]. Available from: https://pjreddie.com/darknet/yolo/

18. Muhammad Monjurul Karim, David Doell, Ravon Lingard, Zhaozheng Yin, Ming C. Leu, Ruwen Qin. A Region-Based Deep Learning Algorithm for Detecting and Tracking Objects in Manufacturing Plants. In Chicago, Illinois, USA; 2019 [cited 2022 Nov 1]. Available from: https://doi.org/10.1016/j.promfg.2020.01.289

19. How single-shot detector (SSD) works [Internet]. [cited 2022 Nov 4]. Available from: https://developers.arcgis.com/python/guide/how-ssd-works/#:~:text=SSD%20uses%20a%20matching%20phase,object's%20class%20and%20its%20location.

20. Hive [Internet]. [cited 2022 Nov 22]. Available from: https://pub.dev/packages/hive

21. INTERPOL Firearms Reference Table [Internet]. INTERPOL; [cited 2022 Nov 6]. Available from: https://www.interpol.int/en/Crimes/Firearms-trafficking/INTERPOL-Firearms-Reference-Table

22. Valldor E, Gustafsson D. Firearm Detection in Social Media. [cited 2022 Nov 12]; Available from: https://www.sto.nato.int/publications/STO%20Meeting%20Proceedings/STO-MP-IST-160/MP-IST-160-S2-4.pdf

23. Qi D. A Gun Detection Dataset and Searching for Embedded Device Solutions. 2021 Mar 5 [cited 2022 Nov 16]; Available from: https://deepai.org/publication/a-gun-detection-dataset-and-searching-for-embedded-device-solutions#S1.p2

24. Battlestate Games. Escape From Tarkov [Internet]. [cited 2022 Dec 4]. Available from: https://www.escapefromtarkov.com/

25. 7 Things about Feature Driven Development [Internet]. [cited 2022 Nov 27]. Available from: https://lvivity.com/7-things-about-feature-driven-development

26. Phuong Anh Nguyen. The Best Software Methodologies for Small Teams [Internet]. 2021 [cited 2022 Nov 27]. Available from: https://wearefram.com/blog/software-development-methodologies/

27. What is Kanban? Explained for Beginners. [Internet]. [cited 2022 Nov 27]. Available from: https://kanbanize.com/kanban-resources/getting-started/what-is-kanban

28. Nick Hotz. What is CRISP-DM? [Internet]. [cited 2022 Dec 5]. Available from: https://www.datascience-pm.com/crisp-dm-2/

29. CRISP-DM Help Overview [Internet]. 2021 [cited 2022 Dec 5]. Available from: https://www.ibm.com/docs/en/spss-modeler/saas?topic=dm-crisp-help-overview

30. steenbakker.dev. Secure Storage [Internet]. 2023. Available from: https://pub.dev/packages/flutter_secure_storage

31. Gun Detection Deep Learning Algorithm [Internet]. [cited 2022 Nov 11]. Available from: https://viso.ai/application/weapon-detection/

32. File-Based Encryption on Android Devices [Internet]. File-Based Encryption. 2022 [cited 2022 Nov 14]. Available from: https://source.android.com/docs/security/features/encryption/file-based

