TecTags: An Automated Object Counting Mobile Application Using Object Detection Algorithms

Jose A. Perez Jr., Joshua Martin A. Peralta, Arvin F. Eugenio, and Armand Sebastian E. Bueno

College of Computer Studies, Technological Institute of the Philippines - Quezon City

IT 400: Capstone Project 2

Dr. Risty M. Acerado

May 2025

Table of Contents

List of Figures

List of Tables

Abstract

Hardware stores often deal with the hassle of manual inventory tracking, which can lead to mistakes and slow things down. This study introduces TecTags, a mobile app designed to make inventory management easier by automatically counting objects in photos taken with a smartphone. Our goal was to create and test a tool that streamlines inventory processes, especially for construction-related businesses. Built with PyTorch for advanced object detection, enhanced by Roboflow for better data handling, and developed using Flutter for seamless use across platforms, TecTags was put through its paces on Android devices. We evaluated it based on ISO 25010 standards, checking its functionality, usability, reliability, and efficiency. Results indicate that TecTags significantly enhances inventory accuracy and improves operational workflow compared to traditional manual methods. CGG Marketing Staff gave the highest ratings (5.45 average), indicating strong real-world performance. Construction Professionals also rated it highly (5.03 average), confirming its effectiveness in construction settings. IT Professionals provided a solid 4.43 average, suggesting good performance but room for technical enhancements. Customers rated it lowest at 4.2, likely due to their academic perspective and less practical engagement. This study shows how mobile object detection can revolutionize inventory management in the construction world. TecTags gives hardware store managers a powerful way to cut errors, streamline processes, and boost overall business performance.

Keywords: PyTorch, Flutter, construction industry, object detection, mobile application

Chapter 1

Introduction

As we move towards technology and digitalization, it is concerning to see construction companies using outdated systems such as complicated log sheets and spreadsheets to track inventories. An outdated inventory system can waste a significant amount of time and company physical resources that could affect the productivity of the hardware stores, particularly in the Construction Industry.

Technology has become a tool for productivity, particularly in the construction industry. Construction workers and staff can now check their data and lower their errors in the hardware stores. According to Asite (2022), over 90% of construction workers use cell phones daily for their tasks; mobile applications assist with every stage of a construction project, particularly back-office activities of hardware stores.

The use of mobile technology in the Philippines construction has improved project management and communication among industry professionals. A study presented at the IEOM Conference (Alviar et al., 2024) shows that mobile platforms play a crucial role in enhancing collaboration through live communication and social networks. The findings suggest that these digital tools are particularly valuable in ensuring efficient project delivery, especially during times of crisis, where seamless coordination is essential.

The researchers of our project, Tetags, want to enhance inventory management for construction companies in the Philippines by developing a counting and number-tagging application that uses object detection. The Tectags application makes the counting process of

construction materials easier. Image data can be stored on the user's device and accessible by cloud storage. The system allows live updates, data synchronization, and inventory tracking.

Background of the Study

Inventory management is crucial for construction projects but often challenging and prone to errors. According to Jenkins (2022), construction companies have issues such as inconsistent tracking methods, inaccurate data, oversupply, and poor use of warehouse space. These problems can lead to project delays and resource mishandling. Counting objects in a pile and objects of different sizes can cause problems on a job. These errors can result in failed accuracy in inventory counting and delays in assigned tasks.

Recent studies in artificial intelligence (AI) have shown computer vision inventory tracking improves efficiency in the construction industry. According to a (2025) article by ScienceSoft, AI can process images of storage areas to detect and count materials, identify misplaced items, and monitor stock levels in real-time. This minimizes inventory counting errors, ensuring proper stock gets replenished.

In the Philippines, the construction industry still relies on manual counting when receiving inventories from their suppliers. Supplies such as logs, angle bars, and other construction materials are hard to count manually. This process increases the chances of human error and delays in finishing the project, showing the need for an easier and more automated system. Recent advancements in artificial intelligence have improved inventory management within the construction industry (Kyanon Digital, 2025). AI object detection has enhanced the accuracy of tracking materials on construction sites. AI systems can see and count construction materials from images, allowing data accuracy of inventory records and lowering errors. AI systems can also keep an eye on equipment for indications of overheating, enabling repair thus, new improvements simplify maintenance processes.

This study aims to address the challenges faced by staff and owners of hardware stores while contributing to the advancement of the construction industry through Technological Innovations.

Project Objectives

General Objective

The study's primary objective is to develop an Android-based mobile application for the Construction Industry that can accurately perform automatic object counting and number tagging from loaded images taken from a mobile phone camera.

Specific Objectives

- Develop a mobile application using object detection algorithms to count automatically and number tagging from captured images.
- Create a mobile app that uses an Object Detection algorithm to count stocks and includes the detected objects in the inventory.
- Evaluate the mobile application using ISO 25010 regarding functionality, reliability, usability, and efficiency.

Significance of the Study

The purpose of the development of the application is to provide a convenient solution to the problem of hardware stores in counting objects. The result of this study can benefit the following:

To Hardware Staff - This mobile application can significantly reduce the time and effort required for manual inventory counting, allowing hardware staff to focus on other essential tasks. By automating the object counting process, the app ensures faster, more accurate counts and minimizes human errors, leading to a more streamlined workflow. Staff no longer need to spend long hours manually checking inventory, thus enabling them to allocate their time to tasks such as assisting customers, restocking, or conducting maintenance. This leads to a more efficient use of time and resources within the hardware store, improving both staff productivity and job satisfaction.

To the Hardware Stores - Hardware stores often face the challenge of keeping track of large volumes of diverse inventory. With a wide range of materials such as nails, lumber, steel, and various construction items, it becomes difficult to efficiently count and manage stock, especially when there are numerous similar items. This manual counting process is labor-intensive, time-consuming, and prone to human error, leading to discrepancies in inventory records. By implementing this mobile application, stores can automate the counting of objects, ensuring accurate and real-time stock tracking. This will help maximize warehouse space, reduce the risk of overstocking or stockouts, and improve inventory accuracy. The application will streamline daily operations, making it easier to manage large inventories and ultimately helping hardware stores operate more efficiently, minimizing errors and improving overall productivity.

To other Researchers - This study can serve as a reference for researchers looking into developing mobile applications for inventory management and object detection in retail environments. It offers insights into integrating AI-powered technologies like PyTorch for object counting and number tagging, specifically tailored to the needs of the hardware industry. Future researchers can use this as a foundational framework to explore improvements in object

detection algorithms, expand the application's capabilities to other industries, or even enhance the user experience by adding advanced features. Moreover, the success of this study can inspire new research in similar domains, paving the way for innovation in AI-based applications for inventory management and logistics.

Scopes and Delimitations

Scope

1. Object Detection Module

- Implement object detection techniques for automated inventory counting for hardware industries.
- Providing an option to adjust detection sensitivity to refine object recognition.
- Optimizing detection accuracy by recommending that objects be placed separately on a flat surface without overlapping.

2. Number Tagging Module

- Ensuring the accuracy of number tags through object detection algorithms.
- Implementing number tagging for each detected object in the uploaded image.
- Enabling users to add or remove number tags for objects counted by the object detection algorithm.
- Allowing customization of number tags, including dragging tags to the correct object and resizing for better visibility.

3. Image Processing & Saving Module

- Automatically generating image labels, including title, date, time, number count, and tags, and saving processed images on the user's smartphone.

4. Cloud Storage & Sharing Module

- Using MongoDB as cloud storage to enable users to share processed images.

Delimitations

- The application will be exclusively developed for Android devices, explicitly targeting versions 5.0 (Lollipop) and higher, limiting its accessibility to other operating systems.
- The application does not include comprehensive inventory management features. While it aids in counting items, it does not manage stock levels or sales transactions.
- Users must maintain an active Internet connection to send, retrieve, update, and save data from a cloud-based database, which may limit usability in areas with poor connectivity.
- Access to the application is restricted to users in the Philippines, ensuring that the content and functionality are tailored to local hardware store needs.
- The application is designed to count the five most sold items in hardware stores: hollow blocks, deform bars, bistay sand, common nails, gravel, and skim coats. Other items will not be considered within the scope of this project.

Chapter 2

Theoretical Framework

A systematic identification of documents with identical information connected to the research topic was conducted. This study was supported by presenting various journals, publications, and articles from both local and foreign settings to determine what approach or techniques should be implemented in the development of the application.

Review of Related Literature

This chapter examines recent literature and studies that is relevant to Tectags app, focusing on object detection algorithms, computer vision tools, mobile development frameworks, and dataset annotation techniques. The review establishes a theoretical framework, highlights current advancements the study aims to address.

Object Detection Algorithms and Computer Vision Tools

OPENCV

OpenCV (Open Source Computer Vision Library) has become a vital tool for computer vision applications. It provides strong functionalities for image processing and object detection. A recent study by Ghosh (2024) demonstrated the integration of YOLOv8 with OpenCV,

emphasizing improvements in object tracking and counting. This integration controls OpenCV's efficient image handling and YOLOv8's advanced detection algorithms, enabling live applications, such as those on mobile devices.

According to Sharma et al. (2021), OpenCV is a handy tool for beginners who wish to learn how real-time object identification and tracking can be done. It also shows the flexibility of a tracking system compared to a moving camera, which is ideal for moving safety applications. Image identification makes use of techniques like the detection of an object, its recognition, and separation.

OpenCV is limited in terms of object counting, needing to manipulate the given image first. OpenCv is reliant on manipulating the image itself to isolate the object that needs to be counted; for example, it needs to do background subtraction to isolate the foreground objects (the ones that are counted) so that these foreground objects would be detected. The problem is when you do background subtraction, the algorithm that removes the background sometimes gets affected by lighting issues (the image is too dark or there is excessive lighting). If there are lighting issues, sometimes the foreground objects do not have a chance of being included in the count because the background subtraction algorithm can unintentionally remove foreground objects that are supposed to be counted.

YOLO

According to Solawetz (2025), The YOLO (You Only Look Once) series of models has become famous in the computer vision world. YOLO's fame is attributable to its considerable accuracy while maintaining a small model size. YOLO models can be trained on a single GPU, which makes it accessible to a wide range of developers. Machine learning practitioners can deploy it for low-cost ed,ge hardware, or in the cloud.

Another study by Jurnal. iii.or.id (2024) explored the enhancement of YOLOv8 with OpenCV for object detection, achieving an error rate as low as 3.15% and validating the combination's potential to increase detection accuracy.

The field of object detection has seen significant progress, with deep learning-based approaches dominating recent research. YOLO (You Only Look Once) remains a leading algorithm due to its real-time capabilities. Wang et al. (2022) introduced YOLOv7, optimizing speed and accuracy for edge devices, achieving up to 70 FPS on standard hardware with a mean average precision (mAP) of 51.2% on the COCO dataset. The subsequent release of YOLOv8 by Ultralytics (2023) further improved performance, incorporating self-attention mechanisms to enhance small object detection, a persistent challenge in single-stage detectors.

As we continued researching, we found YOLO and tried using it as our object detection algorithm. YOLO, at first, was doing great on object detection, but we encountered challenges with Overlapping Objects that made it difficult to distinguish them as separate entities.

When we are trying to use the model that ONNX (best to be deployed on the web, but when you deploy it, there are no free options to keep it running.) produces, it requires normalization, but we cannot get the correct values. The reason we used Yolo is because we used React Native; React Native is not compatible with TensorFlow Lite out of the box, so we found an alternative Yolo; we trained and produced a model using Ultralytics YOLO11, but that model produced does not output the correct coordinates that would be drawn on top of the detected objects. The output is either out of bounds or too small/tiny compared to the resolution of the image being analyzed.

TENSORFLOW

According to Chen et al. (2024), The application of TensorFlow pre-trained models in deep learning is explored, with an emphasis on practical guidance for tasks such as image classification and object detection. The study covers modern architectures, including ResNet, MobileNet, and EfficientNet, and demonstrates the effectiveness of transfer learning through real-world examples and experiments.

- ResNet revolutionized deep CNN training by enabling the creation of very deep and accurate networks through residual connections.
- MobileNet focused on efficiency for mobile and embedded devices by using depthwise separable convolutions and architectural optimizations.
- EfficientNet provided a more systematic approach to scaling CNNs by using compound scaling to achieve better accuracy and efficiency trade-offs.

A comparison of linear probing and model fine-tuning is presented, supplemented by visualizations using techniques like PCA, t-SNE, and UMAP (these are three powerful dimensionality reduction techniques with different strengths and weaknesses), allowing for an intuitive understanding of the impact of these approaches. The work provides complete example code and step-by-step instructions, offering valuable insights for both beginners and advanced users. By integrating theoretical concepts with hands-on practice, the paper equips readers with the tools necessary to address deep learning challenges efficiently.

Tensorflow has a pre-trained model that is already effective at detecting objects; if you still want to enhance it, Tensorflow has transfer learning, making it effective compared to OpenCV, which requires Photoshop to see foreground objects. It is not limited to the technicalities of background subtraction; including a lot of datasets by training increases its accuracy. Tensorflow can also be accessed offline without needing an internet connection. Upon

further research we encountered that Tensorflow has issues on conflicting dependency and outdated making our application TegTags unable to train using the images that we manually annotated from the CGG Marketing.

TENSORFLOW LITE

According to (TensorFlow, 2021), TensorFlow Lite has been used for object detection because of its effectiveness. Their studies say that TensorFlow Lite provides a balance between accuracy and processing speed. The study, however, is focused on object detection without integrating number tagging for applications.

A tutorial by EdjeElectronics (2021) shows the use of TensorFlow Lite models on Android devices, pointing out the importance of efficient storage and restoration of processed images. The tutorial shows the use of cloud storage solutions to maintain the security and accessibility of processed data.

According to Alqahtani et al. (2024), Deep learning frameworks play a crucial role in developing and deploying object detection models by providing comprehensive tools and functionalities. These frameworks facilitate building, training, and implementing object detection systems with pre-built models, data augmentation techniques, and utilities for various development stages. Examples include TensorFlow Lite, PyTorch, and TensorRT. They enable researchers and developers to efficiently manage the complexities of object detection tasks, from data preprocessing and model training to evaluation and deployment.

A guide by Neptune.ai (2025) details the process of training a custom object detector using the TensorFlow Object Detection API. It covers key steps such as data collection, annotation, model selection, training, and evaluation. The study highlights how preprocessing techniques, such as image resizing and augmentation, help improve the model's ability to

recognize objects in various conditions. However, the guide primarily focuses on model training rather than real-time application on mobile devices.

- EfficientDet is an object detection model designed for speed and efficiency.
- R-CNN is a two-stage object detection model that prioritizes accuracy over speed.

This shows the importance of selecting and optimizing models to ensure successful deployment. The study is focused on general model conversion.

Upon further research and testing, we found that TensorFlow Lite is not the most optimal solution for TecTags due to several limitations. While TensorFlow Lite is designed for deploying models on mobile devices, it lacks full support for advanced object detection features and often requires manual conversion of models, which can lead to compatibility issues or reduced accuracy after quantization.

PYTORCH

According to Matani (2022), PyTorch is a deep learning framework for training and running machine learning (ML) models, accelerating the speed from research to production. Typically, a model is trained (either on a CPU or GPU) on a powerful server and then deployed on a mobile platform, which is typically more resource-constrained. The article focuses on the support PyTorch provides for running server-trained models on a mobile device or platform.

TecTags also uses a model trained with PyTorch, which is later deployed to smartphones. By using PyTorch's mobile deployment features, TecTags ensures efficient object detection and counting on devices, making the application faster, lighter, and more accessible for its intended users.

Deploying machine learning models to mobile devices has become increasingly essential as applications demand real-time, on-device intelligence. According to Sling Academy (2024),

deploying PyTorch models to iOS and Android involves converting models into mobile-friendly formats and integrating them into native applications. This process allows applications to operate in real time, providing instant feedback and actions. By exporting, optimizing, and integrating PyTorch models into Android and iOS platforms, developers can significantly enhance the performance and functionality of their mobile solutions.

The techniques in the article will serve as a valuable reference for future improvements. By understanding how PyTorch models are optimized and integrated into mobile platforms, TecTags researchers can potentially enhance the app's performance in future versions, allowing real-time object detection on mobile devices.

According to Mishchenko (2024), This work is devoted to the development of a neural network for the classification of satellite images of the Earth in four classes: water, sand, clouds and green terrain (forests, fields, plant accumulation). To do this, the Python library PyTorch is used, focused on the application of deep learning for computer vision tasks. The proposed approach allows you to automatically classify satellite images, facilitating the analysis of large amounts of data. It was tested on satellite images Sentinel-2, obtained from March to August 2023 for some parts of Kherson and Mykolaiv regions, Ukraine, and 95% accuracy was obtained. The results of the study are useful for specialists in the field of remote sensing of the Earth, computer vision, machine learning, and software development for the analysis of aerospace images.

By adopting the same principles, TecTags can use PyTorch's deep learning capabilities for accurate object detection and number tagging in real-world hardware inventory images. The high accuracy achieved in satellite image classification shows the reliability of PyTorch for visual

recognition tasks, suggesting that TecTags can similarly optimize its performance in identifying, tagging, and counting hardware objects from user-uploaded images.

According to Kirchheim (2022), Machine Learning models based on Deep Neural Networks behave unpredictably when presented with inputs that do not stem from the training distribution and sometimes make egregiously wrong predictions with high confidence. This property undermines the trustworthiness of systems depending on such models and potentially threatens the safety of their users. Out-of-Distribution (OOD) detection mechanisms can be used to prevent errors by detecting inputs that are so dissimilar from the training set that the model can not be expected to make reliable predictions. In this paper, we present PyTorch-OOD, a Python library for OOD detection based on PyTorch. Its primary goals are to accelerate OOD detection research and improve the reproducibility and comparability of experiments. PyTorch-OOD provides well-tested and documented implementations of OOD detection methods with a unified interface, as well as training and benchmark datasets, architectures, pre-trained models, and utility functions. The library is available online under the permissive Apache 2.0 license and can be installed via Python Package Index (PyPI).

Tectags will want to use PyTorch for detecting, counting and tagging hardware objects. The PyTorch-OOD library presented in the study offers a valuable solution by providing tools to detect such unfamiliar inputs and prevent false predictions. Integrating OOD detection into TecTags can enhance its reliability by flagging images that significantly deviate from its training data, such as cluttered backgrounds, blurry uploads, or non-hardware items. PyTorch's flexibility, strong community support, and its alignment with cutting-edge research in computer vision make it a more future-proof and research-backed choice.

ROBOFLOW

According to Villarroel et al. (2025), This system shows the ability to accurately identify and classify dress codes, offering improvements. Comparative analysis emphasizes its superiority in terms of precision and reliability, making it a practical solution for schools aiming to streamline dress code enforcement. Future work of this study will explore the inclusion of extensive datasets and integration with emerging object detection technologies to refine accuracy further and expand applicability. This study aims to utilize AI in educational policy enforcement, contributing to a structured campus environment.

We are currently using Roboflow for our annotation of each picture we take at hardware stores. It is helpful as it allows us to train manually and make our application accurate. Roboflow also offers a lot of free datasets for us to use and train for anything hardware-related moving objects.

Mobile Development Frameworks

FLUTTER

Choosing a proper tech stack for a project is one of the first critical points at the very start. Facebook's React Native and Google's Flutter have gathered widespread attention as development frameworks for their capabilities and features.

According to Valis (2025), Statistics show that Flutter is the top cross-platform mobile framework used by software developers, with the usage rate amounting to 46%. React Native comes second, with the rate equal to 32%. These two technologies comprise the majority of their niche. That's why it doesn't come as a surprise that many are looking for a justified comparison of Flutter and React Native.

Both of these frameworks have pros and cons. We tried using React Native, but we encountered a problem with package compatibility between different React Native versions and

third-party libraries. It also has an extensive application size requiring optimization. However, on Flutter, we had an easy time learning and understanding as we could see the code changes instantly without restarting the app, accelerating development, experimentation, and bug fixing.

MONGODB

According to C.A. Győrödi, Dumşe-Burescu, Zmaranda, and R Győrödi (2022), MongoDB is the most popular type of NoSQL database, with a continuous and secure rise in popularity since its launch. It is a cross-platform, open-source NoSQL database that is document-based (written in C++), completely schema-free, and manages JSON-style documents. Improvements to each version and its flexible structure, which can often change during its development, provide automatic scaling with high performance and availability. The document-based MySQL is not so popular yet, with MySQL providing a solution for non-relational databases only since 2018, starting with version 8.0, which has several similarities and differences regarding the model approach to MongoDB.