# Appendix

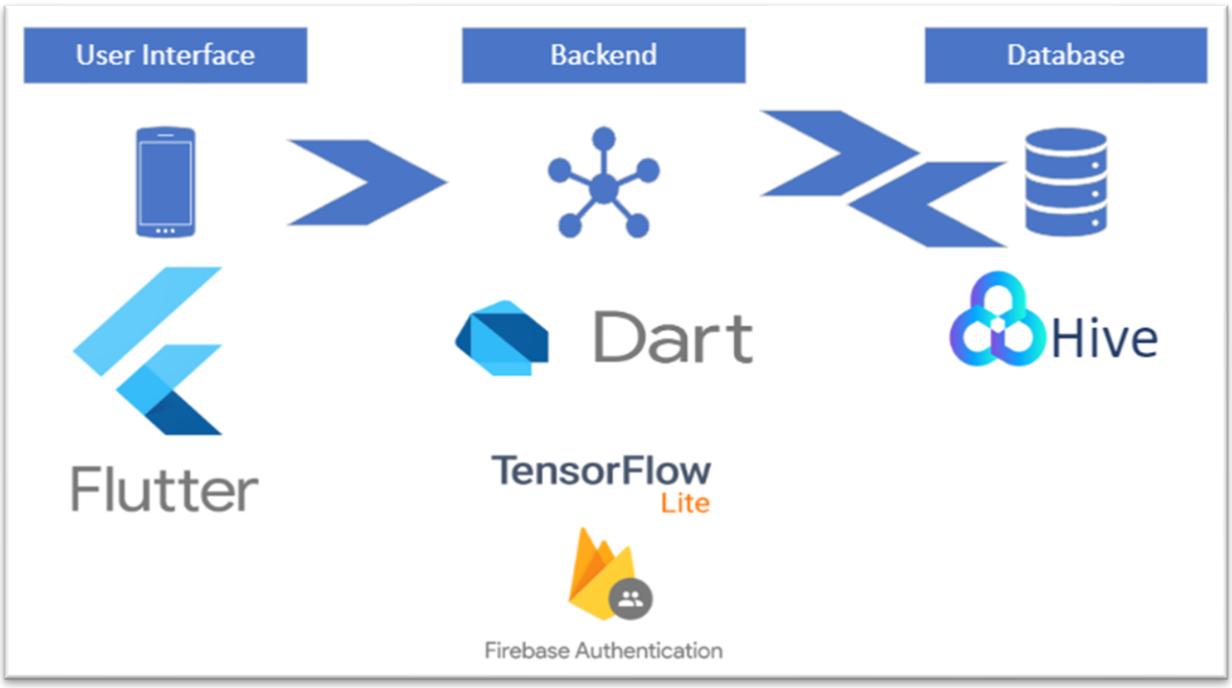


*Figure 48 Old System Architecture*

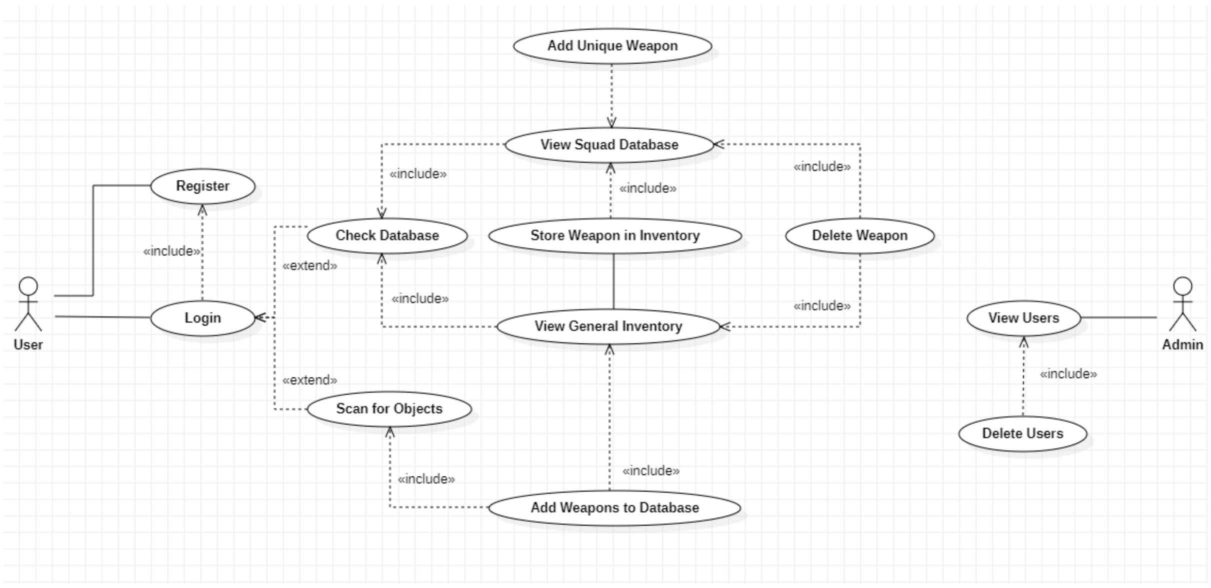


*Figure 49 Old Use Case Diagram of the Prototype*

| User | Weapon |
|---|---|
| `{`<br>`        “Username”: string,`<br>`        “Pin”: int`<br>`}` | `{`<br>`        “Weapon”: string`<br>`         {`<br>`                “Type”: string,`<br>`                “ID”: int,`<br>`                “Quantity”: int,`<br>`                “Operator”: string`<br>`         }`<br>`}` |