Our object detection application utilizes MongoDB as its versatile backend database, expertly handling both our evolving inventory of detectable objects and our detailed application logs. MongoDB's flexible document structure allows us to store rich metadata associated with each object class, including variations in attributes and training data details, facilitating easy updates and queries as our detection capabilities expand. Concurrently, we leverage MongoDB's high write throughput to efficiently capture and organize the application's log data, recording detection events, processing times, and system activities, providing an invaluable resource for performance analysis, debugging, and understanding the application's operational behavior in real time. This unified approach with MongoDB ensures a scalable and adaptable data layer capable of supporting the dynamic nature of our object detection application.

REACT NATIVE

According to Goli (2021), React Native, introduced by Facebook in 2015, aims to streamline the mobile app development process by providing a platform that enables developers

to write a single codebase in JavaScript and React. This framework bridges the gap between web and mobile development, enabling JavaScript developers to transition into mobile app development easily. React Native's promise is that apps built with it can run on both iOS and Android while providing a user experience close to native apps meaning users experience the performance, speed, and feel of a native app despite the app being written in JavaScript. The framework uses native components instead of web components, ensuring the user experience is optimized for each platform while allowing developers to manage the app's logic and interface from a unified codebase.

In the initial iterations of our application, we explored leveraging the robust image processing capabilities of OpenCV directly within our React Native codebase. This involved bridging to native OpenCV libraries to implement functionalities. While this approach granted us access to a robust suite of computer vision algorithms, integrating and maintaining these native dependencies within our cross-platform React Native environment presented complexities and potential performance considerations across different mobile operating systems.

Dataset Annotation Techniques

IMAGE ANNOTATION

According to Bandyopadhyay (2024), image annotation is labeling images in a given dataset to train machine learning models. Image annotation sets the standards, which the model tries to copy, so any label error is replicated, too. Therefore, precise image annotation lays the foundation for neural networks to be trained, making annotation one of the most critical tasks in computer vision. Given the vast variety of image annotation tasks and storage formats, various tools can be used for annotations, from open-source platforms, such as CVAT and Labeling for simple annotations to more sophisticated tools like V7 for annotating large-scale data.

In developing the Tectags application, image annotation is essential in training machine learning models to recognize and tag images accurately. Using Roboflow's robust annotation tools, Tectags can efficiently label images with bounding boxes or polygons, facilitating precise object detection. Roboflow's AI-assisted labeling feature, Label Assist, further streamlines this process by offering automated annotation suggestions, significantly reducing manual labeling efforts. Once the dataset is annotated, Tectags can leverage Roboflow's seamless integration to train custom models, ensuring that the application accurately identifies and tags images based on the labeled data.

Comparative Analysis

Tool	Primary Use	Strengths	Limitations	TecTags
TecTags	Automated object detection, counting, and tagging	AI-powered counting, number tagging, optimized for inventory	Still in development; focused on hardware/construction use cases	Designed for the hardware industry, automates object counting and tagging from photos
Stocktake	Manual inventory management	Inventory tracking and stock alerts	Manual entry; no image-based detection	TecTags automates the counting process using images, reducing manual work

CamScanner	Document scanning and OCR	Scanning, PDF creation, text recognition	No object detection or inventory-specific tools	TecTags analyzes objects and provides count-based inventory management
Square for Retail	POS and inventory for retail	Integrated with sales systems, suited for stores	Manual input; limited for warehouse or construction inventory	TecTags supports backend inventory with visual object detection for materials
Sortly	Visual inventory with barcode/QR code	Easy to organize visually, mobile-friendly	Tagging is manual; no automatic object recognition	TecTags automates object identification and tagging from smartphone images
Google Lens	General image recognition	Strong AI object detection and info retrieval	Not tailored for inventory; no tagging/counting features	TecTags is inventory-specific and outputs count and tags for hardware use cases

TecTags uniquely combines object detection, tagging, and counting in a single mobile app. It is tailored for hardware and construction industry needs, offering a lightweight solution between manual inventory tracking and AI-powered tool that helps users count objects from photos.

The comparative analysis highlights the strengths and limitations of TecTags compared to other popular inventory and object detection tools: Stocktake, CamScanner, Square for Retail, Sortly, and Google Lens. TecTags is designed for hardware and construction-related inventory management unlike general-purpose apps. It integrates object detection, counting, and tagging using PyTorch, enabling users to automate and streamline stock verification processes directly from their mobile phones.

While apps like CamScanner and Google Lens excel in document scanning and broad object recognition, they lack specialized inventory features. Sortly and Square for Retail offer robust inventory tracking and organization but often require manual data entry or barcode scanning. Stocktake is inventory-focused.

TecTags stands out by combining object detection, automated counting, and inventory tracking into one dedicated solution for the hardware industry, reducing error and improving efficiency.

Conceptual Framework

To provide an understanding of the system, the researchers developed a conceptual framework on how TecTags will be implemented. This framework was designed to address the ongoing challenge of manually counting construction materials and other inventory items, a common issue in hardware-related operations. By presenting the system through the Input–Process–Output (IPO) model, this framework shows how the application integrates relevant technologies to streamline inventory tracking and improve operational efficiency.

Figure 1:

Input–Process–Output (IPO) Diagram of the TecTags System

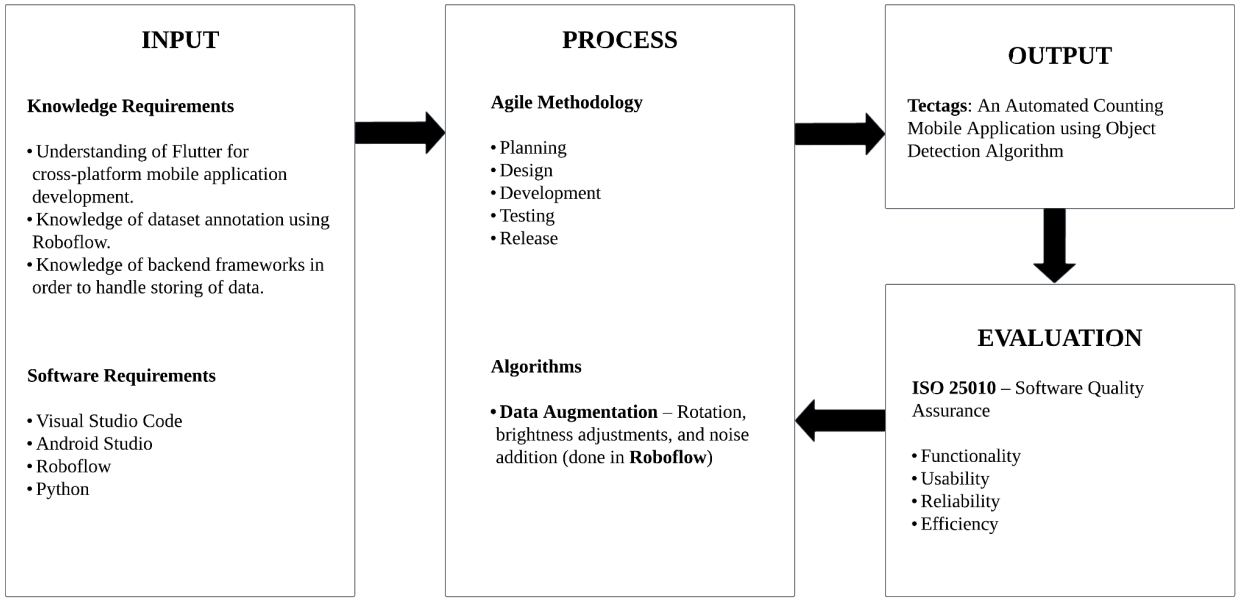


Figure 1 is the representation of the Conceptual Framework that shows the Input, Process, Output, and Evaluation of TecTags, an automated object-counting mobile application. The Input phase focuses on understanding Flutter and Android mobile development to create a responsive mobile application. Dataset annotation using Roboflow is needed to make sure it is accurate object detection, and it will improve image recognition. Software tools are utilized to develop TecTags. Visual Studio Code serves as the Integrated Development Environment (IDE) for coding, while Android Studio is used for testing and ensuring compatibility with mobile platforms. Visual Studio Code is used for training object detection, while Roboflow assists in dataset annotation. The programming language used is Python, which plays an important role in machine learning and computer vision functions in the system.

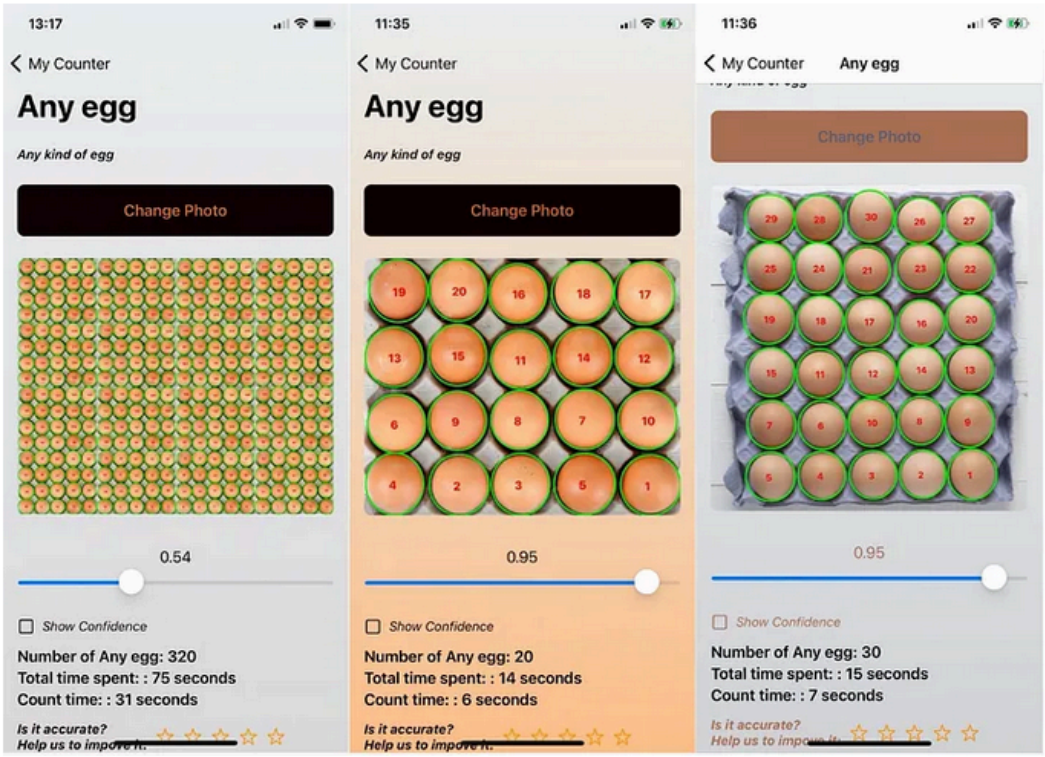
The Process follows the Agile Methodology, which consists of planning, design, development, testing, and release. Flutter is chosen due to its ability to create cross-platform applications with a seamless user interface. Several techniques are used to improve the object detection capabilities of TecTags. Roboflow is used for data augmentation, applying techniques such as rotation to enhance model accuracy. Flutter is integrated to develop the mobile application interface, providing a real-time, responsive, and interactive user experience.

The Output is TecTags, a mobile application capable of automated object counting and number tagging. The application can assist hardware store owners and employees by simplifying inventory management, reducing errors, and improving efficiency. By automating the object counting process, TecTags lowers the challenges of manual labor and makes sure a more reliable approach.

The Evaluation phase ensures the software quality of TecTags through the ISO 25010 standards, focusing on functionality, usability, reliability, and efficiency. Functionality is assessed based on the accuracy of object detection and counting, ensuring the system meets its intended purpose. Usability and reliability are evaluated to confirm that the application provides a user-friendly experience for hardware store staff. Efficiency measures the system's ability to process inventory data quickly and accurately, ensuring optimal performance. By adhering to these evaluation criteria, TecTags delivers an efficient and practical solution for inventory management in hardware stores.

Figure 2:

An examples of object counting on an image using an object detection algorithm



System Architecture

Figure 3:

System Architecture of TecTags

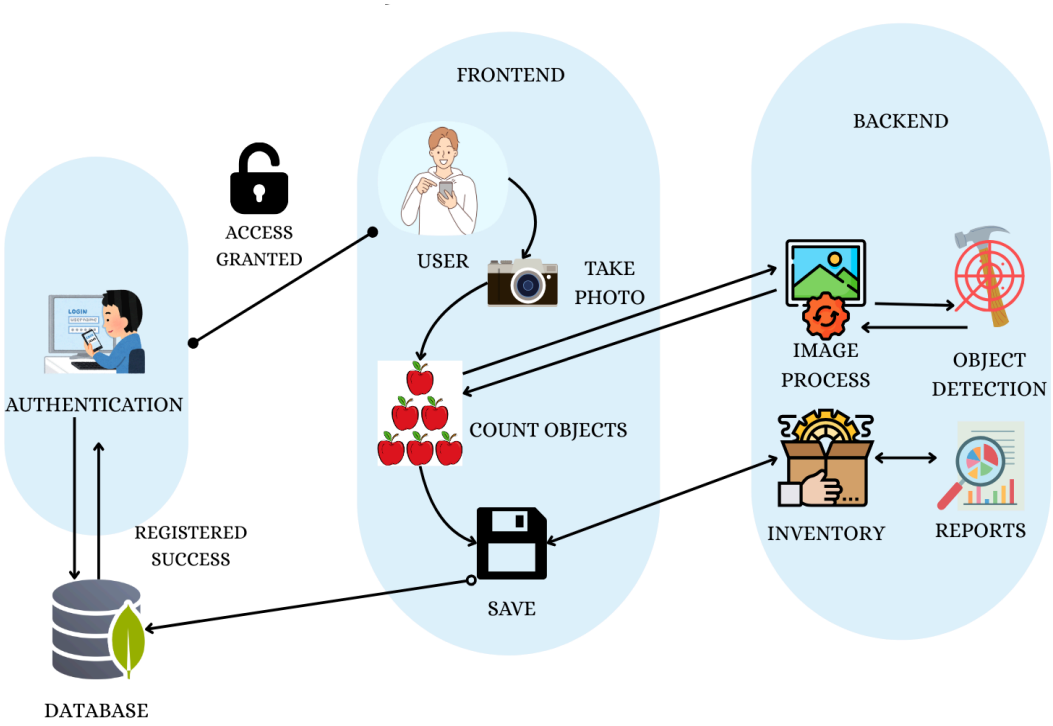


Figure 3 illustrates the overall system architecture of the TecTags mobile application, showcasing the flow of processes across the Authentication, Frontend, and Backend components.

Object Detection Algorithm

We use the algorithm to detect images, classifying them by location, and labeling them automatically. This process involves using a camera to picture a certain item/living thing to be detected regardless of their numbers and angles. To do this process, you need lots of images to be annotated and trained to be able to identify and tag how many are inside the images.

Figure 4:

Object Detection Example

Definition of Terms

- **API (Application Programming Interface)** – A set of rules that allows different software components to communicate.
- **Back End** – The part of the system that handles data processing, logic, and communication with the database. It operates behind the scenes and ensures that the application functions correctly.
- **Express** – A lightweight Node.js framework used to build APIs for managing data operations such as creating, reading, updating, and deleting object count records.
- **Flutter** – A UI toolkit from Google for developing cross-platform applications from a single codebase. It is used in this project to build the mobile interface of TecTags.

- **Front End** – The user-facing part of the application that includes the design and interactive elements such as buttons, forms, and object count displays.
- **ISO 25010** – An international standard that defines software product quality metrics such as functionality, performance, usability, and maintainability. It is used as a framework to evaluate the quality of the TecTags application.
- **MongoDB** – An open-source NoSQL database that stores information in a flexible, JSON-like format (BSON). It is used in this project to store object counts and user-related data.
- **Node.js** – A server-side JavaScript runtime environment used with Express to build the backend API for TecTags.
- **Number Tagging** – The process of assigning numeric labels to detected objects in an image to help users track and differentiate multiple items during inventory management.
- **OpenCV (Open Source Computer Vision Library)** – A computer vision library for real-time image processing tasks such as filtering, contour detection, and edge detection.
- **Python** – A programming language commonly used in AI and machine learning. It is utilized in this study for training the object detection model with PyTorch and preprocessing images.
- **PyTorch** – is an open source machine learning (ML) framework based on the Python programming language and the Torch library. Torch is an open source ML library used for creating deep neural networks and is written in the Lua scripting language. It's one of the preferred platforms for deep learning research. The framework is built to speed up the process between research prototyping and deployment.

- **Roboflow** – A platform that helps in image annotation, augmentation, and dataset preparation. It simplifies the creation of datasets for training object detection models like YOLO, TensorFlow Lite and PyTorch.
- **TensorFlow** – An open-source machine learning framework by Google used for training AI models. In this project, it is used to develop and optimize object detection models before converting them into mobile-ready formats.
- **TensorFlow Lit**e – A lighter version of TensorFlow designed to run efficiently on mobile and edge devices.
- **UI (User Interface)** – The design and layout of the application that users interact with, including buttons, text boxes, and object tagging tools. Flutter is used to build the UI for TecTags to ensure smooth and intuitive navigation.
- **YOLO (You Only Look Once)** – A real-time object detection algorithm that divides images into regions and predicts bounding boxes and class probabilities. It is known for its speed and accuracy and can be used during the model training phase for object detection tasks in hardware environments.

Chapter 3

Research Methodology

This chapter outlines the methods used to develop the project, covering key areas such as Project Design, Design Discussion, Developmental Research, Project Development, Testing and Operating Procedures, Project Evaluation, and the Work Plan. The researchers establish methods for selecting the relevant procedures to fulfill the project requirements. The project utilizes agile methodology to implement the system's functional flow effectively.

Project Design

The researchers developed TecTags, a mobile application designed to assist hardware stores in automating inventory tracking and object detection. The development process integrated both front-end and back-end technologies to ensure an efficient and user-friendly experience. The user interface was initially designed using Figma, a design tool that enabled the team to plan layouts, buttons, and icons. The mobile frontend was developed using Flutter, an open-source framework that allowed the application to be built for Android devices. Flutter enabled prototyping and cross-platform capabilities while ensuring the final design adhered closely to the Figma prototypes. For the backend and object detection module, the team used Python along with the PyTorch library. PyTorch was selected for its flexibility, performance, and wide support for deep learning and computer vision tasks. Object detection models were trained using PyTorch and then integrated into the mobile application to enable real-time object identification and counting. The entire application was built and managed in Visual Studio Code, which served as the primary development environment. Testing and debugging were conducted using BlueStacks, an Android emulator that allowed for testing APK builds in a simulated device environment without relying on physical Android hardware. This development approach resulted in an

efficient application that combined UI design, powerful machine learning capabilities, and reliable mobile performance.

Design Discussion

The TecTags application was developed with an intuitive interface for hardware stores. The design simplicity through a minimalistic approach ensures users can navigate the app's features. The design began with Figma, where prototypes were created to visualize the flow and layout of each screen. These designs were then implemented using Flutter.

Interface designs were developed:

- **Interface Design 1** introduced the splash screen, which appeared immediately after launching the app. It featured the TecTags logo and a loading bar.
- **Interface Design 2** displayed the login screen, where users could enter their credentials or choose to sign up.
- **Interface Design 3** showcased the registration page, accessed through the "Create an Account" option. This design included form fields for entering details such as first name, middle name (optional), last name, email, and password, with validation.
- **Interface Design 4** represented the homepage or main dashboard that users were redirected to after a successful login. This screen provided access to the core functionalities of TecTags, including object detection, inventory view, and settings.

Each screen was tested to ensure usability across devices using Flutter's widget system and tested on both physical Android devices and emulators such as BlueStacks.

The TecTags team ensured that the application's interface met functional expectations for its target users.

Mobile Design

Figure 5:

Splash Screen

Figure 5 shows the splash screen for the mobile application. This will be displayed after the user clicks the TecTags application.

Figure 6:

Welcome Menu Screen

Figure 6 shows the Welcome menu screen module. The user can log into their account or create a new one.

Figure 7:

Login Screen

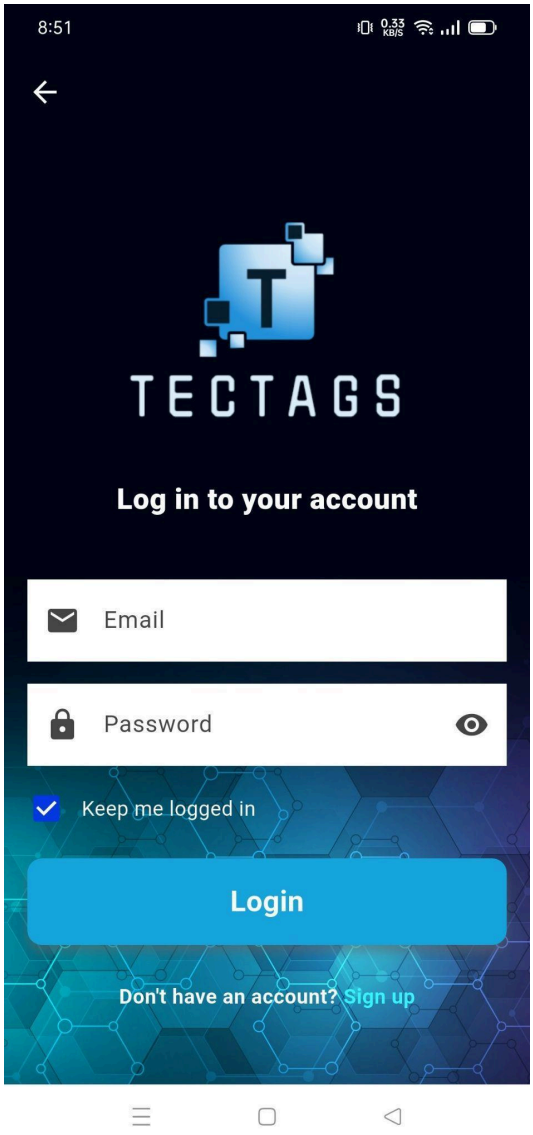
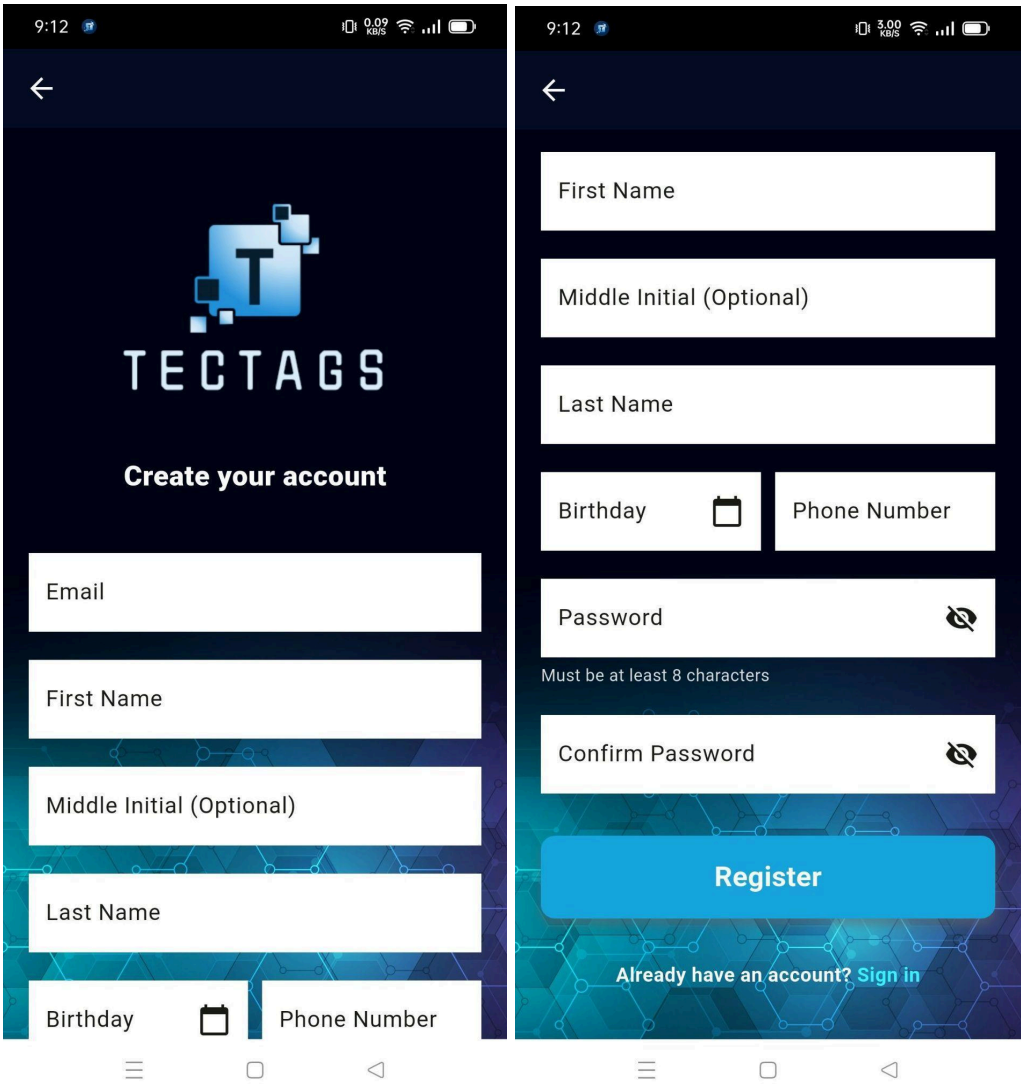


Figure 7 shows the Login screen module. The user may either manually login or automatically log in to their existing account. If the user has no account, they can create a new account using the login button.

Figure 8:

Create Account Screen



```
// PUBLIC REGISTRATION (default role only)
app.post('/api/register', async (req, res) => {
  try {
    const {
      email,
      password,
      firstName,
      middleName,
      lastName,
      contactNumber,
      birthday,
    } = req.body;

    if (
      !email ||
      !password ||
      !firstName ||
      !lastName ||
      !contactNumber ||
      !birthday
    ) {
      return res.status(400).json({ message: 'All fields are required.' });
    }

    const existingUser = await User.findOne({ email });
    if (existingUser) {
      return res.status(400).json({ message: 'Email is already in use.' });
    }

    const hashedPassword = await bcrypt.hash(password, 12);

    const newUser = new User({
      email,
      hashedPassword,
      firstName,
      middleName, // 👈 this can be undefined, which is okay in Mongoose
      lastName,
      contactNumber,
      birthday: new Date(birthday),
      role: 'employee', // 👈 Force default role
    });

    await newUser.save();

    await Activity.create({ userId: newUser._id, action: 'Registered' });

    const token = createToken(newUser._id, newUser.role);

    res.status(201).json({
      message: 'Registration successful!',
      token,
      userId: newUser._id.toString(),
    });
  } catch (error) {
    console.error('❌ Registration Error:', error);
    res
      .status(500)
      .json({ message: 'Something went wrong.', error: error.message });
  }
});
```

server.js

```
const mongoose = require('mongoose');

const userSchema = new mongoose.Schema({
  email: {
    type: String,
    required: true,
  },
  hashedPassword: {
    type: String,
    required: true,
  },
  firstName: {
    type: String,
    required: true,
  },
  middleName: { type: String }, // 👈 optional
  lastName: {
    type: String,
    required: true,
  },
  contactNumber: {
    type: String,
    required: true,
  },
  birthday: {
    type: Date,
    required: true,
  },

  // ROLE BASED ACCESS CONTROL
  role: {
    type: String,
    enum: ['employee', 'manager'],
    default: 'employee',
  },
});

module.exports = mongoose.model('User', userSchema);
```

user.js

```dart
// POST REQUEST: REGISTRATION
static Future<Map<String, dynamic>?> registerUser(
    Map<String, dynamic> userData) async {
  debugPrint("Sending request to: ${baseUrl}register");
  debugPrint("Request body: ${jsonEncode(userData)}");

  var url = Uri.parse("${baseUrl}register");

  try {
    final res = await http.post(
      url,
      headers: {"Content-Type": "application/json"},
      body: jsonEncode(userData),
    );

    debugPrint("Response Code: ${res.statusCode}");
    debugPrint("Response Body: ${res.body}");

    var data = jsonDecode(res.body.toString());

    if (res.statusCode == 201) {
      debugPrint("SUCCESS: $data");

      String userId = data['userId'];
      String token = data['token'];

      SharedPreferences prefs = await SharedPreferences.getInstance();
      await prefs.setString('userId', userId);
      await prefs.setString('token', token);

      return data;
    } else {
      String errorMessage = data['message'] ?? "Unknown error";
      return {"error": "Registration failed: $errorMessage"};
    }
  } on SocketException catch (_) {
    return {"error": "No internet connection"};
  } catch (error) {
    debugPrint("⚠️ Exception: $error");
    return {"error": "Network error: ${error.toString()}"};
  }
}
```

api.dart

Figure 8 shows the Create Account screen wherein we created a secure user registration system in an express. js app with Mongoose and to get it to work with that we have simply had to parse the data, as we are using AngularJS. After the user has submitted their details, the backend validates the input, checks whether the email address already exists, and securely hashes the password via Bcrypt. Subsequently we create a new user with its default role being "employee", and it is saved to the database. Optional fields such as the middle name are handled as well. The system will also record the registration activity and create a JWT token for handling the “Remember me” feature that automatically logs in a user if the token is present in the phone’s

shared preference. Each user is guaranteed to have the correct schema structure and role-based access control with mongoose schema.

Figure 9:

Main Screen

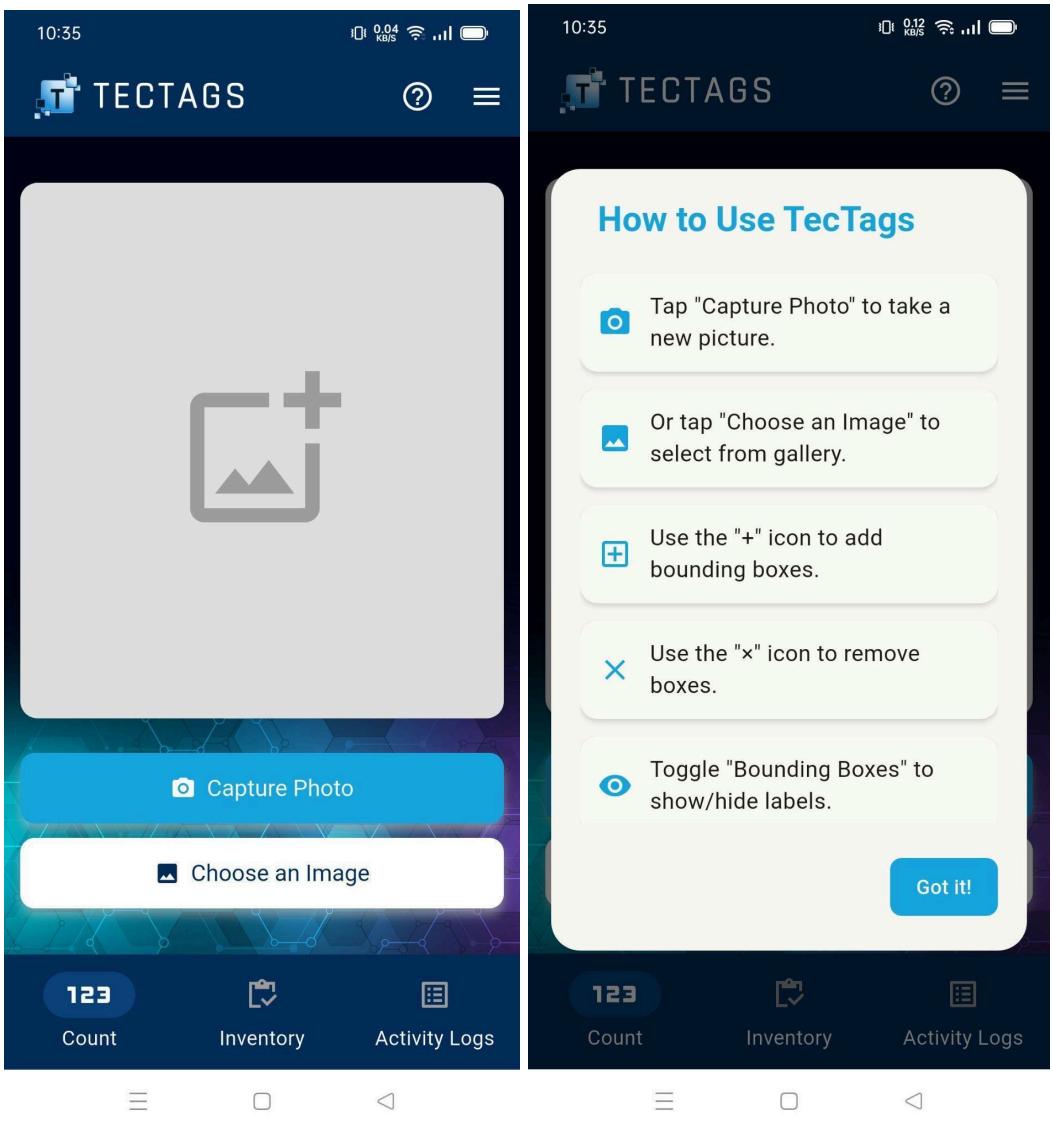


Figure 9 shows the application's Main Screen, where you can capture an image or select an image from the gallery for object detection. This screen also has a navigation drawer for the user where you can navigate to a specific screen or log out of the user's account. It also has tutorial icon that will help the user how to use the application.

Figure 10:

Inventory Screen

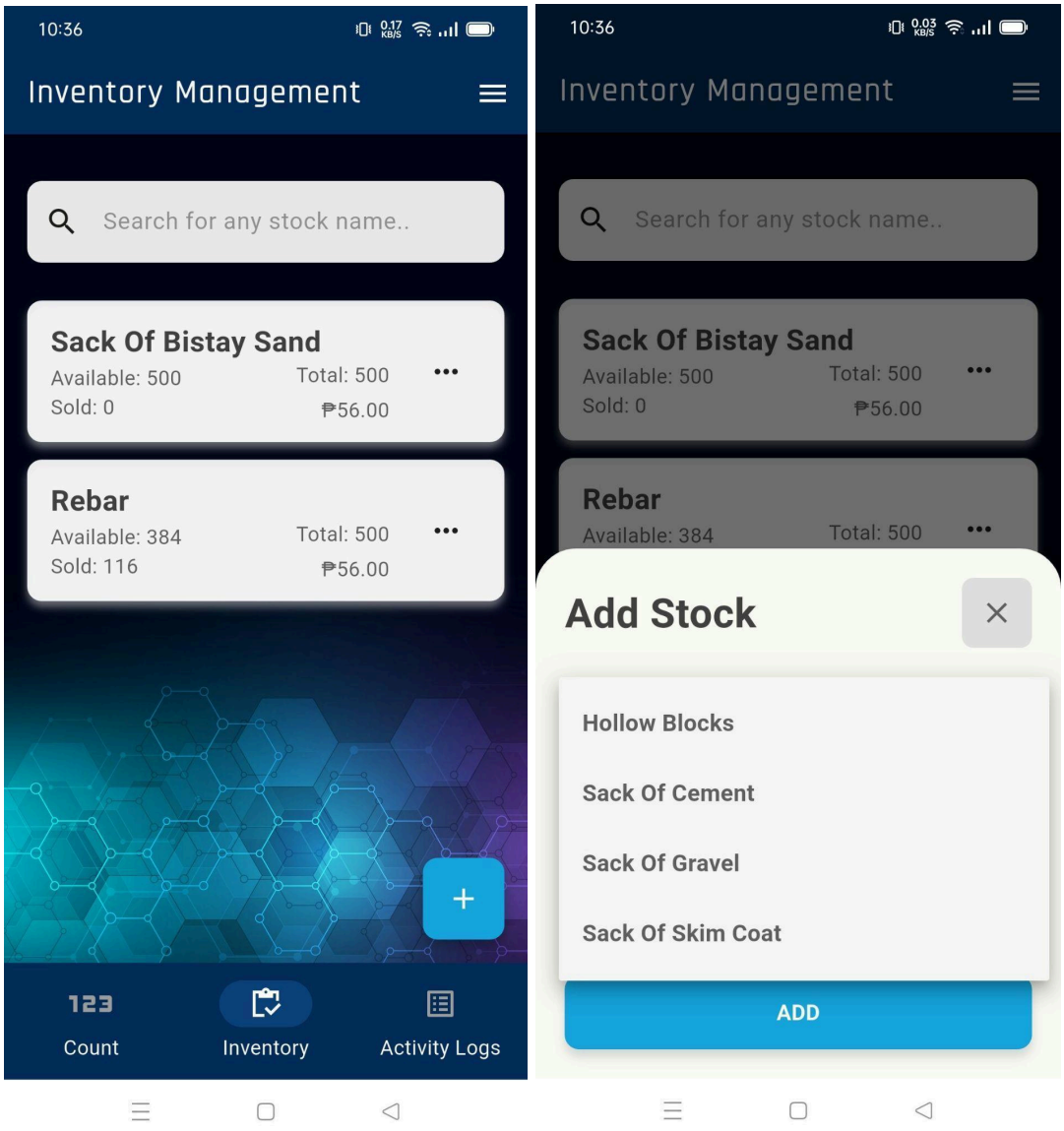


Figure 10 shows the Inventory widget allowing users to add a specific Stock item and its associated total count and save the data into the MongoDB database. The data would be displayed back in the Inventory widget, and the user can edit the total count by adding another item or deleting a stock item from the hardware supplies.

Figure 11:

Activity Logs Screen

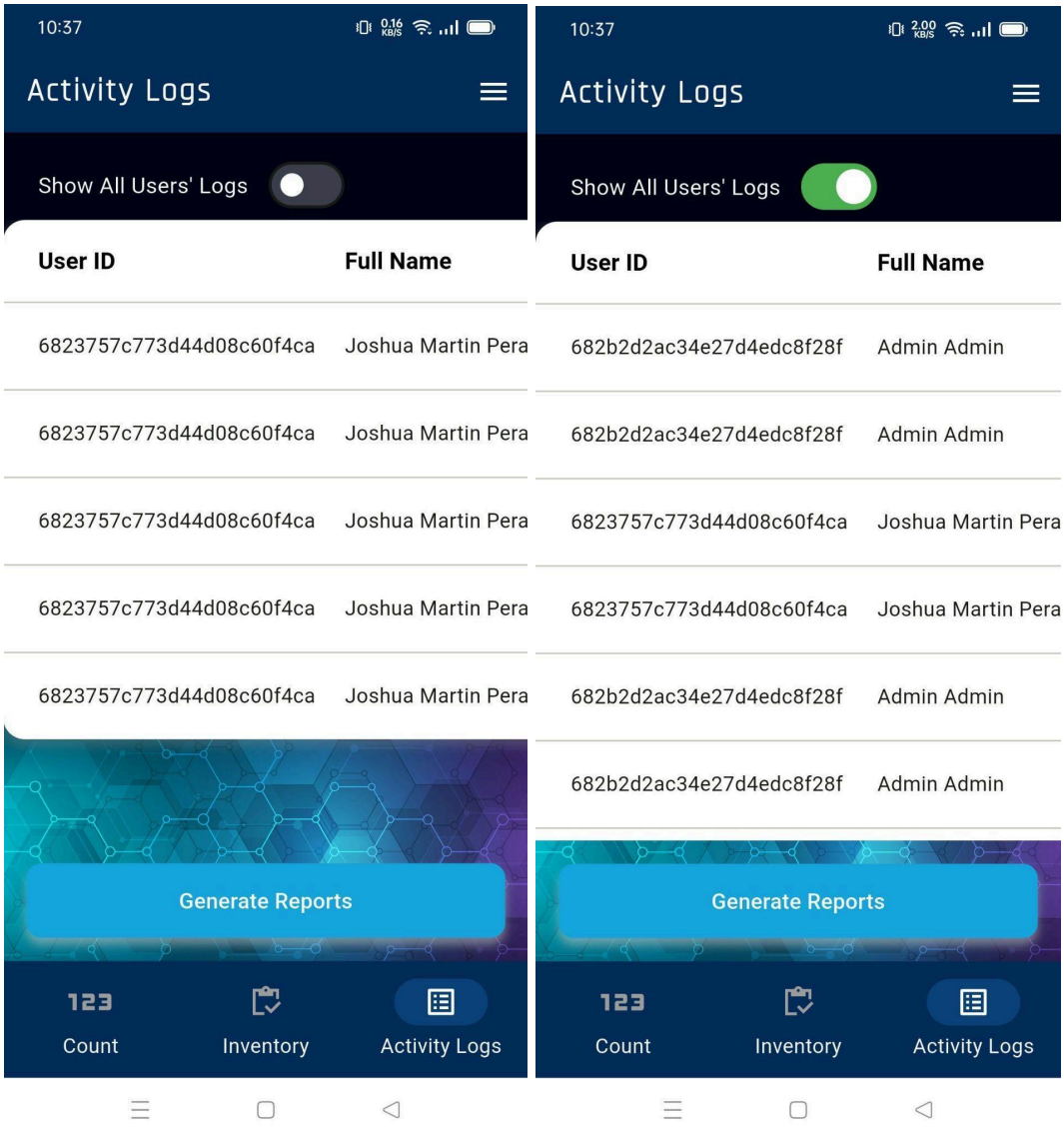


Figure 11 shows the Activity Logs widget, which displays the activities of the user who has logged in. It also implements a toggle switch that displays all the activity logs of all the other users who have signed up for the application.