*Figure 50. Old Database Diagram*

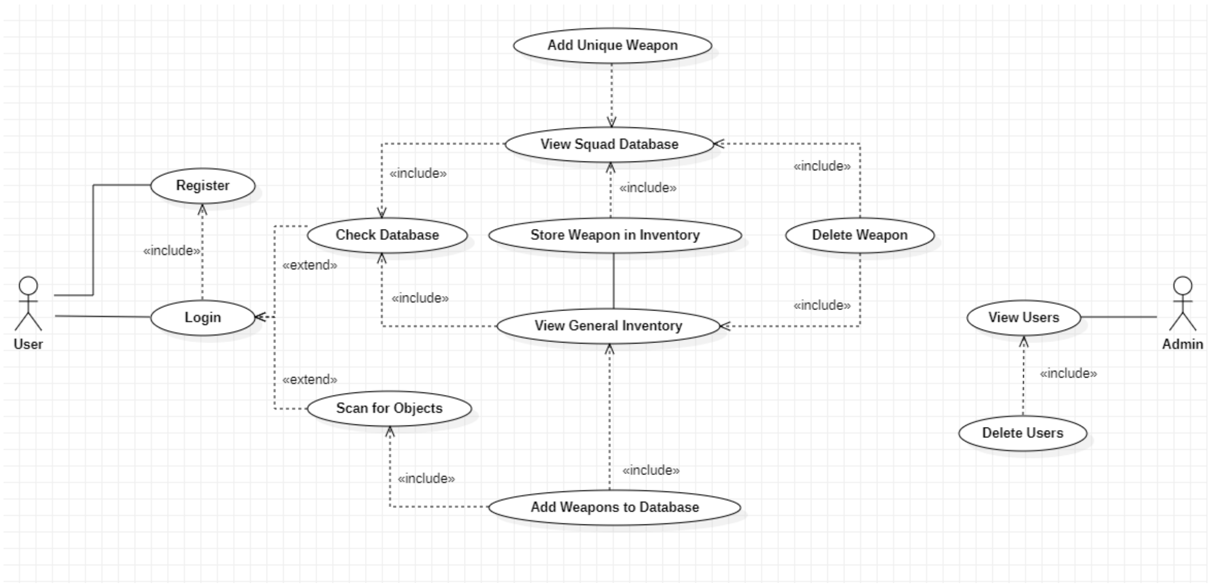


*Figure 51 Old Use Case Diagram of the Prototype*