Figure 12:

Reports Screen

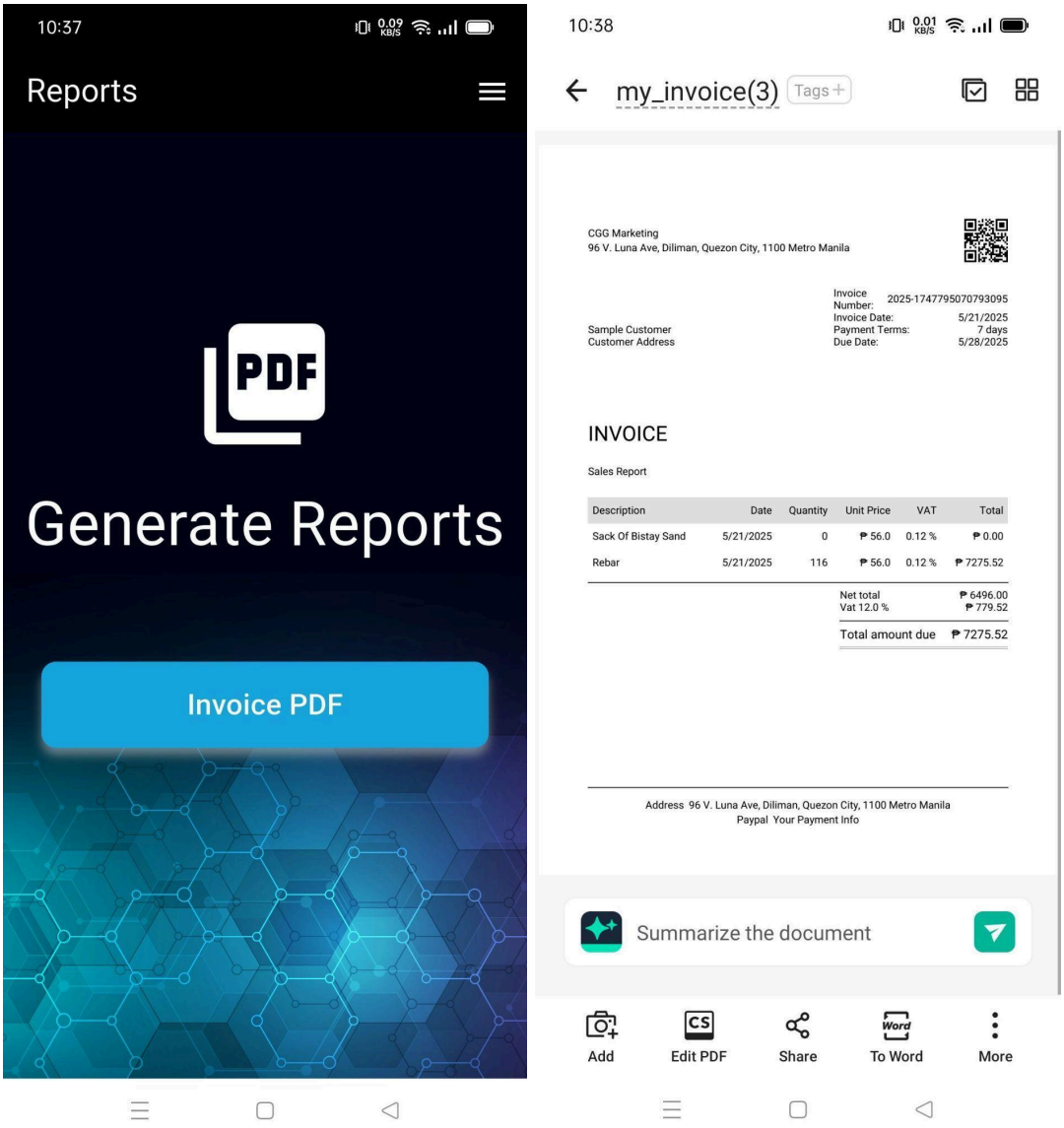


Figure 12 shows the Reports Screen where it can generate reports based on the transaction made from the inventory. It also displays the receipt for transaction in PDF.

Figure 13:

Sidebar Menu

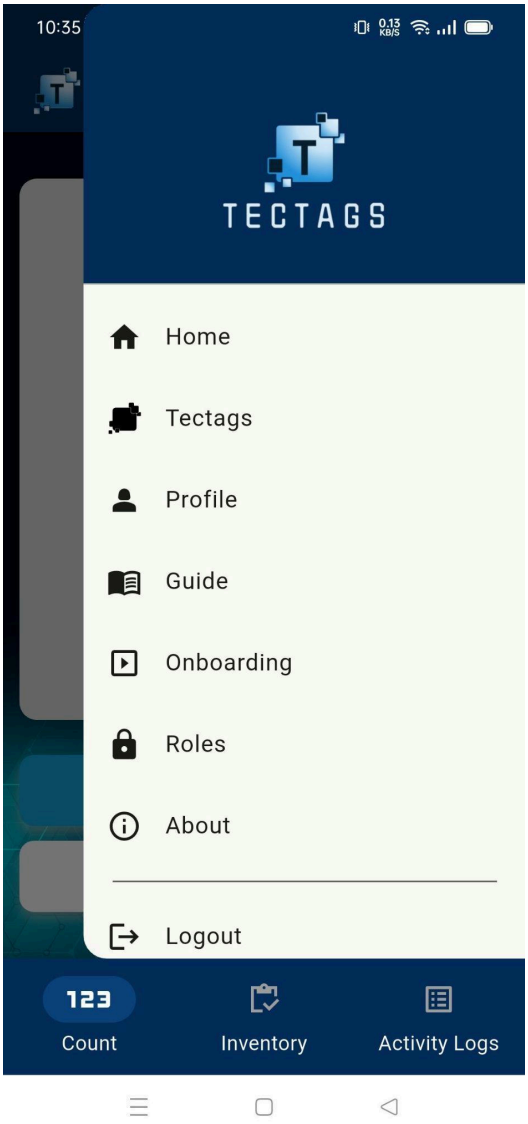


Figure 13 shows the Sidebar Menu of the application wherein it shows the Home screen, Profile screen, Guide screen, About the application screen, and also the Logout button.

Figure 14:

Profile Screen

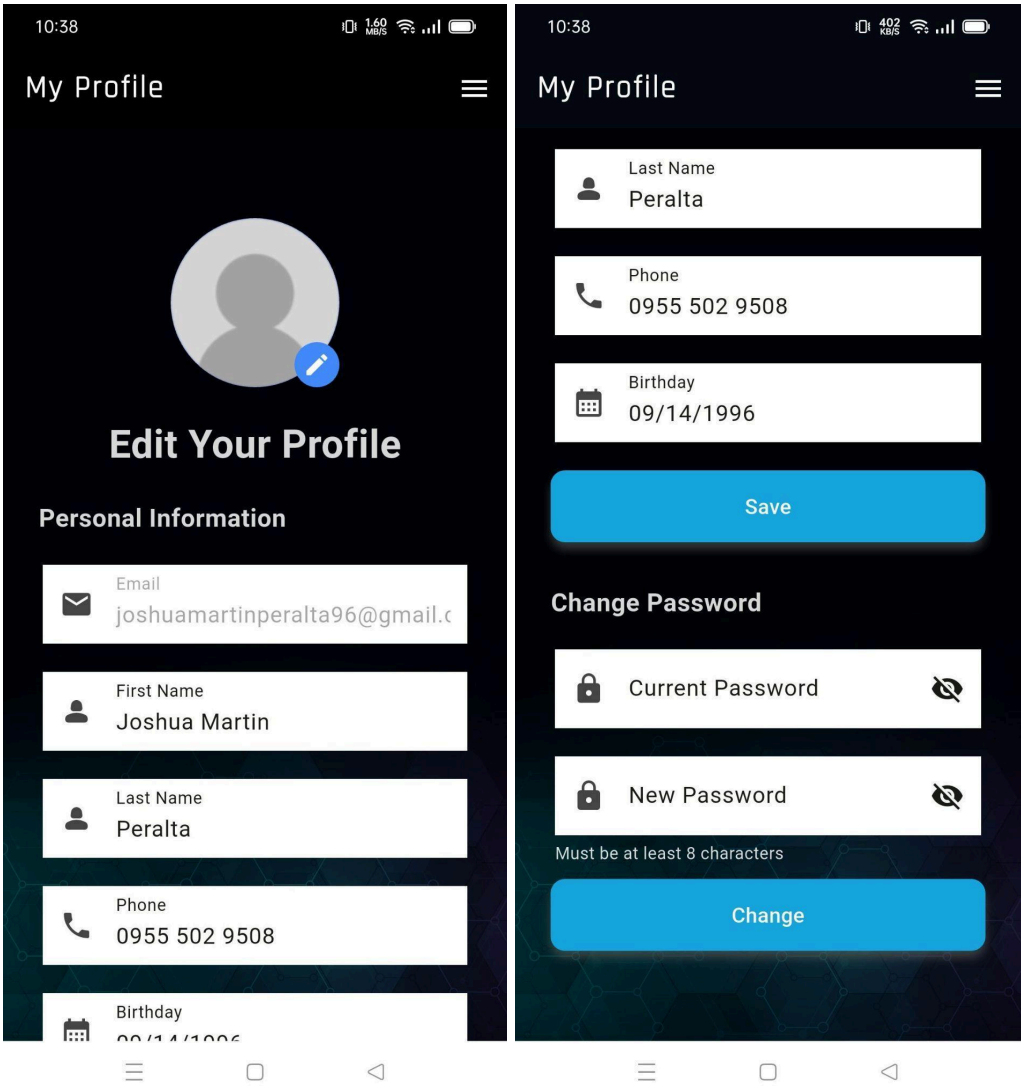


Figure 14 shows the Profile Screen where you can edit your own information in which you have registered from the sign up screen. The user can also change their password to make their application safe.

Figure 15:

Guide Screen

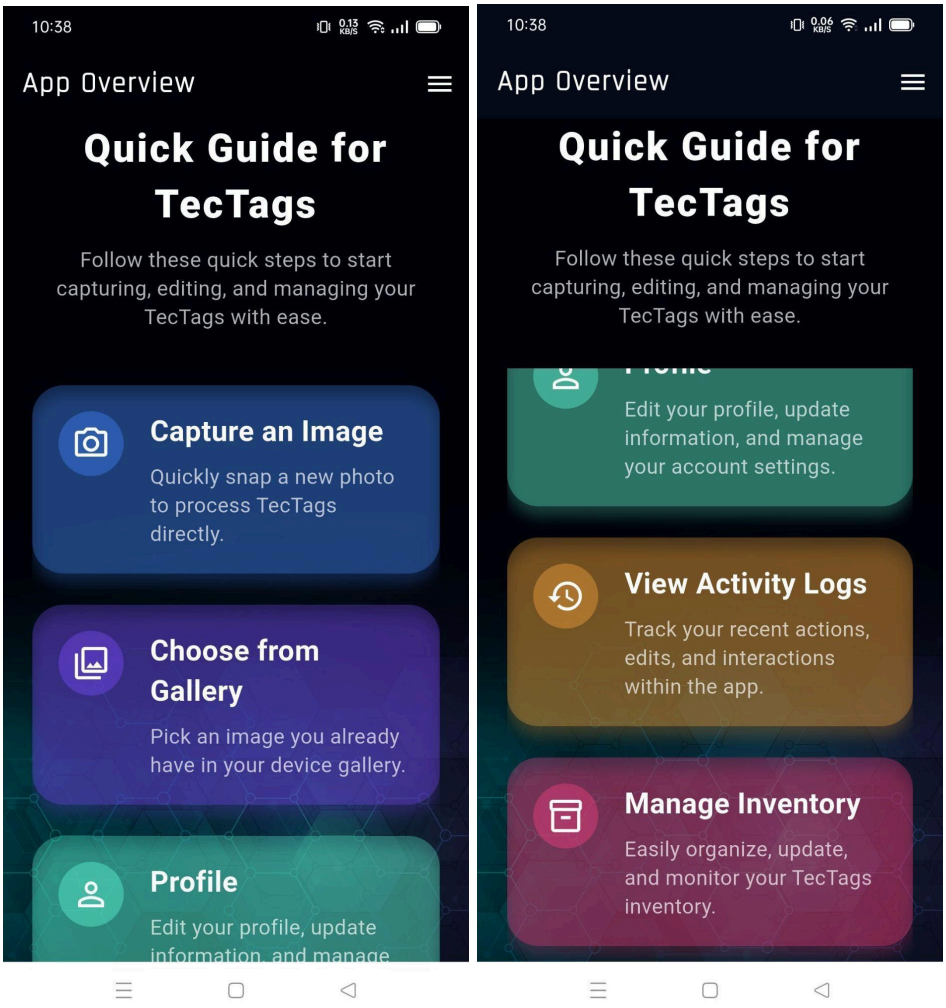
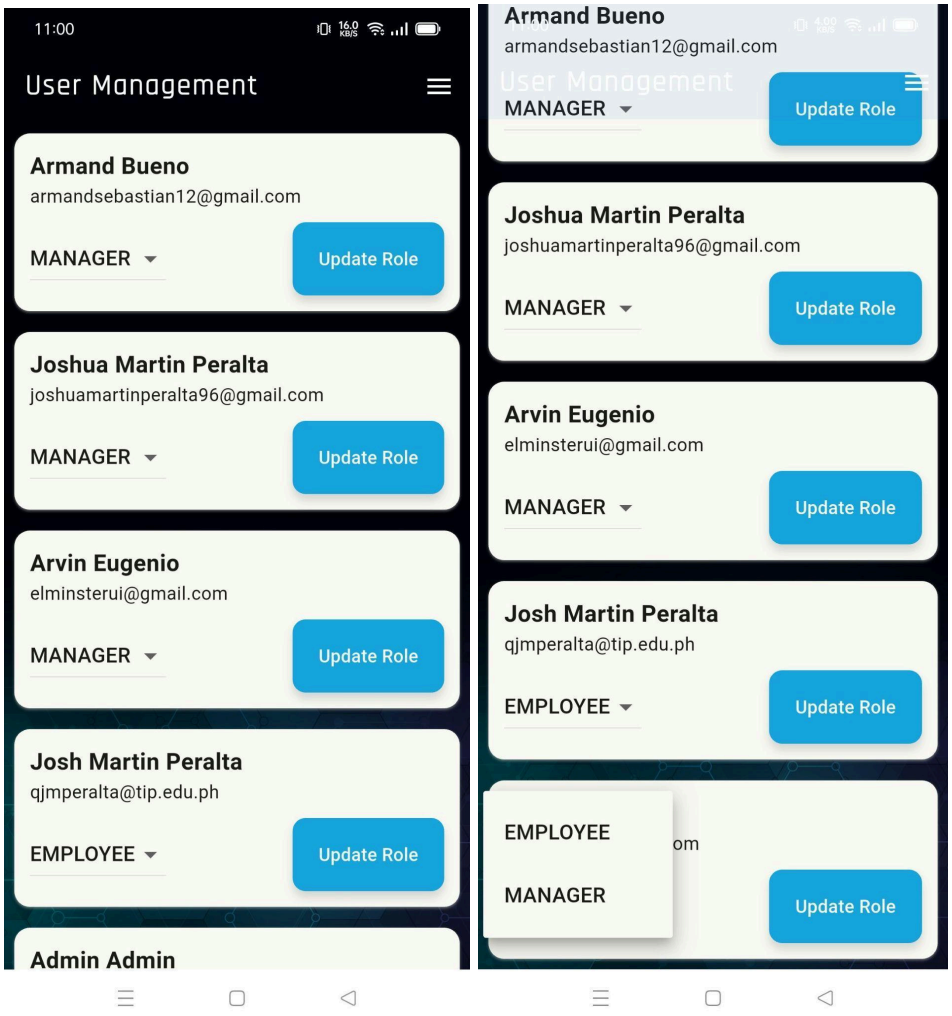


Figure 15 shows the Guide Screen where it shows the overview of the application and how each module works

Figure 16:

User Management Screen



```
const requireAuth = (req, res, next) => {
  const authHeader = req.headers.authorization;
  if (!authHeader)
    return res.status(401).json({ message: 'Authorization header missing' });

  const token = authHeader.split(' ')[1];

  try {
    const decoded = jwt.verify(token, process.env.SECRET);
    console.log('Decoded token payload:', decoded);
    req.user = decoded; // Attach the user to the request
    next();
  } catch (err) {
    return res.status(401).json({ message: 'Invalid or expired token' });
  }
};

// ROLE BASED ACCESS CONTROL
// ✅ Middleware: Role Authorization
const authorizeRoles = ( ...allowedRoles) => {
  return (req, res, next) => {
    if (!req.user)
      return res.status(401).json({ message: 'No user in request context' });

    if (!allowedRoles.includes(req.user.role)) {
      return res
        .status(403)
        .json({ message: 'Forbidden: Insufficient privileges' });
    }

    next();
  };
};
```

server.js

```
app.put(
  '/api/users/:id/role',
  requireAuth,
  authorizeRoles('manager'),
  async (req, res) => {
    try {
      const { role } = req.body;
      const userIdBeingUpdated = req.params.id;
      const requestingUserId = req.user._id.toString();

      if (!['employee', 'manager'].includes(role)) {
        return res.status(400).json({ message: 'Invalid role.' });
      }

      // Prevent self-demotion from manager to employee
      if (
        requestingUserId === userIdBeingUpdated && // It's the same user
        role !== 'manager' // Trying to change role to non-manager
      ) {
        return res.status(403).json({
          message:
            'Modifying your own role to a lower privilege level is not permitted.',
        });
      }

      const user = await User.findByIdAndUpdate(
        userIdBeingUpdated,
        { role },
        { new: true }
      );

      await Activity.create({
        userId: req.user._id,
        action: `Updated role of user ${userIdBeingUpdated} to ${role}`,
      });

      res.json({ message: 'User role updated.', user });
    } catch (error) {
      res.status(500).json({ message: 'Server error.', error: error.message });
    }
  }
);
```

server.js

```dart
static Future<Map<String, dynamic>> updateUserRole(
  String userId,
  String newRole,
) async {
  final prefs = await SharedPreferences.getInstance();
  final token = prefs.getString('auth_token');
  final url = Uri.parse('${baseUrl}users/$userId/role');

  try {
    final response = await http.put(
      url,
      headers: {
        'Authorization': 'Bearer $token',
        'Content-Type': 'application/json',
      },
      body: jsonEncode({'role': newRole}),
    );

    debugPrint("Update Role Response Code: ${response.statusCode}");
    debugPrint("Update Role Response Body: ${response.body}");

    if (response.statusCode == 200) {
      return {'success': true, 'message': 'Role updated successfully'};
    } else {
      return {
        'success': false,
        'message':
            jsonDecode(response.body)['message'] ?? 'Failed to update role',
      };
    }
  } catch (e) {
    return {'success': false, 'message': e.toString()};
  }
}
```

api.dart

Figure 16 shows the User Management screen where it implements a secure role-based access control system for modifying user roles in a Node.js/Express backend integrated with a Flutter frontend. The backend employs the requireAuth middleware to check the JWT token in the Authorization header, guaranteeing that only validated users can reach secured routes. It subsequently utilizes authorizeRoles to permit access exclusively for users with the 'manager' role. When a manager makes a request to /api/users/:id/role, the backend verifies valid roles, prevents self-demotion, modifies the user's role in the database, and records the action in the Activity collection.