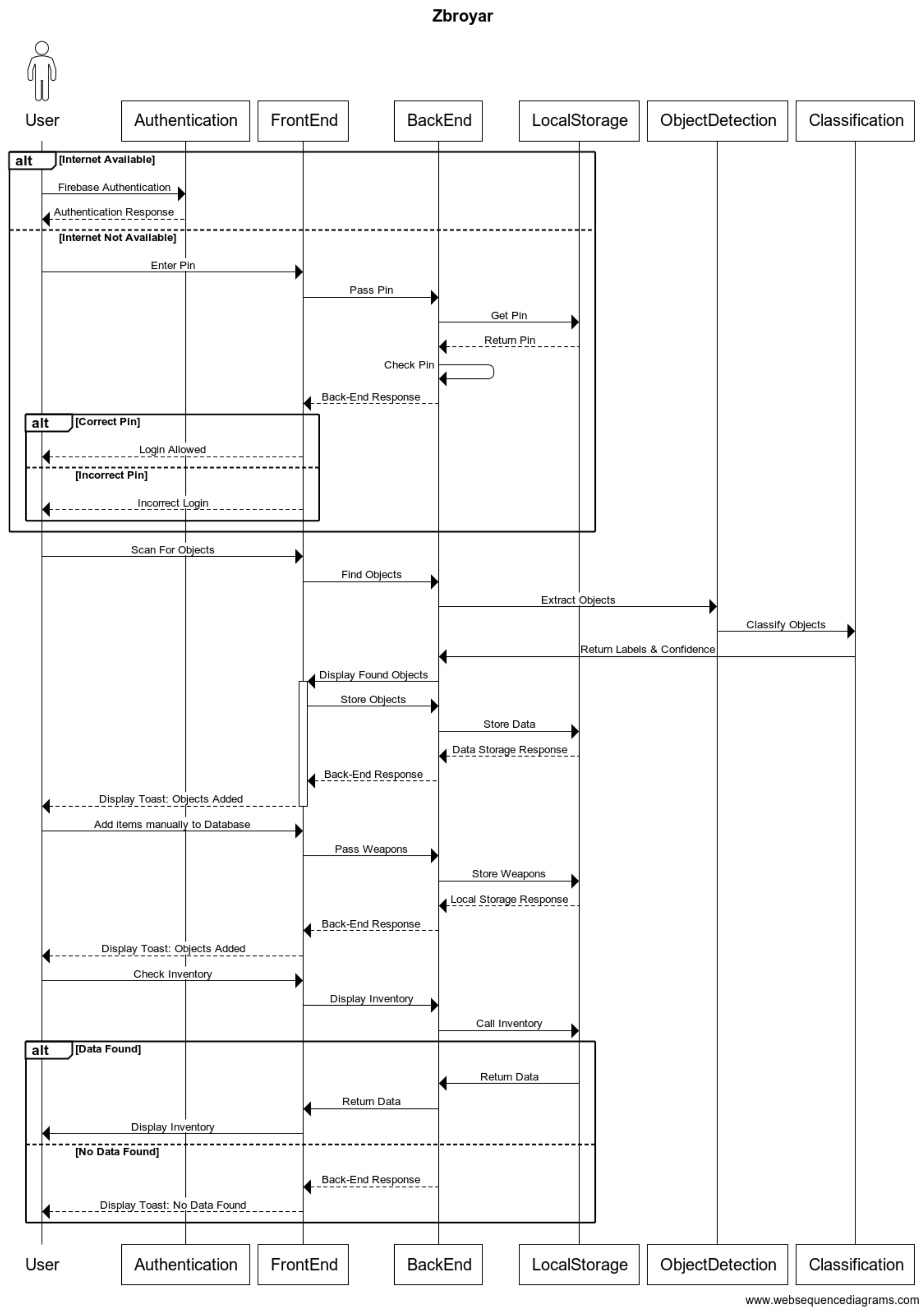


*Figure 52 Old Sequence Diagram of the Prototype*

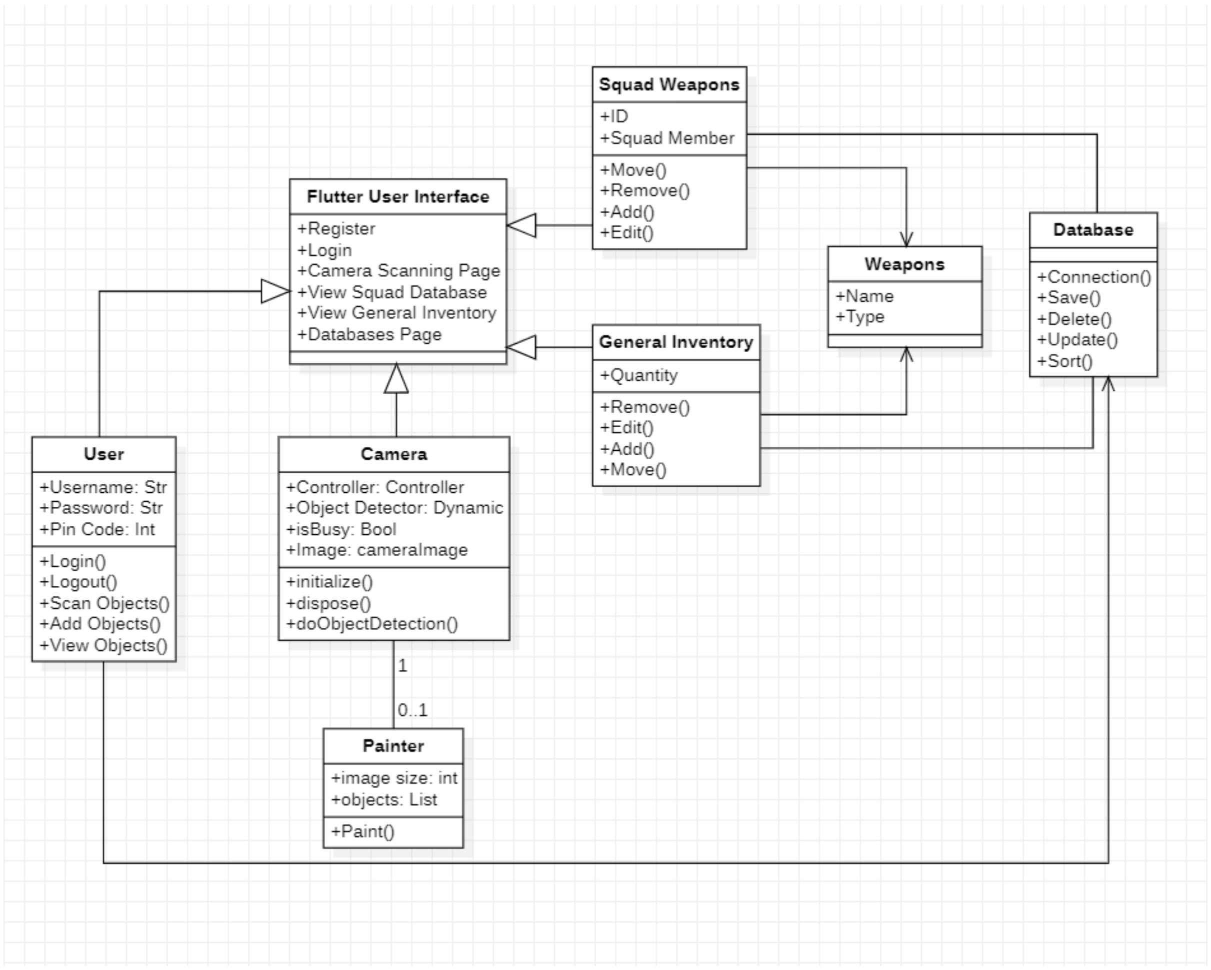