In Flutter, the updateUserRole function manages the frontend request securely by extracting the token from SharedPreferences, appending it to the request headers, and including

the new role in the PUT request body. This guarantees that only authorized managers are able to start role changes. The backend and frontend work together to implement strict access control while facilitating secure and traceable role management

Figure 17:

About the App Screen

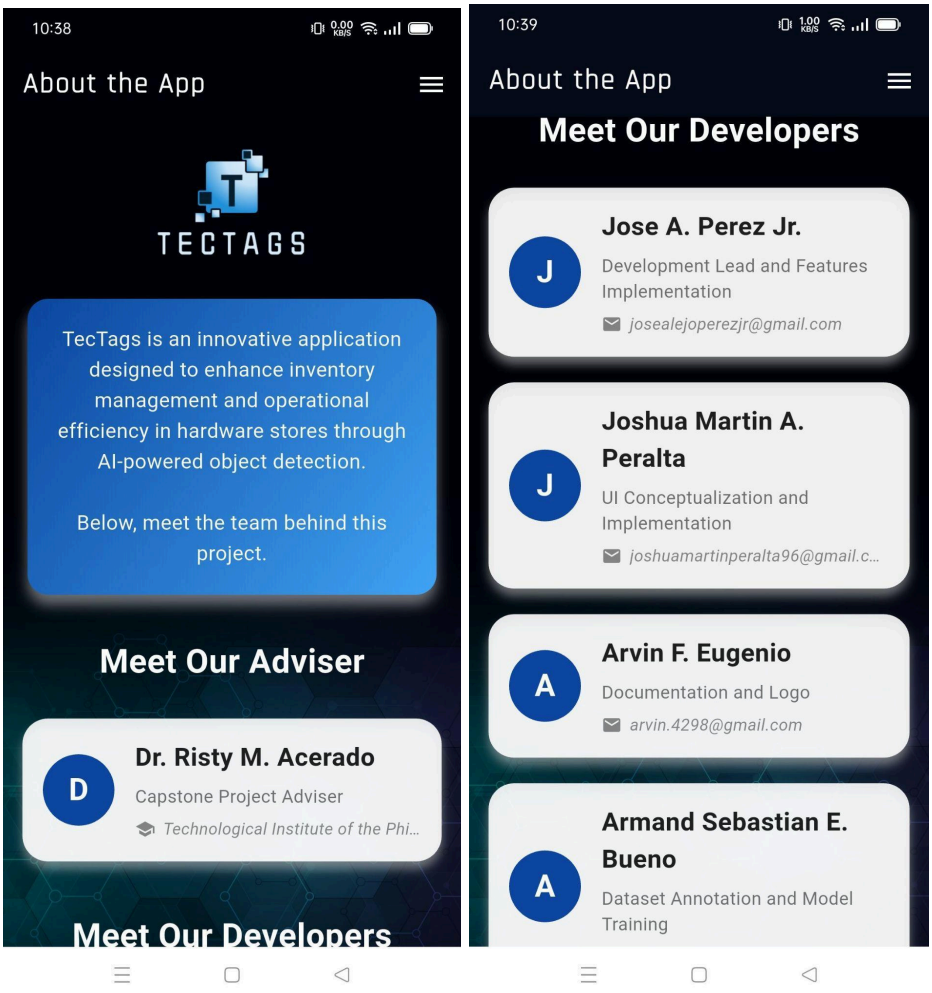


Figure 17 shows the About the App screen where it shows the overview of the application and introduces the developers of the project

Figure 18:

Onboarding Screen

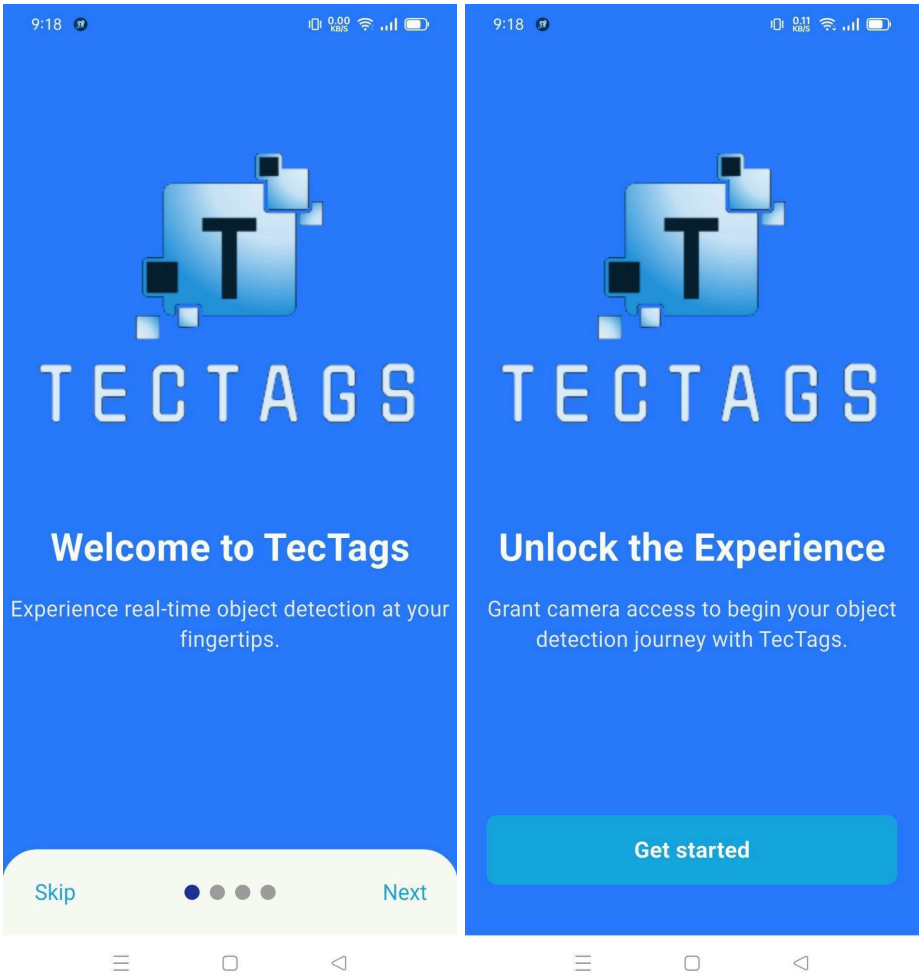


Figure 18 shows the Onboarding Screen where it is a one-time application guide that will appear before you can use the features of the application.

Figure 19:

Dashboard Screen

Figure 19 shows the Dashboard Screen where it will show the modules to explore like the Home Screen, Guide Screen and Supplies Screen.

Figure 20:

Supplies Screen

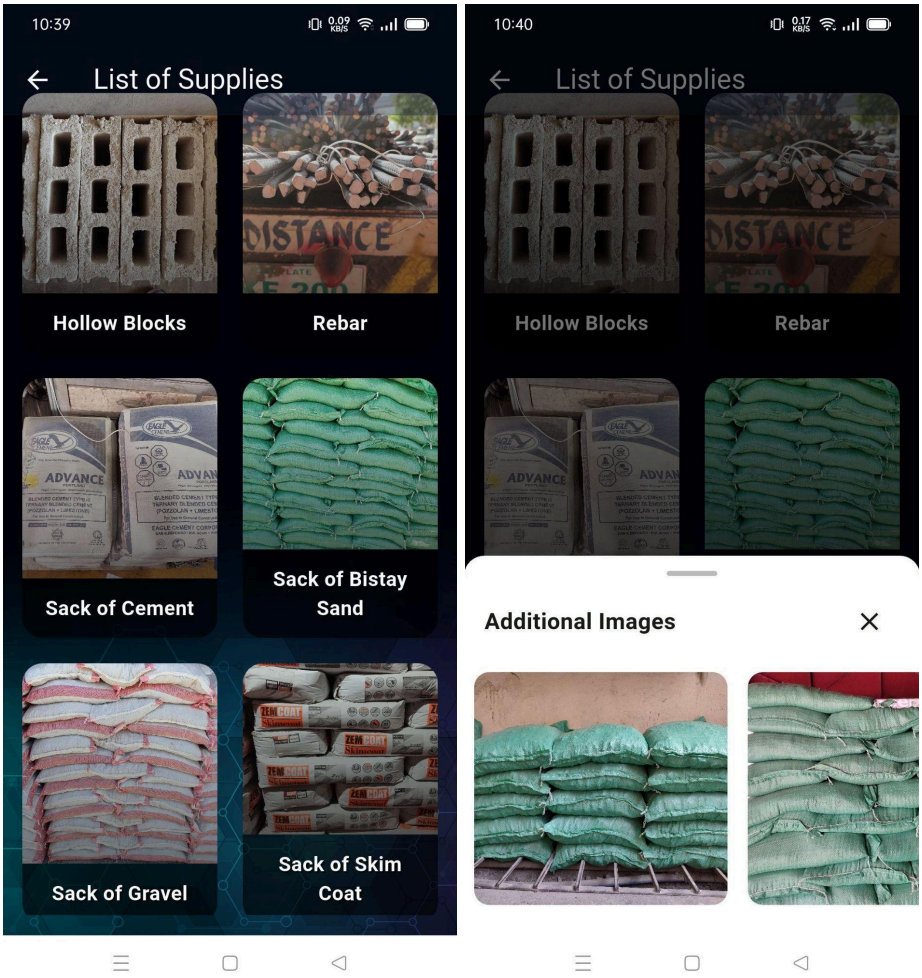


Figure 20 shows the Supplies Screen where all the categories of the list of supplies are shown here with additional images.

Developmental Research

The study of the gradual changes that occur throughout the project development process is known as Developmental Research. There are many changes in the application development used in this study. A prototype showing the initial design for the application was shown in terms of user interface design. Changes occur when the development process continually improves and simplifies the user interface design. As the development process advances, changes become noticeable. For the development process, collecting user feedback or opinions was a massive help because it allowed them to analyze the program itself, and it will serve as a guide for developers to maintain or modify the accuracy of the application features.

Work Plan

Table 1:

Work Breakdown Structure

Phases	Activities	Member	Schedule 2025				
			Jan	Feb	Mar	Apr	May
Planning Phase	Topic Proposal	Team					
	Adviser Approval	Team					
	Project Requirements	Team					
	Finalization of Project Requirements	Team					
Design Phase	System Architecture	Perez					
	UI/UX Design for Object Counting Interface	Team					
	Algorithm Selection	Perez					
Development Phase	Integrate Object Detection Model (TF Lite)	Team					
	Alpha Testing (Accuracy Validation)	Team					
Testing	Beta Testing	Team					
	Edge Case Handling	Team					
Release	App Deployment	Team					
	User Documentation	Team					

Table 1 is the project's work plan. Work Plan Table 6 (as represented by this table) is the project's Work Breakdown Structure. It displays the specific tasks that need to be accomplished from the Planning, Design, Development, and Testing phases up to the Release. The work plan shows that the project development will be from January 2025 to May 2025.

Project Development

Figure 21:

Agile Methodology

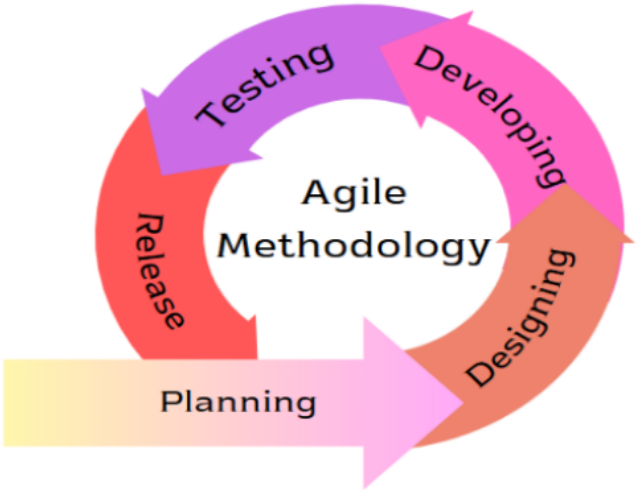


Figure 21 is a representation of Agile methodology. Agile methodology is a type of project management technique that is primarily utilized in software development, and requests and solutions emerge from the collaborative efforts of self-organizing and cross-functional teams as well as their clients.

Planning Phase: In this phase, the problems, potential solutions, aims, relevance, target user, features, and application functioning were discussed. It is where the developers decide the software's foundations. It informs the development team of accomplishments and the project's final destination.

Design Phase: In this phase, Figma and Visual Studio Code were used to create the UI design of the application, which should be readable, adequate, and user-friendly. It is where the developers begin designing the software and systems throughout the design phase to meet each

requirement. The architectural design defines all of the development's components, as well as communications with third-party services, user flows, database communications, and front-end representations and behavior of each component.

Development Phase: In this phase, we implemented the application's features and functionalities based on the specifications outlined during the requirements gathering stage. The development focused on integrating the object detection model into the mobile environment to support real-time object detection and number tagging. To achieve this, we will utilize PyTorch to enable on-device machine learning capabilities. PyTorch will be used specifically for object detection and computer vision tasks, allowing the application to process images efficiently on mobile devices. This integration ensures fast, reliable performance without relying on external APIs, which is crucial for offline functionality and responsiveness.

Testing Phase: The team tested the system's functionality and performance during this phase. The results of the development and this phase were discussed in Chapter 4.

Testing and Operating Procedure

Unit testing will be performed on the proposed system to ensure that all application parts are working according to specifications and that all required requirements are met accurately. This technique determines whether the project meets the activity's requirements and meets the project's intended use and user requirements.

- Alpha Testing – The first phase of a product's end-to-end testing to confirm that it meets business requirements and performs appropriately. Internal employees usually do it, and it takes place in a lab environment.

● Beta Testing – When it comes to beta testing, which is also known as user testing or customer validation, the goal is to ensure end users are satisfied with a software product before making it widely available.

Project Evaluation

Figure 22:

ISO 25(

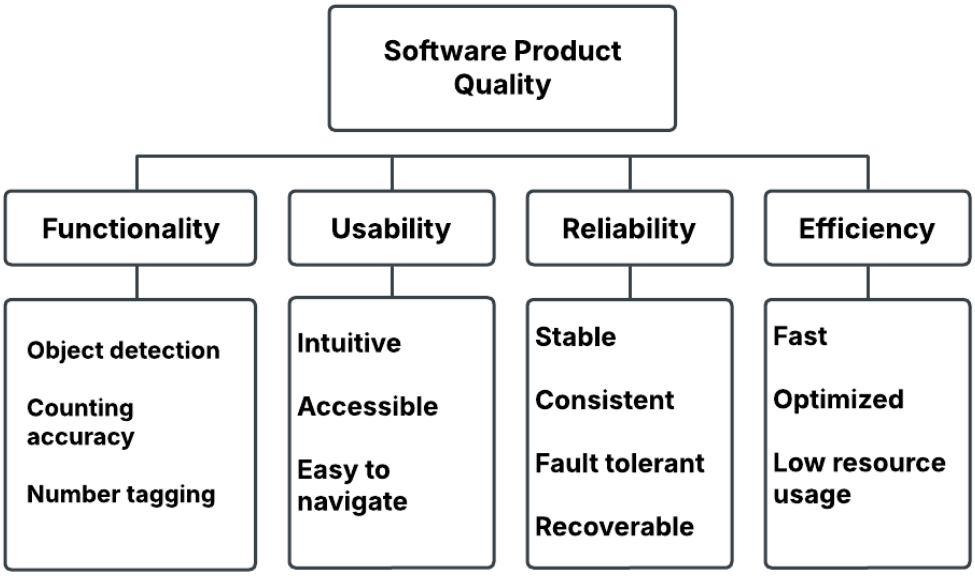


Figure 22 shows that the system will be evaluated based on ISO 25010, a standard for software quality assessment. This standard makes sure that the software meets quality attributes. The evaluation criteria include the following:

- **Functionality** – Measures how the system performs its tasks, including object detection, counting accuracy, and number tagging. The system is designed to identify objects accurately, count them accurately, and perform number tagging. The integration of

PyTorch for object detection ensures that the application can detect and count various hardware items, reducing errors in manual labor.

- **Usability** – Assesses the user experience, ensuring the interface is direct, accessible, and easy to navigate. The user interface is designed to be user-friendly, making it easy for hardware store staff to use TecTags in their daily work. By following usability, the application ensures that even non-technical users can efficiently navigate and operate its features.
- **Reliability** – Evaluate the system, ensuring it performs correctly with repeated failures. The application is built to be stable, fault-tolerant, and recoverable in case of unexpected crashes or interruptions. TecTags ensures availability by optimizing performance across different Android devices (Android 5.0 and above) and minimizing downtime during operation.
- **Efficiency** – Examines the system's ability to process and detect live objects. The system is optimized for multiple processes while maintaining fast and responsive performance. By leveraging on-device inference with PyTorch, TecTags minimizes processing power requirements, ensuring smooth operation without excessive battery or memory consumption.
- **Usability -** The development team must release the system application to the user and present the outcome achieved in meeting the requirements after finishing all previous stages of development.

The evaluation of TecTags was conducted using two approaches: online surveys using Google Forms and physical paper questionnaires distributed to CGG Marketing staff. 50 respondents comprised 25 CGG Marketing staff and 25 IT professionals. This balanced selection

ensures that IT professionals and CGG staff perspectives can see if technical insights were obtained.

The IT professional's evaluation was conducted online using Google Forms. The online survey gathered feedback for the application, mainly its functionality and efficiency. The IT professionals assessed how well TecTags handled object detection, counting, and number tagging and its compatibility across various mobile devices.

For the CGG Marketing staff, who are more focused on day-to-day work, paper questionnaires were given. These questionnaires aimed to capture their experiences regarding the usability, reliability, and overall impact of TecTags on their inventory management processes. The feedback from the paper surveys provided insights into how the application simplified the task of manual labor counting.

The combined evaluation confirmed that TecTags reduces manual errors, enhances real-time inventory tracking, and offers a user-friendly interface. The insights from both groups will guide enhancements to ensure the application meets a hardware store's needs.

A 6-point Likert scale was used to evaluate TecTags, designed to measure the intensity of agreement or disagreement with various statements about the system’s functionality, usability, reliability, and efficiency. The scale ranges from “Strongly Agree” to “Strongly Disagree,” intentionally omitting a neutral option to encourage respondents to make more definitive judgments. Each choice corresponds to a numeric score that is used in the quantitative analysis of the survey results.

Table 2:

Likert Scale

Score	Rating	Interpretation
6	5.51 – 6.00	Strongly Agree
5	4.51 – 5.50	Agree
4	3.51 – 4.50	Slightly Agree
3	2.51 – 3.50	Slightly Disagree
2	1.51 – 2.50	Disagree
1	1.00 – 1.50	Strongly Disagree

Frequency Distribution

The survey respondents comprised 25 CGG Marketing Staff (50%), 10 Construction Professionals (10%), 10 Customers (10%), and 5 IT Professionals (5%), totaling 50 participants from both technical and industry-related backgrounds.

Table 3:

Respondent Distribution

Respondents	No. of Respondents	Percentage
IT Professionals	5	10%
Customers	10	20%
Construction Professionals	10	20%
CGG Marketing Staff	25	50%
Total	50	100%

Statistical Treatment of Data

The study will calculate the result and average of the responses collected via the evaluation form using the weighted mean formula:

$$\textbf{\textit{Weighted Mean}} = T(R6\times6)+(R5\times5)+(R4\times4)+(R3\times3)+(R2\times2)+(R1\times1)$$

Where:

R6 to R1 *= Number of respondents selecting ratings from 6 (highest) to 1 (lowest)*

T *= Total number of respondents*

Questionnaire

Appendices A and C present the evaluation tools based on the ISO/IEC 25010 standard, which includes four key software quality characteristics: functionality, usability, reliability, and

efficiency. These tools were used to gather feedback from various respondent groups, including CGG Marketing Staff, IT Professionals, Customers, and Construction Professionals.

Potential for Commercialization

The TecTags mobile application for Android devices is designed to help hardware stores and staff efficiently detect and count specific construction supplies. This tool especially benefits individuals who are managing extensive inventories and streamlining their counting processes. Additionally, researchers using the TecTags application can gain valuable insights to aid in their development.

Market Model

Figure 23:

Market Model of TecTags Application

Figure 23 shows the TAM SAM SOM of the Market Model. The TAM, or the Total Available Market, has 12, 284 hardware stores in the Philippines. The SAM, or the Serviceable

Available Market, is 2,610 hardware stores in the Philippines' National Capital Region (NCR). The SOM, or the Serviceable Obtainable Market, is 461 hardware stores in Quezon City.

Measurable Benefits

TecTags is a mobile application developed by Information Technology students from the Technological Institute of the Philippines. Explicitly designed for hardware stores, the application automates inventory monitoring through object detection and counting. By reducing reliance on manual processes, TecTags minimizes human error, improves inventory accuracy, and enhances operational efficiency. It supports restocking and contributes to day-to-day operations.