*Figure 53 Old Class Diagram of the Prototype*

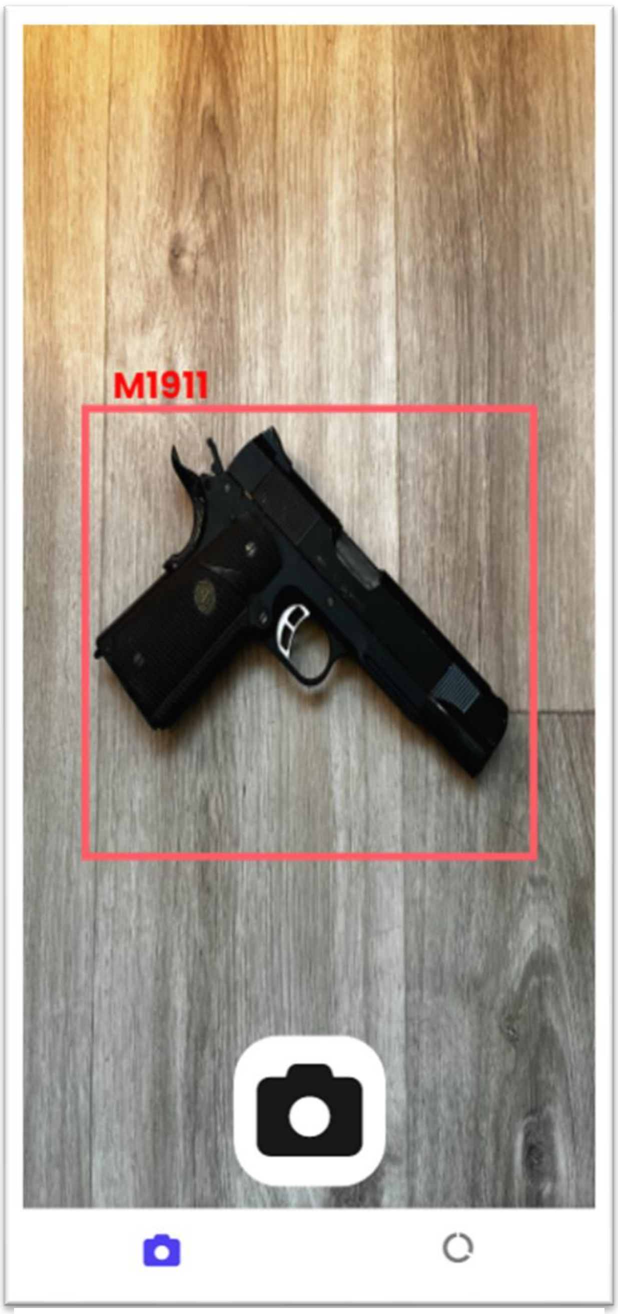


*Figure 54 Old Prototype Object Detection View*

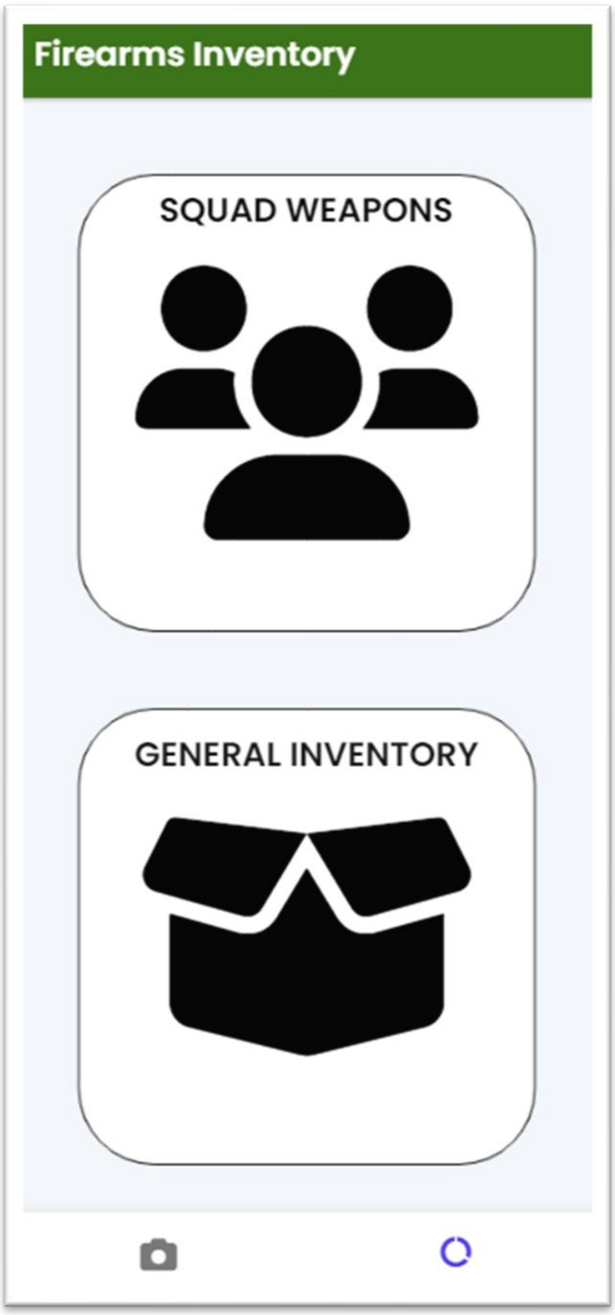
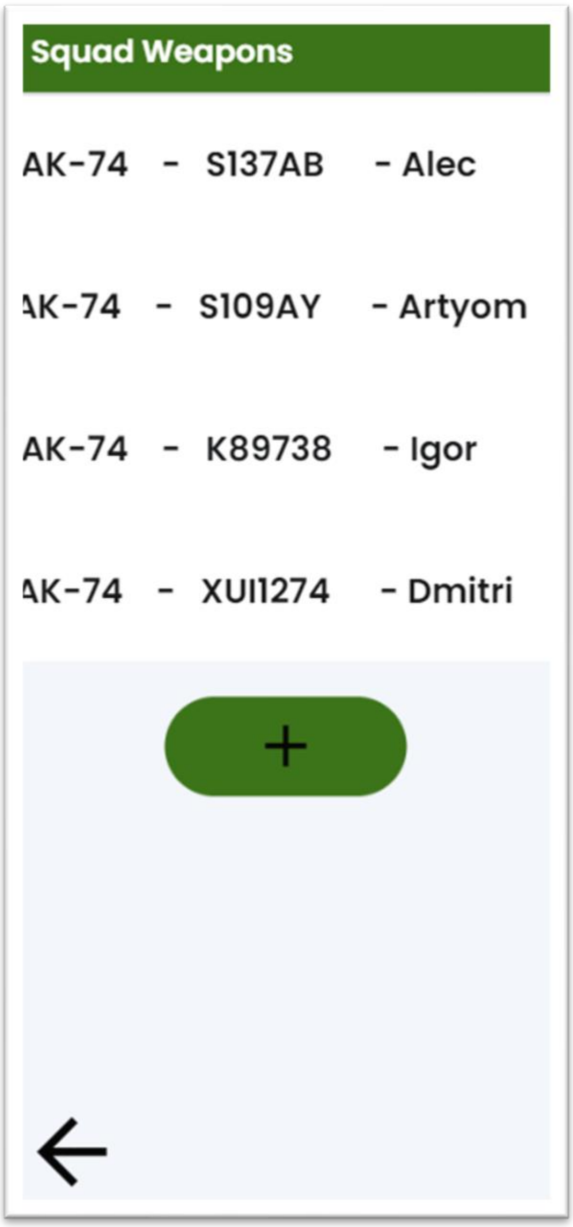