Three-year Product Roadmap

Figure 24:

Three-year Product Roadmap for TecTags Mobile Application

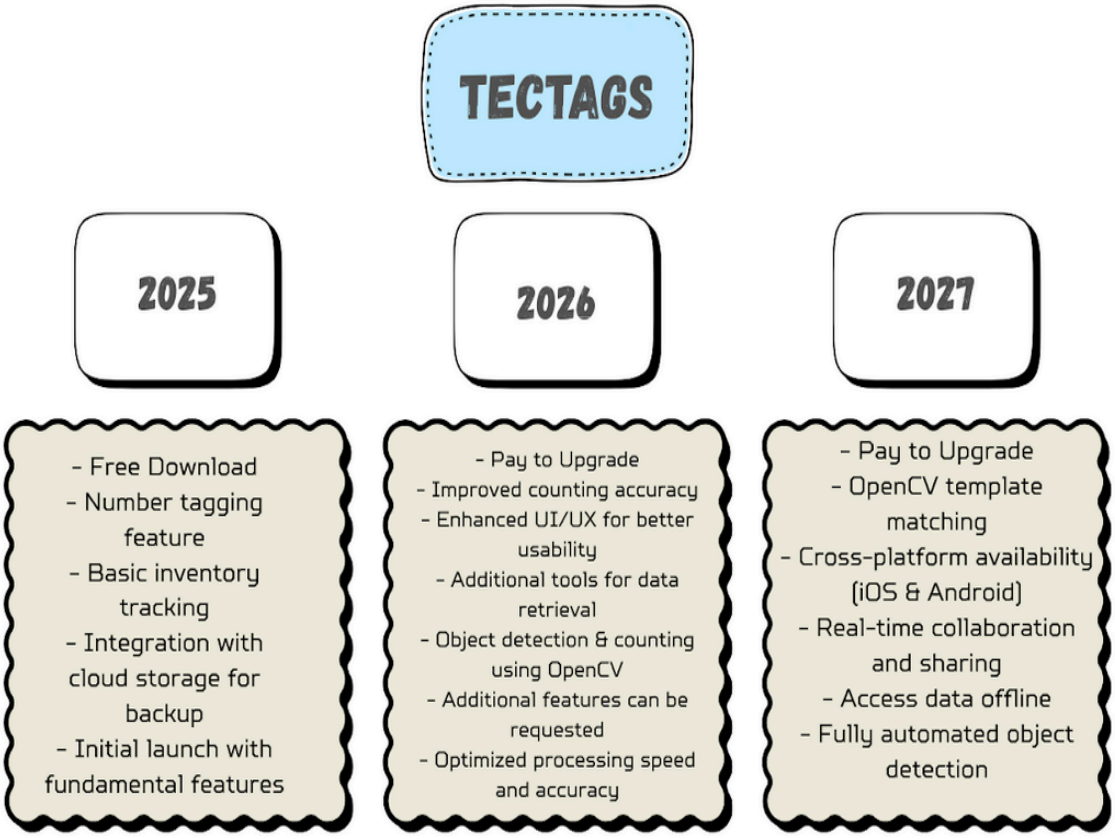


Figure 24 shows the Three-year Product Roadmap development of the TecTags application. In Year 1 (2025), The app will focus on number tagging, basic inventory tracking, and cloud storage for data backup. In Year 2 (2026), Optimizing counting accuracy, enhancing UI/UX, and new tools for data retrieval. Improved object detection and counting using PyTorch. By Year 3 (2027), TecTags will be available for iOS. PyTorch template matching will be integrated to enhance object recognition and tracking efficiency.

Go-to-Market Strategy

Figure 25:

Market Strategy

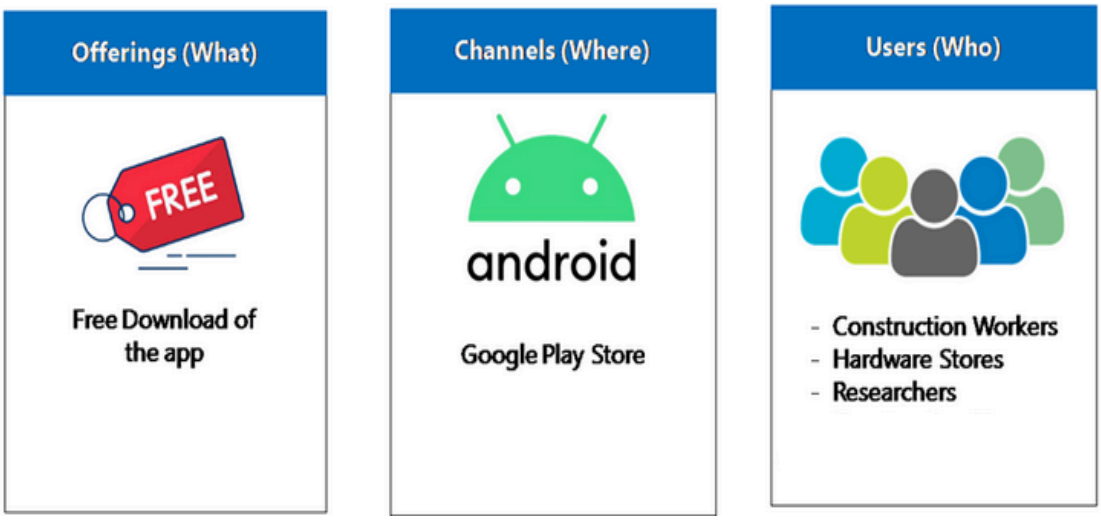


Figure 25 illustrates the Go-To-Market Strategy that will be implemented once the proposed system is completed. As previously stated, the App will be distributed for free to intended users. Because the App is only intended for mobile devices, the distribution channel is the Google Play Store. It will be available to all users, particularly those in hardware stores.

Business Model

Figure 26:

Business Model

<table>
<tr><th>Key Partners</th><th>Key Activities</th><th>Value Propositions</th><th>Customer Relationship</th><th>Customer Segment</th></tr>
<tr><td>CGG Marketing</td><td>- Object Counting Development

- Number Tagging

- Cloud Storage Integration</td><td>Accurate and Automated Object Counting: Reduces human error in inventory management.

Efficient Workflow: Lessens workload for personnel tracking inventory.

Mobile Convenience: Accessible via smartphones for on-the-go use.</td><td>- Contact through email</td><td>- Hardware Stores</td></tr>
<tr><td></td><td>Key Resources</td><td></td><td></td><td></td></tr>
<tr><td></td><td>Android 5.0 Phones and up</td><td></td><td></td><td></td></tr>
<tr><td colspan="3">Cost Structure</td><td colspan="2">Revenue Streams</td></tr>
<tr><td colspan="3">• Development
• App Maintenance</td><td colspan="2">• Advertisement
• Donation</td></tr>
</table>

Figure 26 shows the Business Model that the proponents agreed upon. TecTags is a free application for CGG Marketing, which supports the development and promotion of the application. The target users include hardware stores, as they can benefit from the app's automated object-counting and number-tagging features. The proponents will develop the application for Android 5.0 and above, ensuring compatibility with modern devices. The app will be made available for free through APK. The cost structure will primarily cover development

and app maintenance. Advertising and donations will be potential revenue to support the project's sustainability. For Customer Relationships, contact through email will be used.

Chapter 4

Results and Discussions

In this chapter, the study's results are discussed based on data collected and analyzed. This study aimed to develop TecTags, a mobile application that automates object counting using object detection algorithms. The evaluation focused on accuracy, processing speed, and usability in hardware industry applications. This chapter overviews the system's performance, field testing results, and evaluation outcomes.

Project Description

TecTags is an application for object counting using object detection algorithms. The system detects, identifies, and counts objects from images captured by smartphone cameras or uploaded from a gallery. Using artificial intelligence, our application provides object detection, making it a solution for the hardware industry. Hardware store owners and employees can struggle with counting hardware supplies. TecTags can help with these issues by providing a mobile application that can solve solutions for hardware stores.

The application has two main functionalities: object detection and automated counting. Users can capture images of hardware items such as Hollow blocks, Deform bar, Bistay sand, Gravel, and Skim coats or upload images from their gallery. The system processes these images, identifying and counting objects to lessen errors. TecTags enables store personnel to verify stock counts. TecTags provides a good solution for modern hardware store operations by enhancing inventory control and reducing inconsistency.

Development Discussion

The project TecTags: An Automated Object Counting Mobile Application for Hardware Stores was developed to help improve inventory management solutions using object detection algorithms. The system uses artificial intelligence to detect and count hardware items such as Hollow blocks, Deform bar, Bistay sand, Gravel, and Skim coats, improving inventory control. The development of TecTags ensures that the system meets the needs of hardware store owners and employees. The project team focused on key phases of development, including requirements planning, system design, implementation, testing, and evaluation. In the requirements planning phase, the team defined the project scope and objectives by researching hardware store inventory challenges, gathering data, and identifying target users. This phase ensured that the application was designed specifically for hardware store operations. In the design phase, the team selected the tools and system architecture to support object detection and counting. The interface was designed to be user-friendly, allowing easy image capture and processing. The construction phase involves the actual development and testing of the application. The implementation uses PyTorch for object detection and counting. We chose PyTorch because it's lightweight, making it fit for mobile applications while maintaining accurate detection. The evaluation phase followed the ISO 25010 software quality model, assessing TecTags based on functionality, performance efficiency, usability, reliability, and maintainability. The system was evaluated to ensure it accurately detects and counts objects for hardware stores. Reliability tests verify the consistency of detection results, while maintainability assessments ensure that the system can be easily updated in the future. After testing and improvements, TecTags is now for deployment to the CGG Marketing hardware store. The project team has gathered user feedback to identify areas for future enhancements, such as live detection and cloud-based processing for large-scale inventory management.

Project Structure

In developing TecTags, the team needed a tool for labeling and training the object detection model. Roboflow and Pytorch were the best choices for accurate object detection. Roboflow was used to handle dataset management, specifically image annotation. This tool allowed us to draw bounding boxes around objects of hardware supplies like hollow blocks, deform bar, bistay sand, gravel, and skim coat, ensuring the model can recognize them accurately. By using Roboflow's features, such as image resizing, the training dataset was optimized for better performance. PyTorch was selected for the object detection algorithm because of its lightweight and efficiency on mobile devices. The model was trained using Visual Studio Code as the primary development environment, which connected to remote cloud platforms with GPU support to accelerate the training process. The PyTorch model was used in the TecTags application, allowing users to capture or upload images for automated object counting. PyTorch ensures that TecTags only returns a count when objects are correctly detected. It removes the issue of false readings and improves the system's reliability. The capability of PyTorch also ensures that inventory counting is done quickly.

Figure 27:

VS Code Model Training

```python
import glob
import os
from multiprocessing import freeze_support
import torch
from ultralytics import YOLO
from tqdm import tqdm

def train_model_with_retry(model, data, epochs, initial_batch_size, device, imgsz, w
    """Train the model with retry on OOM error and dynamic batch size adjustment."""
    current_batch_size = initial_batch_size
    print(f"current_batch_size = {current_batch_size}")

    while True:
        try:
            print(f"Instantiating training with batch size: {current_batch_size}")
            model.train(
                data=data,
                epochs=epochs,
                batch=current_batch_size,
                imgsz=imgsz,
                device=device,
                workers=workers,
                optimizer="Adam",
                patience=20,
                cos_lr=True,
                save_period=5,
                exist_ok=True,
                close_mosaic=5
            )
            break
        except RuntimeError as e:
            if "out of memory" in str(e).lower():
                print(f"Out of memory! Reducing batch size from {current_batch_size}
                current_batch_size = max(current_batch_size // 2, 4)
            else:
                raise e
```

Figure 27 shows that Python code defines a function train_with_retry that orchestrates the training of a machine learning model. This function takes several arguments, including the model itself, training data, the number of epochs, the initial batch size, the device to train on (CPU or GPU), the input image size, and the number of worker processes for data loading. The core of the training loop resides within a while True block, indicating that it will continue indefinitely until a break statement is encountered. Inside this loop, the code attempts to train the model using the provided data and hyperparameters. Notably, it prints the starting training batch size at the beginning of each training attempt. The model.train() method is called with various training

configurations, such as the data, number of epochs, current batch size, device, image size, optimizer (AdamW), patience for early stopping, mixed-precision training (amp), cosine learning rate scheduler, saving frequency, and flags for existing checkpoint loading and closing mosaic augmentation. If the training completes without encountering a RuntimeError, the break statement is executed, and the loop terminates. However, if a RuntimeError occurs, specifically an "Out of memory" error, the code enters an exception block. Within this block, it prints a message indicating that it's reducing the batch size. The new batch size is calculated as the maximum of the initial batch size divided by 2 and 4, ensuring that the batch size doesn't become excessively small. This mechanism implements a dynamic batch size adjustment strategy to handle potential out-of-memory issues during training.

```
torch.cuda.empty_cache()
current_batch_size = max(current_batch_size // 2, 4)
print(f"current_batch_size = {current_batch_size}")

model = YOLO("yolov8n.pt").to(device)
else:
    raise e

def main():
    if torch.cuda.is_available():
        print(f"CUDA available: {torch.cuda.is_available()}")
        print(f"Using GPU: {torch.cuda.get_device_name()}")
        torch.backends.cudnn.benchmark = True
        device = "cuda"
    else:
        device = "cpu"
        model = YOLO("yolov8n.pt").to(device)

    initial_batch_size = 16
    epochs = 100
    imgsz = 640
    workers = 12

    train_model = {
        "model": model,
        "data": "config.yaml",
        "epochs": epochs,
        "initial_batch_size": initial_batch_size,
        "device": device,
        "imgsz": imgsz,
    }
```

After a potential "Out of memory" error and the subsequent reduction of the current_batch_size, the code attempts to clear the CUDA empty cache using torch.cuda.empty_cache(). This is likely done to free up memory before the next training attempt with the reduced batch size. If the RuntimeError is not an "Out of memory" error, the else block is executed, and the original exception e is re-raised, halting the training process. The code then defines a main() function, which serves as the entry point of the script. Inside main(), it first checks if CUDA is available and prints the result. If CUDA is indeed available, it prints a message indicating that it's using the GPU and sets torch.backends.cudnn.benchmark to True, which can potentially improve performance by allowing cuDNN to find the most efficient algorithms for the hardware. The device variable is then set to 'cuda' if CUDA is available, otherwise it defaults to 'cpu'. A YOLO model is initialized using YOLO("yolov8n.pt") and moved to the selected device. Several training hyperparameters are then defined: initial_batch_size is set to 16, epochs to 100, imgsz (image size) to 640, and workers (number of data loader workers) to 12. Finally, the train_model_with_retry function is called with all the defined parameters, initiating the training process with the specified model, data configuration, and hyperparameters.

Figure 28:

VS Code Yolo to Torchscript Conversion

```python
import torch
from ultralytics import YOLO
import argparse

def convert_to_torchscript(input_weights, output_path, imgsz=640):
    """
    Converts YOLO PyTorch model to TorchScript format
    :param input_weights: Path to .pt weights file
    :param output_path: Output path for torchscript file
    :param imgsz: Image size (must match training size)
    """
    device = "cpu"
    print(f"Using device: {device.upper()}")

    try:
        model = YOLO(input_weights)
        ts_path = model.export(
            format="torchscript",
            imgsz=imgsz,
            device=device,
            optimize=True,
            half=False,
            simplify=True
        )

        # Output path
        import shutil
        shutil.move(ts_path, output_path)
        ts_path = output_path

        print(f"\n✅ Successfully exported TorchScript model to: {ts_path}")
```

Figure 28 shows the Python script where it is likely part of the YOLO (You Only Look Once) object detection framework, focusing on converting a pre-trained YOLO model into a TorchScript format. This conversion makes the model more efficient and deployable, especially for production environments. The script defines a function called convert_to_torchscript that takes the path to the YOLO model's weight file, the desired output path for the TorchScript file, and the image size the model was trained on as inputs. It then loads the YOLO model using these weights and attempts to export it into the TorchScript format, specifying various export options like the format itself, the input image size, the device to use for the export (defaulting to CPU), whether to optimize the model, and whether to simplify the ONNX graph (an intermediate

representation). If an output path is provided, the script moves the generated TorchScript file to that location and prints a success message indicating where the converted model is saved.

```
def verify_torchscript(ts_path, imgsz=640):
    """Verifies TorchScript model can load and run inference"""
    print(f"\n✨ Verifying TorchScript model...")

    try:
        model = torch.jit.load(model_path)

        dummy_input = torch.randn(1, 3, imgsz, imgsz)

        with torch.no_grad():
            output = model(dummy_input)

        print(f"✅ Verification successful - model works! Output shape: {output[0].shap
        return ts_path
    except Exception as e:
        print(f"❌ Verification failed: {str(e)}")
        return None

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description="Convert YOLOv8 model to TorchScript")
    parser.add_argument("--input", type=str, required=True, help="Input .pt model path"
```

There is also a crucial function defined called verify_torchscript where its function is to take the path to the newly created TorchScript model and the expected image size as input. Its main purpose is to check if the converted model is actually usable. Inside this function, it first attempts to load the TorchScript model using torch.jit.load(). Then, to simulate a forward pass (how the model processes an image), it creates a random input tensor of the expected shape (batch size of 1, 3 color channels, and the specified image size). It then runs this dummy input through the loaded model within a torch.no_grad() context, which disables gradient calculations for faster inference. If this process completes without errors, it prints the shape of the output tensor and a success message, indicating that the TorchScript model has been successfully loaded and can run inference. However, if any exception occurs during the loading or inference process, it catches the error, prints a failure message along with the error details, and returns None.

```
if __name__ == "__main__":
    parser = argparse.ArgumentParser(description="Convert YOLOv8 model to TorchScript")
    parser.add_argument("--input", type=str, required=True, help="Input .pt model path"
    parser.add_argument("--output", type=str, default="best_torchscript", help="Output
    parser.add_argument("--imgsz", type=int, default=640, help="Image size used during

    args = parser.parse_args()

    convert_to_torchscript(
        input_weights=args.input,
        output_path=args.output,
        imgsz=args.imgsz
    )
```

After setting up the argument parser to handle command-line inputs, the script proceeds to actually parse these arguments using parser.parse_args(). The results of this parsing are stored in the args variable. This args object now contains the values provided by the user (or the default values if the user didn't specify them) for the --input, --output, and --imgsz arguments.

Lastly, the script calls the convert_to_torchscript function. It passes the values obtained from the command-line arguments to this function. Specifically, args.input (the path to the input .pt weights file), args.output (the desired output path for the TorchScript file), and args.imgsz (the image size) are passed as arguments to convert_to_torchscript. This means that the script will use the user-provided (or default) values to perform the YOLO to TorchScript conversion.

Figure 29:

VS Code Data Augmentation

```
import albumentations as A
import cv2
import os
import random

# Set paths
INPUT_IMAGES_DIR = "C:/Users/Armand Bueno/Desktop/TRAINER Pytorch/dataset for training
INPUT_LABELS_DIR = "C:/Users/Armand Bueno/Desktop/TRAINER Pytorch/dataset for training
OUTPUT_IMAGES_DIR = "C:/Users/Armand Bueno/Desktop/TRAINER Pytorch/dataset for trainin
OUTPUT_LABELS_DIR = "C:/Users/Armand Bueno/Desktop/TRAINER Pytorch/dataset for trainin

# Create output directories if they don't exist
os.makedirs(OUTPUT_IMAGES_DIR, exist_ok=True)
os.makedirs(OUTPUT_LABELS_DIR, exist_ok=True)

def get_augmenter():
    return A.Compose([
        A.HorizontalFlip(p=0.5),
        A.VerticalFlip(p=0.3),
        A.Rotate(limit=45, p=0.7),
        A.RandomBrightnessContrast(p=0.5),
        A.RandomGamma(p=0.4),
        A.HueSaturationValue(hue_shift_limit=20, sat_shift_limit=30, val_shift_limit=2
        A.RandomScale(scale_limit=0.3, p=0.7),
        A.ShiftScaleRotate(shift_limit=0.1, scale_limit=0.2, rotate_limit=30, p=0.5),
        A.GaussianBlur(blur_limit=[3, 5], p=0.3),
        A.RandomShadow(p=0.3),
        A.CLAHE(p=0.2),
    ])
```

Figure 29 shows the Python script which is designed to perform image augmentation, a technique used to artificially increase the size of a training dataset by creating modified versions of the images. It starts by importing necessary libraries for the augmentation techniques, cv2 for image manipulation, os for file system operations, shutil for file operations, and random for random number generation.

The script then defines the paths to the input images and labels, as well as the directories where the augmented images and their corresponding labels will be saved. It ensures that the output directories for augmented images and labels exist; if they don't, they are created.

The core of the augmentation process lies in the get_augmentor() function. This function uses the albumentations library to define a sequence of image transformations that will be applied. These transformations include horizontal and vertical flips, random rotations, random rotations with a specific angle limit, adjustments to brightness and contrast, changes in gamma, modifications to hue, saturation, and value, random scaling, random shift, scale, and rotation, Gaussian blur, the addition of random shadows and rain effects, and Contrast Limited Adaptive Histogram Equalization (CLAHE). Each of these transformations has a probability (p) associated with it, determining how often it will be applied to an image. The script sets a target_total_augmented variable to 1000, indicating the desired number of augmented images to be generated. The subsequent part of the script (not shown in the image) would likely iterate through the input images, apply these augmentations, and save the resulting images and their corresponding labels in the designated output directories until the target number of augmented images is reached.

```
original_images = [f for f in os.listdir(INPUT_IMAGES_DIR) if f.endswith(('.jpg', '.png
print(f"Will create approximately {per_image_augmentations * len(original_images)} augm

for filename in original_images:
    img_path = os.path.join(INPUT_IMAGES_DIR, filename)
    img = cv2.imread(img_path)

    label_filename = filename.rsplit('.', 1)[0] + '.txt'
    label_path = os.path.join(INPUT_LABELS_DIR, label_filename)

    if not os.path.exists(label_path):
        print(f"label not found for {filename}, skipping...")
        continue

    shutil.copy(img_path, os.path.join(output_labels_dir, label_filename))
    shutil.copy(label_path, os.path.join(output_labels_dir, label_filename))

    for i in range(per_image_augmentations):
        augmenter = get_augmenter()
        augmented = augmenter(image=img)['image']
        new_image_name = f"{filename.rsplit('.', 1)[0]}_{i:03d}.{aug[i]}.jpg"
        new_label_name = f"{filename.rsplit('.', 1)[0]}_{i:03d}.{aug[i]}.txt"

        cv2.imwrite(os.path.join(OUTPUT_IMAGES_DIR, new_image_name), augmented)
        shutil.copy(label_path, os.path.join(OUTPUT_LABELS_DIR, new_label_name))

print("✓ Massive augmentation completed!")
```

The script takes the augmentation process into action. First, it gathers a list of all image filenames present in the input_images_dir, specifically looking for files ending with '.jpg' or '.png'. It then calculates the number of augmentations to perform for each original image to reach the target_total_augmented count. This is done by dividing the target total by the number of original images. A message is then printed indicating how many augmented versions will be created for each original image.

The script also iterates through each filename in the original_images list. For each original image, it constructs the full path to the image and reads it using cv2.imread(). It also determines the corresponding label filename by replacing the image file extension with '.txt' and constructs the full path to the label file. It then checks if the label file exists. If a label file is not found for a particular image, a message is printed, and the script moves on to the next image. If the label file exists, the script copies both the original image and its corresponding label file to

the output directories for augmented data. This ensures that the original, un-augmented data is also present in the output directories.

Lastly, the script enters a loop that runs for the calculated per_image_augmentations number of times. Inside this loop, it gets the augmentation pipeline defined by the get_augmentor() function. It then applies this augmentation pipeline to the current original image, resulting in an augmented image. New filenames for both the augmented image and its label are generated by appending an augmentation index to the original filename. The augmented image is then saved to the output_images_dir using cv2.imwrite(). Finally, the original label file is copied to the output_labels_dir with the new augmented label filename. After processing all original images and performing the specified number of augmentations for each, the script prints a "Massive augmentation completed!" message, signaling the end of the augmentation process.

Figure 30:

Confusion Matrix of Object Detection Model

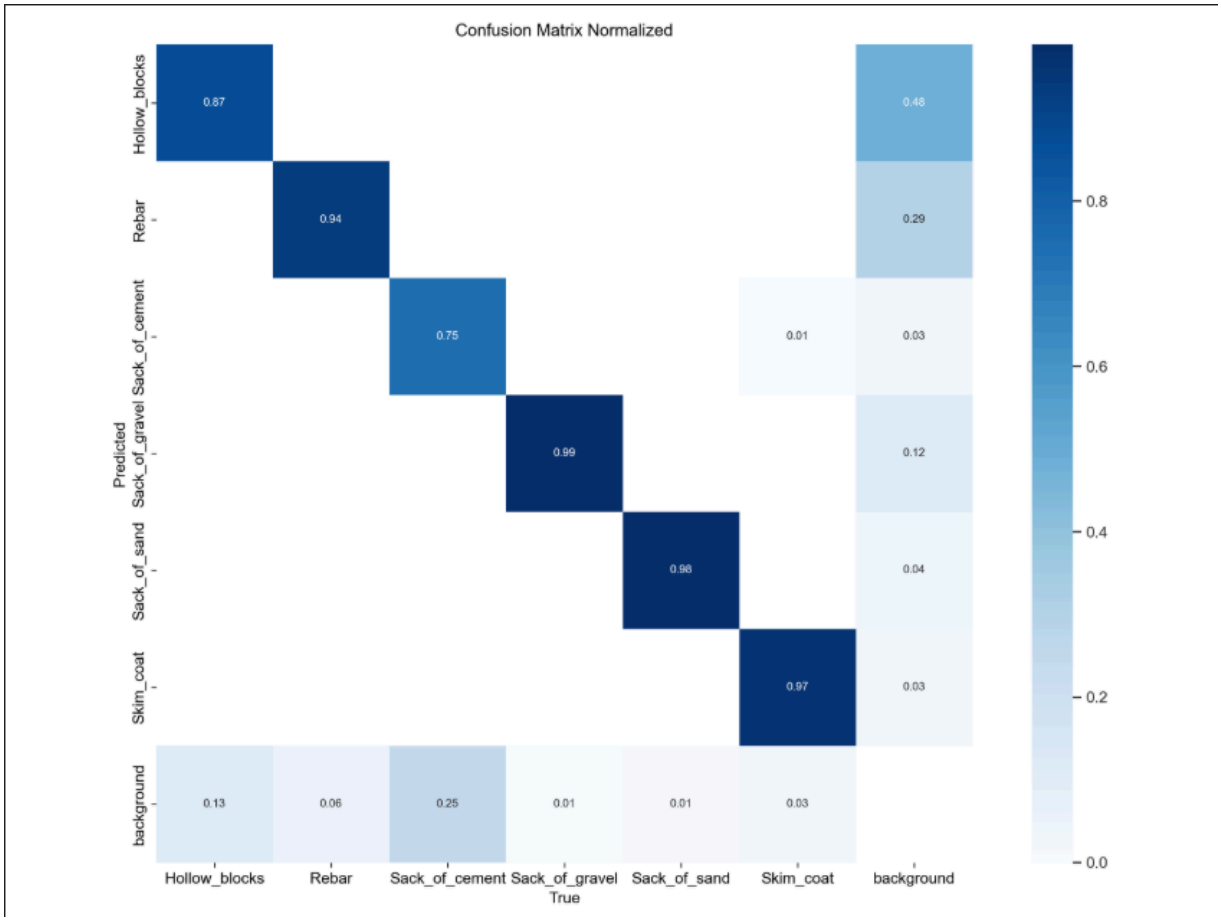


Figure 30 shows the normalized confusion matrix provides a clearer understanding of the model's object detection performance for each class relative to the total number of actual instances. Higher values along the diagonal represent correct classifications, while off-diagonal values indicate misclassifications.

In the TecTags evaluation:

- **Gravel** achieved a high true positive rate of **99%**, showing strong classification accuracy.
- **Skim Coat** and **Bistay Sand** were predicted correctly **97%** and **96%** of the time, respectively.
- **Cement** had a **75%** correct classification rate, showing some confusion with other classes, particularly background noise (**25%** misclassification as background).
- **Rebar** also achieved a **94%** high classification rate of detection.
- **Hollow Blocks** performed very well, achieving an **87%** correct classification rate with minimal confusion.

The background class shows that a significant portion of object instances, especially for Cement and Hollow Blocks, were sometimes misclassified as background, highlighting areas where further model refinement or more diverse training data may improve performance.

The matrix reflects that TecTags' object detection model is reliable, especially for distinguishing materials like Gravel, Skim Coat, Bistay Sand and Rebar, but there is room for enhancement in differentiating certain hardware materials and reducing background misclassifications.

Figure 31:

Image Annotations

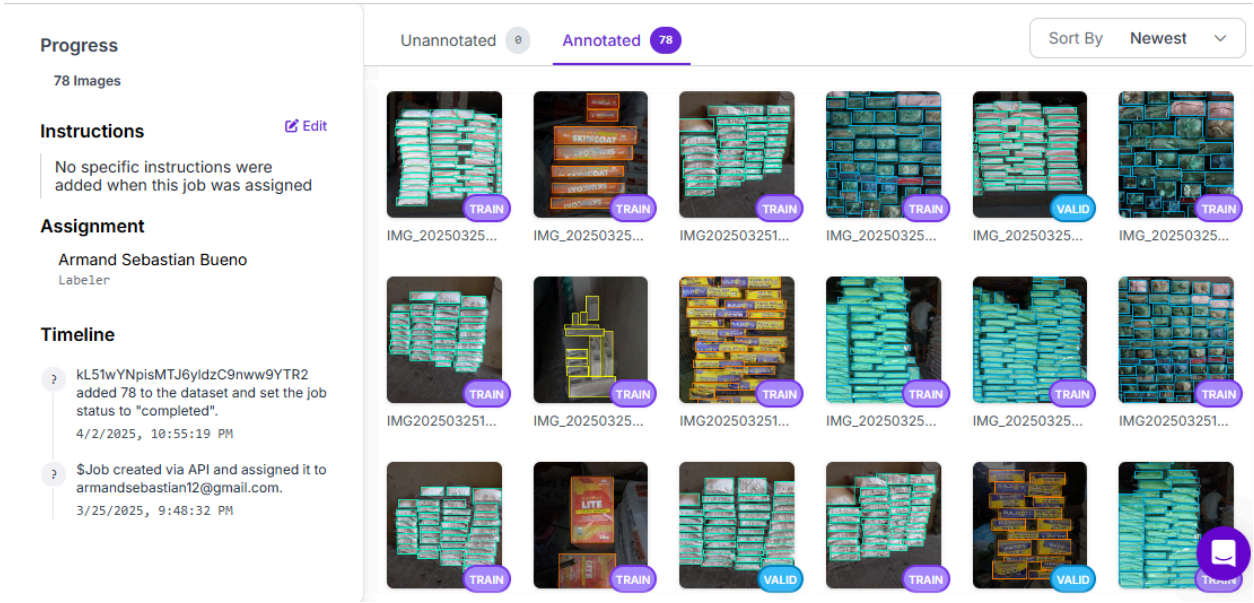
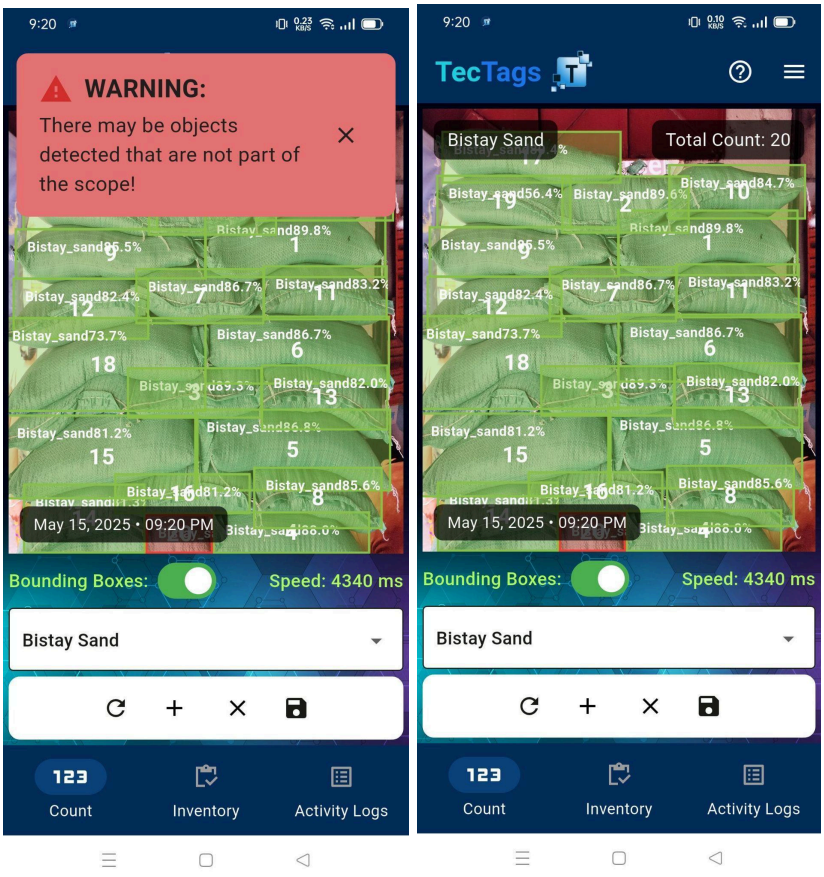


Figure 31 shows how Roboflow annotates hardware materials and is streamlined through AI-assisted tools that expedite the labeling process for components such as cement, sand, skim coat, hollow blocks, and deformed bars. The platform offers features like bounding boxes and polygon annotations to precisely define the positions of these materials within images. Roboflow's Label Assist feature can utilize pre-trained models to suggest annotations, significantly reducing manual effort and enhancing consistency across datasets. This efficient annotation process is particularly beneficial for training computer vision models in construction.

Figure 32:

Object Detection Capture Function



```
// PYTORCH OBJECT DETECTION

// initialize object detector
late _model<ObjectDetector> _objectModel=YoloV8;
// List<DetectedObject> editableBoundingBoxes = []; // Editable list of bounding boxes
List<DetectedObject> editableBoundingBoxes = [];
bool _isAddingBox = false;
bool _isRemovingBox = false;
// for inference speed checking
String? _textToShow;

// toggle bounding box, labels, and scores except the number tag
bool _showBoundingInfo = true;

// auto populate detected stocks from doObjectDetection method
List<String> _detectedStockList = []; // Your fixed list
List<String> _allStocks = []; // Dynamic from detection
Map<String, Map<String, int>> _stockCounts = {};

// PREVENT MULTIPLE REENTRY FOR OPENING ADD PRODUCT MODAL
bool _isAddProductModalOpen = false;
```

Figure 32 illustrates the process of object detection, where an image is either captured through a camera or selected from the gallery where it is analyzed to identify and locate objects within it. Object detection models, such as those based on convolutional neural networks, process the input image to recognize various objects and identify them with bounding boxes. This technique enables applications to interpret visual data, facilitating functionalities like automated tagging. For instance, Google's ML Kit offers on-device object detection capabilities that can analyze both live camera feeds and static images to detect and track objects in real-time.

Figure 33:

Model Training Confidence Level

Class	Images	Instances	Box(P)	Box(R)	Box(F1)	mAP50
all	97	3732	0.916	0.873	0.926	0.663
Hollow_blocks	97	1584	0.893	0.797	0.886	0.574
Rebar	97	933	0.89	0.862	0.934	0.594
Sack_of_cement	97	215	0.945	0.679	0.802	0.486
Sack_of_gravel	97	567	0.929	0.974	0.975	0.544
Sack_of_sand	97	281	0.944	0.972	0.981	0.643
Skim_coat	97	152	0.896	0.954	0.978	0.778

Figure 33 shows that the results of an object detection model trained on a dataset of 97 images containing seven different classes of construction materials: "Hollow blocks," "Rebar," "Sack of cement," "Sack of gravel," "Sack of sand," and "skim coat." The table provides several evaluation metrics for the model's performance, both overall ("all") and for each specific class. Looking at the "Instances" column, we can see the number of objects detected for each class, with "Hollow blocks" having the most instances (1584) and "skim coat" having the fewest (152). The columns "BOX(P)," "BOX(R)," "MAP50," and "MAP50-95" represent precision, recall, and mean average precision at different Intersections over Union (IoU) thresholds for the bounding box predictions. Higher values in these columns indicate better performance.

Overall, the model achieves high precision (0.916) and recall (0.873) for bounding box predictions across all classes. The mean average precision at an IoU of 0.50 (MAP50) is 0.926, and the mean average precision across IoU thresholds from 0.50 to 0.95 (MAP50-95) is 0.683.

Examining the per-class performance, "Sack of sand" and "skim coat" show particularly high MAP50-95 scores (0.643 and 0.778, respectively), suggesting the model is very effective at accurately detecting these classes. "Sack of cement" has the lowest MAP50-95 score (0.486), indicating it might be the most challenging class for the model to detect accurately across different IoU thresholds.

Project Evaluation

The TecTags mobile application was presented to respondents who were selected for both technical and non-technical evaluations. During the demonstration, the functionality of the application, specifically its object detection and counting capabilities was showcased using sample hardware materials. Respondents were able to observe how the system captured, processed, and labeled objects such as hollow blocks, deform bars, and gravel through the app interface. Following the demonstration, respondents were given printed evaluation forms to assess the application. The technical and functional aspects of TecTags were then reviewed and rated by the respondents based on the ISO/IEC 25010 Software Product Quality standards.

Risk Management Plan

Appendix E shows the detailed risk management plan of the study to help prioritize risk and focus resources on mitigating the most significant threats.

Summary of Results

The TecTags mobile application was evaluated by 50 respondents, including CGG Marketing Staff and IT Professionals. Using the ISO/IEC 25010 standards, the app was rated based on functionality, usability, portability, security, and reliability.

Table 4

Gender Distribution of Respondents

Gender	Number of Respondents	Percentage
Male	41	82.0%
Female	9	18.0%
Prefer not to say	0	0%
Total	50	100%

Table 4 shows the 50 respondents from the survey. Most of them are identified as Male respondents have 41 individuals which is 82% while the Female respondents have 9 representing 18% of the total. No participants selected “Prefer not to say”, resulting in 0% for that category. This distribution shows that there are more male than females from professional and non-professional respondents.

Table 5

Age Distribution of Respondents

Age	Number of Respondents	Percentage
18-20	2	4.0%
21-30	13	26.0%
31-40	20	40.0%
40 and above	15	30.0%
Total	50	100.0%

Table 5 shows the 50 respondents from the survey, which are categorized into four age groups. The largest count of participants fell within the 31–40 age bracket, resulting to 40.0% (20 respondents) of the sample. This was followed by the 40 and above group, representing 30.0% (15 respondents). The 21–30 age group accounted for 26.0% (13 respondents), while the 18–20 group was the smallest, with 4.0% (2 respondents). This distribution indicates that the majority of respondents are aged between 31 and 40, most of them are professionals and working from CGG Marketing staff.

Table 6

Evaluation Result in terms of Functional Suitability

Sub-characteristics	Non-professionals Rating	Interpretation	Professionals Rating	InterpretIation
Accurately Detects	5.6	Strongly Agree	4.53	Agree
Number Tagging	5.6	Strongly Agree	4.60	Agree
Reduce Human Error	5.6	Strongly Agree	4.67	Agree
Total	5.6	Strongly Agree	4.60	Agree

Table 6 shows the results indicating that CGG Marketing Staff strongly affirm the app's capability to detect and tag objects accurately while minimizing human error. Professionals, while still agreeing overall, rated it slightly lower, possibly due to higher expectations.

Table 7

Evaluation Result in terms of Efficiency

Sub-characteristics	Non-professionals Rating	Interpretation	Professionals Rating	Interpretation
Runs Smoothly	5.3	Strongly Agree	4.53	Agree
Result Real-time	5.3	Strongly Agree	4.57	Agree
No Excessive Battery Drain	5.3	Strongly Agree	4.57	Agree
Total	5.3	Strongly Agree	4.56	Agree

Table 7 shows that CGG Marketing Staff were delighted with the app's performance and responsiveness, noting smooth operation and efficient resource usage. Professionals agreed but offered more moderate ratings.

Table 8

Evaluation Result in terms of Usability

Sub-characteristics	Non-professionals Rating	Interpretation	Professionals Rating	Interpretation
Easy to Navigate	5.4	Strongly Agree	4.53	Agree
Easy to Understand	5.4	Strongly Agree	4.50	Agree
Efficiently Capture	5.4	Strongly Agree	4.23	Agree
Total	5.4	Strongly Agree	4.42	Agree

Table 8 shows that Non-professionals found the app highly user-friendly, highlighting easy navigation and understanding. Professionals also found it usable, though their slightly lower scores may reflect a critical eye for interface design and efficiency.

Table 9

Evaluation Result in terms of Reliability

Sub-characteristics	Non-professionals Rating	Interpretation	Professionals Rating	Interpretation
Object Counting Result	5.5	Strongly Agree	4.60	Agree
Does Not Crash	5.5	Strongly Agree	4.50	Agree
Consistent	5.5	Strongly Agree	4.60	Agree
Total	5.5	Strongly Agree	4.57	Agree

Table 9 shows that the CGG Marketing Staff expressed strong confidence in the app's stability and reliability during use. While professionals also agreed on reliability, they rated it slightly lower, likely due to their broader experience with app testing under varied conditions.

Table 10:

Summary Results

Respondents	Standards	Results	Interpretation
CGG Marketing Staff	Functionality	5.6	Strongly Agree
	Usability	5.4	Strongly Agree
	Reliability	5.5	Strongly Agree
	Efficiency	5.3	Strongly Agree
	Average	5.45	Strongly Agree
IT Professionals	Functionality	4.2	Agree
	Usability	4.5	Agree
	Reliability	4.6	Agree
	Efficiency	4.4	Agree
	Average	4.43	Agree
Construction Professionals	Functionality	5.1	Strongly Agree
	Usability	5	Strongly Agree
	Reliability	4.8	Agree
	Efficiency	5.2	Strongly Agree
	Average	5.03	Strongly Agree
Customers	Functionality	4.3	Agree
	Usability	4	Agree
	Reliability	4.4	Agree
	Efficiency	4.1	Agree
	Average	4.2	Agree

Table 10 shows the evaluation results were derived from the feedback of four respondent groups: CGG Marketing Staff, IT Professionals, Construction Professionals, and Customers. Each group rated the TecTags mobile application based on the ISO/IEC 25010 software quality characteristics: Functionality, Usability, Reliability, and Efficiency, using a 6-point Likert scale.