*Figure 55 Old Prototype Database View*



Squad Weapons

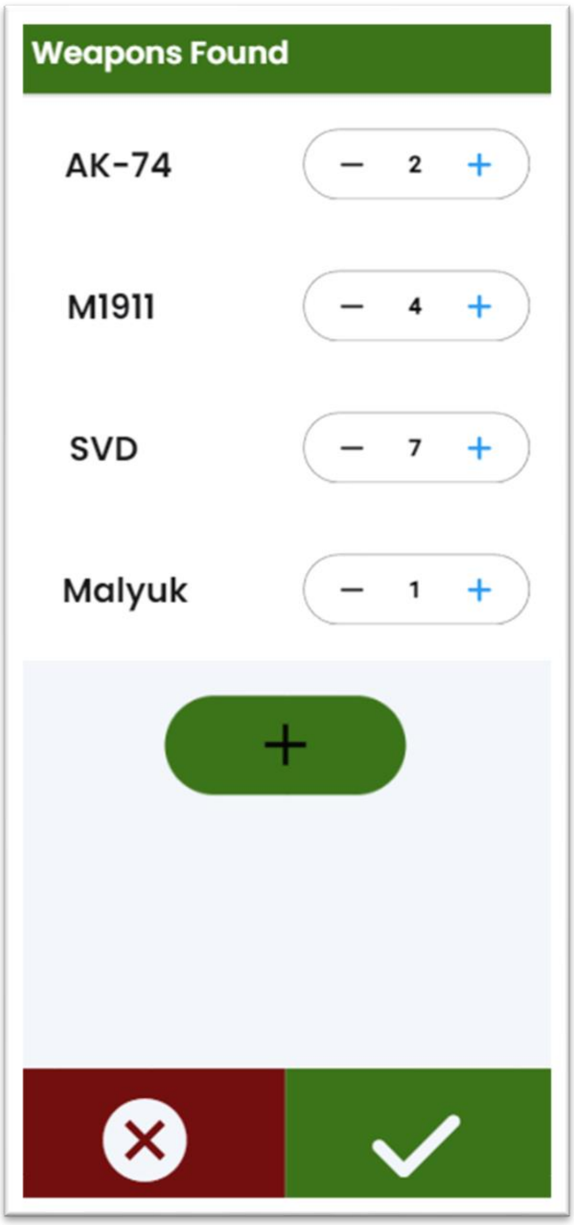
AK-74 - S137AB - Alec

AK-74 - S109AY - Artyom

AK-74 - K89738 - Igor

AK-74 - XUI1274 - Dmitri

*Figure 57 Old Squad Weapons*



Weapons Found

| | |
|---|---|
| AK-74 | 2 |
| M1911 | 4 |
| SVD | 7 |
| Malyuk | 1 |

*Figure 56 Old Object Detection Modal*

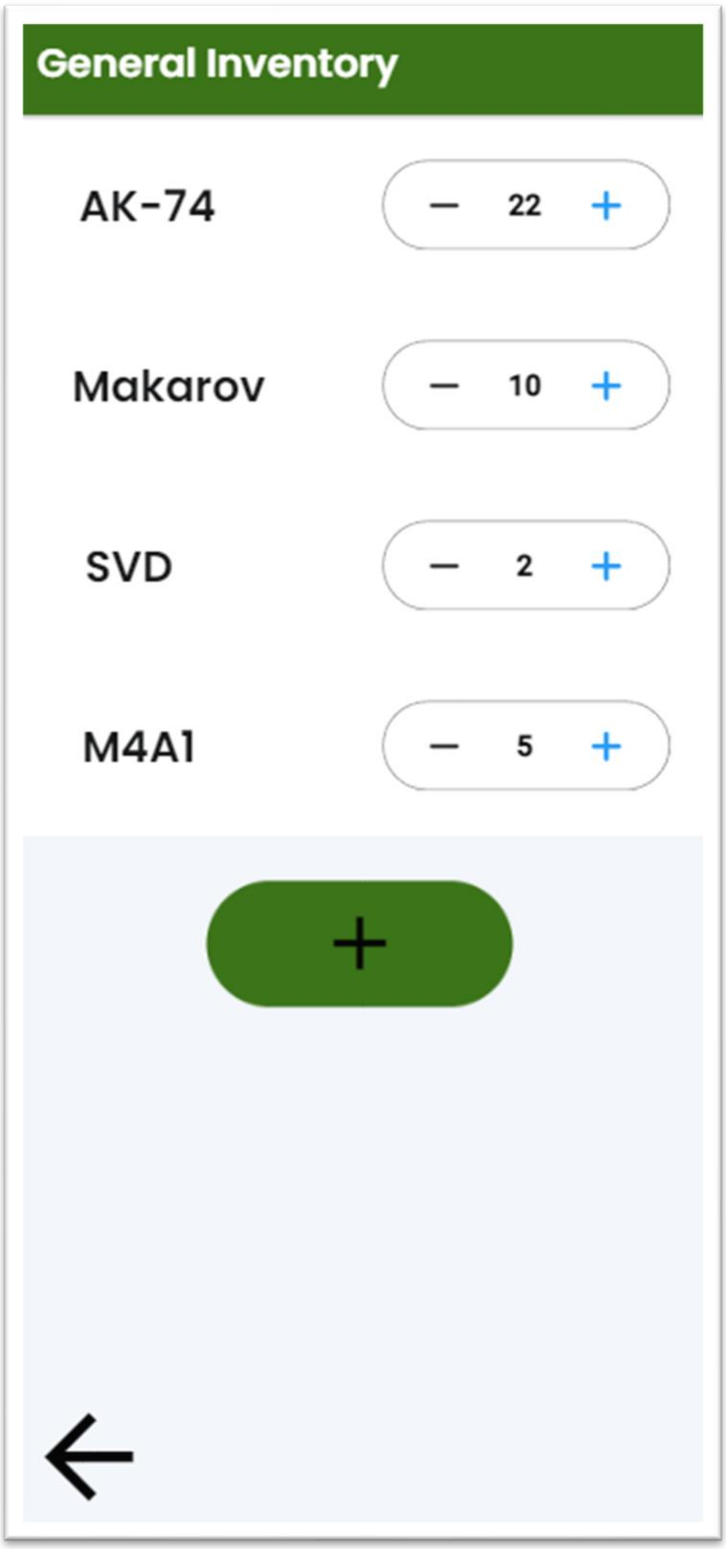


*Figure 58 Old General Inventory*