- CGG Marketing Staff gave the highest ratings across all categories, averaging 5.45, indicating that the application meets their expectations in real-world usage.
- IT Professionals rated the app with an overall average of 4.43, showing general agreement with the application's performance but implying room for technical improvement.
- Construction Professionals also provided strong ratings (5.03 average), validating the system's effectiveness in a construction-related environment.
- Customers, with an average of 4.2, agreed with the app's capabilities but had slightly lower satisfaction levels, possibly due to their academic and less practical perspective.

These findings confirm that TecTags is well-received by end-users and industry professionals, with particularly high approval from non-technical users who benefit most from its features.

Computing Standards and Modern Tools and Techniques Applied

Computing Standards

TecTags follows the ISO 25010 software quality model, which evaluates software performance to ensure a high-quality and reliable mobile application.

Development Tools

Several tools were utilized in the development of TecTags to ensure efficiency and accuracy in object detection and counting.

- **Flutter**: A UI toolkit developed by Google for building natively compiled applications for mobile, web, and desktop from a single codebase. Flutter will be used to create the

front-end interface of the mobile application, providing a smooth and responsive user experience.

- **Python** – The primary programming language used for developing TecTags. Python's extensive libraries, such as PyTorch, enabled seamless integration of object detection and object detection techniques.
- **PyTorch**: An open-source deep learning framework used for implementing and training object detection models. Its dynamic computation graph and ease of debugging make it ideal for research and development. PyTorch will serve as the core framework for building TecTags' object detection and number tagging functionalities.
- **MongoDB**: A NoSQL cloud-based database used to store user-generated data such as processed images, image labels, and related metadata. MongoDB provides efficient storage and fast retrieval, essential for image-heavy applications.
- **Roboflow** – A platform used for dataset management, including annotation, augmentation, and preprocessing of training images. Roboflow helped improve the model's accuracy by enhancing data quality and diversity.

- **Visual Studio Code (VS Code)** – The primary integrated development environment (IDE) used for coding and testing the application. VS Code provided a lightweight yet powerful platform for writing Python scripts, debugging, and integrating various libraries for object detection.

Techniques Applied

Techniques for image preprocessing and augmentation were used to improve object detection accuracy. By increasing the model's resilience, the dataset was resized, flipped, its brightness changed, and noise added. Several training iterations were used to optimize PyTorch, achieving great accuracy by adjusting batch size, epochs, and learning rate parameters. Bounding box detection and instance segmentation were used to achieve object counting, guaranteeing accurate identification of products like Hollow blocks, Deform bars, Bistay sand, Gravel, and Skim coats frequently seen in hardware stores.

Trade-offs

Table 11:

Design Constraints

Design Constraints			
Design	**Model size**	**Battery Usage**	**Latency**
Design 1	20 MB	32 mAh	34 ms
Design 2	27 MB	30 mAh	32 ms
Design 3	40 MB	34 mAh	25 ms

Table 11 shows several trade-offs were considered during the development of TecTags.

- In the Design constraints, this study uses Model Size, Battery Usage, and Latency as the criteria to evaluate the object detection models.
- Model Size refers to the storage footprint of the object detection model, which affects installation size and loading time.

- Battery Usage is measured based on the power consumed during continuous inference on Android devices.
- Latency is the time it takes for the model to process one frame, representing the responsiveness of the detection system.

These values were obtained by testing each design on an Android environment, using PyTorch models, and measuring metrics through Android Studio Profiler and real-device benchmarking.

Sensitivity Analysis

The sensitivity analysis assessed the importance of each criterion. The analysis utilized a scale of 1 to 10 to rank each standard.

1st Combination for Sensitivity Analysis

Table 12:

Sensitivity Analysis using 10-9-8 Criterion Rank

Criteria	Weight	PyTorch	YOLO	TensorFlow Lite	OpenCV
Total App Size (MB)	10	11	9	7.5	12
Memory Consumption (MB)	9	9	8	9.5	10
Response Time (Seconds)	8	9.5	8.5	9	7.5
Weighted Total Score		255	230	232.5	270

Table 12 shows that the combination 10-9-8 prioritized app size the most, followed by memory consumption and response time. In Table 6, PyTorch achieved the highest overall score

of 255, indicating that it effectively balances storage efficiency, memory usage, and response time, outperforming other frameworks under this prioritization.

2nd Combination for Sensitivity Analysis

Table 13:

Sensitivity Analysis using 8-9-10 Criterion Rank

Criteria	Weight	PyTorch	YOLO	TensorFlow Lite	OpenCV
Total App Size (MB)	8	11	9	7.5	12
Memory Consumption (MB)	9	9	8	9.5	10
Response Time (Seconds)	10	9.5	8.5	9	7.5
Weighted Total Score		249	229	235.5	261

Table 13 shows the combination 8-9-10 placed the most importance on response time, followed by memory consumption and app size. In Table 7, PyTorch again ranked the highest with a score of 249, demonstrating superior performance in terms of fast response time while maintaining low memory and storage impact.

3rd Combination for Sensitivity Analysis

Table 14:

Sensitivity Analysis using 9-10-8 Criterion Rank

Criteria	Weight	PyTorch	YOLO	TensorFlow Lite	OpenCV
Total App Size (MB)	9	11	9	7.5	12
Memory Consumption (MB)	10	9	8	9.5	10
Response Time (Seconds)	8	9.5	8.5	9	7.5
Weighted Total Score		253	229	234.5	268

Table 14 shows the 9-10-8 weighting emphasized memory consumption first, then app size and response time. According to Table 8, PyTorch maintained the top position with a score of 253, suggesting that it is highly suitable for applications requiring optimized memory handling.

4th Combination for Sensitivity AnalysisF

Table 15:

Sensitivity Analysis using 10-8-9 Criterion Rank

Criteria	Weight	PyTorch	YOLO	TensorFlow Lite	OpenCV
Total App Size (MB)	10	11	9	7.5	12
Memory Consumption (MB)	8	9	8	9.5	10
Response Time	9	9.5	8.5	9	7.5

(Seconds)					
Weighted Total Score		254	230.5	232	267.5

Table 15 illustrates the 10-8-9 combination, app size was weighted most heavily, followed by response time and memory. Based on Table 9, PyTorch scored 254, maintaining its lead as the most efficient choice, particularly for scenarios constrained by storage and response speed.

5th Combination for Sensitivity Analysis

Table 16:

Sensitivity Analysis using 9-8-10 Criterion Rank

Criteria	Weight	PyTorch	YOLO	TensorFlow Lite	OpenCV
Total App Size (MB)	9	11	9	7.5	12
Memory Consumption (MB)	8	9	8	9.5	10
Response Time (Seconds)	10	9.5	8.5	9	7.5
Weighted Total Score		251	230	233.5	263

Table 16 shows that this ranking prioritized response time the most, with slightly lower emphasis on app size and memory. As seen in Table 10, PyTorch led again with a score of 251, reinforcing its edge in delivering fast performance without sacrificing efficiency.

6th Combination for Sensitivity Analysis

Table 17:

Sensitivity Analysis using 8-10-9 Criterion Rank

Criteria	Weight	PyTorch	YOLO	TensorFlow Lite	OpenCV
Total App Size (MB)	8	11	9	7.5	12
Memory Consumption (MB)	10	9	8	9.5	10
Response Time (Seconds)	9	9.5	8.5	9	7.5
Weighted Total Score		250	228.5	236	263.5

Table 17 shows the final combination placed memory consumption first, followed by response time and app size. Table 11 reveals that PyTorch continued its dominance with a score of 250, confirming its robustness and adaptability across diverse resource priorities. PyTorch was the optimal choice for TecTags, offering the best performance across app size, memory usage, and response time, ensuring an efficient and user-friendly mobile application for object detection and counting in hardware stores.

Multiple Constraints

- **Scope and Time**

Given the limited timeframe for development and testing, the project team focused on building the core functionalities of TecTags, specifically, image-based object counting and number tagging using PyTorch. Due to time, additional features such as cloud-based inventory were halted for future development.

- **Time and Resources**

With four of us as a development team, the project scope had to be managed so we could deliver the application on time; the team prioritized important features and conducted regular reviews and feedback for consultations with IT professionals. Object classification and real-time tracking were simplified or removed to match the team's remaining time before the defense.

- **Hardware Compatibility and Device Limitations**

TecTags runs object detection on mobile devices to ensure smooth performance and was tested on Android phones. Older devices may experience slow processes or limited functionality. The app was mainly optimized for mid-range Android phones used in hardware store settings.

- **Dataset and Model Training Constraints**

The team used Roboflow and Visual Studio Code to prepare and train the detection model, but limited GPU availability and dataset size posed constraints. As a result, the model was trained on a curated set of construction and hardware materials, with general counting accuracy prioritized over recognizing a wide variety of object types. Additional training and data collection would be needed to expand the app's detection capabilities in the future.

Chapter 5

Summary, Conclusion, and Recommendations

Summary

The study aimed to develop TecTags: An Automated Object Counting Mobile Application Utilizing Object Detection Algorithms, which offers an efficient solution for inventory management within the hardware industry. The application leverages PyTorch for object detection and counting, enabling users to capture images of materials and automatically generate accurate inventory records. TecTags reduces human errors and streamlines inventory tracking processes by integrating number tagging and real-time object recognition. The development of TecTags involved several processes, including image preprocessing, data augmentation, and model training, ensuring high object detection accuracy.

The application was designed with a user-friendly interface, facilitating easy navigation and using its features by construction personnel. Furthermore, the integration of cloud storage allows users to back up and retrieve inventory data effortlessly. Evaluation results indicate that TecTags meets quality standards regarding functionality, reliability, efficiency, usability, and maintainability. User feedback was predominantly positive, highlighting the application's effectiveness in enhancing inventory accuracy and alleviating workload.

With its automated counting system and mobile accessibility, TecTags presents a practical and scalable solution for construction firms managing substantial quantities of materials. Overall, the study successfully met its objectives, demonstrating that AI-driven object detection can significantly enhance inventory management. The application adheres to international software quality standards and presents opportunities for further improvements, including cross-platform availability, enhanced object recognition, and real-time collaboration features in future updates.

Conclusion

The development of TecTags: An Automated Object Counting Mobile Application Utilizing Object Detection Algorithms successfully achieved its primary objective of creating an Android-based mobile application for the construction industry that accurately performs automatic object counting and number tagging from images captured by mobile phone cameras. The application effectively utilized object detection algorithms to automate the counting and tagging of stock items, thereby enhancing the efficiency of inventory management processes. Specific objectives were met, including developing a mobile application that implements advanced object detection algorithms and evaluating the application using ISO 25010 standards. The evaluation confirmed that TecTags met the expected software quality standards regarding functionality, reliability, usability, and efficiency. User feedback indicated that the application provided an intuitive interface, streamlined workflows, reduced time spent on manual counting, and minimized discrepancies in stock records.

The application was evaluated using the ISO/IEC 25010 software quality framework, focusing on Functionality, Usability, Reliability, and Efficiency. Feedback was collected from 50 respondents, including IT professionals, Customers, Construction Professionals, and CGG Marketing staff. Results indicated a strong level of agreement regarding the usefulness, performance, and user-friendliness of the application, especially from non-technical users who benefit most from its features. While IT professionals and customers provided constructive ratings, they generally agreed that the system met its technical expectations, albeit with room for further enhancement.

With its mobile accessibility and cloud storage integration, TecTags ensured that inventory data could be easily stored, retrieved, and updated in real-time. The application also

holds potential for further enhancements, such as cross-platform compatibility, improved object recognition, and real-time collaboration features, which would expand its usability across various industries.

In summary, TecTags represents a significant innovation in inventory management within the construction sector, demonstrating that AI-driven solutions can effectively improve traditional processes, making them more efficient, precise, and scalable. Future research and development efforts should focus on expanding the application's capabilities and integrating additional features to maximize its impact across multiple domains.

Recommendation

Future researchers may reference this study as a foundation for developing other AI-powered mobile applications focused on object detection and inventory management. The findings of this research establish a basis for enhancing machine learning-based inventory solutions that can serve diverse industries. Additionally, researchers are encouraged to consider enhancements to TecTags by integrating advanced features, optimizing detection algorithms, and employing supplementary tools and techniques.

This study advocates for further research in the following areas:

1. **Enhancing Object Detection Accuracy:** Future investigations should improve object detection accuracy by expanding the dataset to include various images captured under different lighting and environmental conditions.
2. **Optimizing Performance and Efficiency:** Research efforts should reduce processing time while maintaining high levels of detection accuracy, ensuring

smooth operation across a broad spectrum of mobile devices with varying hardware capabilities.

3. **Expanding Application Features:** Future developments could include implementing cross-platform support for both Android and iOS. Additionally, incorporating real-time collaboration and cloud-based data sharing would significantly improve inventory management capabilities.

By pursuing these avenues, researchers can contribute to advancing AI-driven applications, further enhancing the effectiveness of inventory management solutions across multiple sectors.

References

Deploying PyTorch models to IOS and Android for Real-Time Applications - Sling Academy. (n.d.-b). https://www.slingacademy.com/article/deploying-pytorch-models-to-ios-and-android-for-real-time-applications/

Matani, D. (2022, January 4). Deep Learning on your phone: PyTorch Lite Interpreter for mobile platforms. *Medium*. https://medium.com/data-science/deep-learning-on-your-phone-pytorch-lite-interpreter-for-mobile-platforms-ae73d0b17eaa

Kirchheim, K., Filax, M., & Ortmeier, F. (2022). *PyTorch-OOD: a library for Out-of-Distribution Detection based on PyTorch*. https://openaccess.thecvf.com/content/CVPR2022W/HCIS/html/Kirchheim_PyTorch-OOD_A_Library_for_Out-of-Distribution_Detection_Based_on_PyTorch_CVPRW_2022_paper.html

M, M., & V, H. (2024). DETECTING OBJECTS ON SATELLITE IMAGES USING PYTORCH. *International Scientific and Technical Conference Information Technologies in Metallurgy and Machine Building*, 508–511. https://doi.org/10.34185/1991-7848.itmm.2024.01.098

Carey, R. (2022, October 25). *5 Ways Mobile Technology Can Benefit the Construction Industry*. https://www.asite.com/blogs/5-ways-mobile-technology-can-benefit-the-construction-industry

Alviar, R. P., Casuat, C. D., & Belizar, J. P. (2024b, February 12). *Mobile Technology for Improving Project Management in the Construction Industry - Philippine Setting [Online forum post]*. https://doi.org/10.46254/an14.20240042

Jenkins, A. (2022b, December 16). *20 Inventory Management Challenges and Solutions for 2022 and beyond*. Oracle NetSuite. https://www.netsuite.com/portal/resource/articles/inventory-management/inventory-management-challenges.shtml

Computer Vision for inventory counting in 2025. (n.d.-b). https://www.scnsoft.com/scm/inventory-counting-with-computer-vision

Kyanon.Digital. (2025, March 1). Revolutionizing industry with object detection in AI Technology - 2025. *Kyanon Digital*. https://kyanon.digital/revolutionizing-industry-with-object-detection-in-ai-technology/

Easier object detection on mobile with TensorFlow Lite. (n.d.-b). https://blog.tensorflow.org/2021/06/easier-object-detection-on-mobile-with-tf-lite.html

Morgunov, A. (2023, August 29). *How to train your own object detector using TensorFlow Object Detection API*. neptune.ai. https://neptune.ai/blog/how-to-train-your-own-object-detector-using-tensorflow-object-detection-api

Qin, D., Leichner, C., Delakis, M., Fornoni, M., Luo, S., Yang, F., Wang, W., Banbury, C., Ye, C., Akin, B., Aggarwal, V., Zhu, T., Moro, D., & Howard, A. (2024b, April 16). *MobileNetV4 -- Universal models for the mobile ecosystem*. arXiv.org. https://arxiv.org/abs/2404.10518

EdjeElectronics. (n.d.). *GitHub - EdjeElectronics/TensorFlow-Lite-Object-Detection-on-Android-and-Raspberry-Pi: A tutorial showing how to train, convert, and run TensorFlow Lite object detection models on Android devices, the Raspberry Pi, and more!* GitHub. https://github.com/EdjeElectronics/TensorFlow-Lite-Object-Detection-on-Android-and-Raspberry-Pi

Győrödi, C. A., Dumşe-Burescu, D. V., Zmaranda, D. R., & Győrödi, R. Ş. (2022). A Comparative Study of MongoDB and Document-Based MySQL for Big Data Application Data Management. *Big Data and Cognitive Computing*, *6*(2), 49. https://doi.org/10.3390/bdcc6020049

Goli, V. R. (2021). REACT NATIVE EVOLUTION, NATIVE MODULES, AND BEST PRACTICES. *INTERNATIONAL JOURNAL OF COMPUTER ENGINEERING & TECHNOLOGY*, *12*(2), 73–85. https://doi.org/10.34218/ijcet_12_02_009

Top 10+ Best hardware stores in Quezon City, Philippines. (n.d.). Philippines Business Directory. https://www.businesslist.ph/category/hardware-stores/city%3Aquezon-city

Bandyopadhyay, H. (n.d.). Image Annotation: Definition, Use Cases & Types [2024]. *V7*. https://www.v7labs.com/blog/image-annotation-guide

Appendices

Appendix A

Questionnaire

Table 18:

Questionnaire for IT Professionals / Construction Professionals / Customers

TECHNOLOGICAL INSTITUTE OF THE PHILIPPINES - QUEZON CITY
College of Computer Studies

Evaluation Questions For "TecTags: An Automated Object Counting Mobile Application using Object Detection Algorithms."

Name of Respondent: _____ Date: _____

"We are committed to protecting your privacy and ensuring transparency about how we collect, use, and safeguard the information you provide through our surveys. This paper is for research purposes only."

Criteria	**Rating**					
	(6) Strongly Agree	(5) Agree	(4) Slightly Agree	(3) Slightly Disagree	(2) Disagree	(1) Strongly Disagree
Functional Suitability						
The application accurately detects and counts objects.						
The number-tagging feature is precise and reliable.						
The app effectively reduces human error in inventory counting.						
Performance Efficiency						
The application runs smoothly on supported devices.						
The object detection algorithm provides results in real-time.						

The app does not cause excessive battery drain or device overheating.						
Usability						
The user interface is intuitive and easy to navigate.						
Instructions and features are easy to understand.						
Users can efficiently capture and upload images for object detection.						
Reliability						
The application consistently delivers accurate object counting results.						
The app maintains stability and does not crash frequently.						
The results remain consistent across different lighting and object arrangements.						

Table 19:

Questionnaire for CGG Marketing Staff

TECHNOLOGICAL INSTITUTE OF THE PHILIPPINES - QUEZON CITY
College of Computer Studies

Evaluation Questions For "TecTags: An Automated Object Counting Mobile Application using Object Detection Algorithms."

Name of Respondent: _____ Date: _____

"We are committed to protecting your privacy and ensuring transparency about how we collect, use, and safeguard the information you provide through our surveys. This paper is for research purposes only."

Criteria	**Rating**					
	(6) Lubos na Sumasang-ayon	(5) Sumasang-ayon	(4) Medyo Sumasang-ayon	(3) Medyo Hindi Sumasang-ayon	(2) Hindi Sumasang-ayon	(1) Lubos na Hindi Sumasang-ayon
Functional Suitability						
Tama ba ang bilang ng TecTags app sa mga materyales na kinukunan ng larawan?						
Nananatiling wasto ba ang bilang kahit magkadikit-dikit ang mga bagay?						
Mabilis bang maibigay ng app ang resulta ng bilang matapos kunan ng larawan?						
Performance Efficiency						
Mabilis bang maibigay ng app ang resulta ng bilang matapos kunan ng larawan?						
Bumabagal ba ang pagbilang ng app kung papalawakin ang area ng inii-scan?						
Gumagana ba nang maayos						

ang app sa iba't ibang modelo ng cellphone?						
Usability						
Madali bang maintindihan ang resulta ng bilang habang nagtatrabaho?						
Madali bang matutunan at gamitin ang app kahit walang masyadong training?						
Madali bang gamitin ang app sa mga karaniwang gawain sa imbentaryo?						
Reliability						
Ang application ay palaging gumagana nang tama at walang mga error.						
Ang application ay matatag at hindi madalas nagka-crash.						
Ang mga resulta at impormasyon na ibinibigay ng application ay palaging tama.						

Appendix B

Risk Management Plan

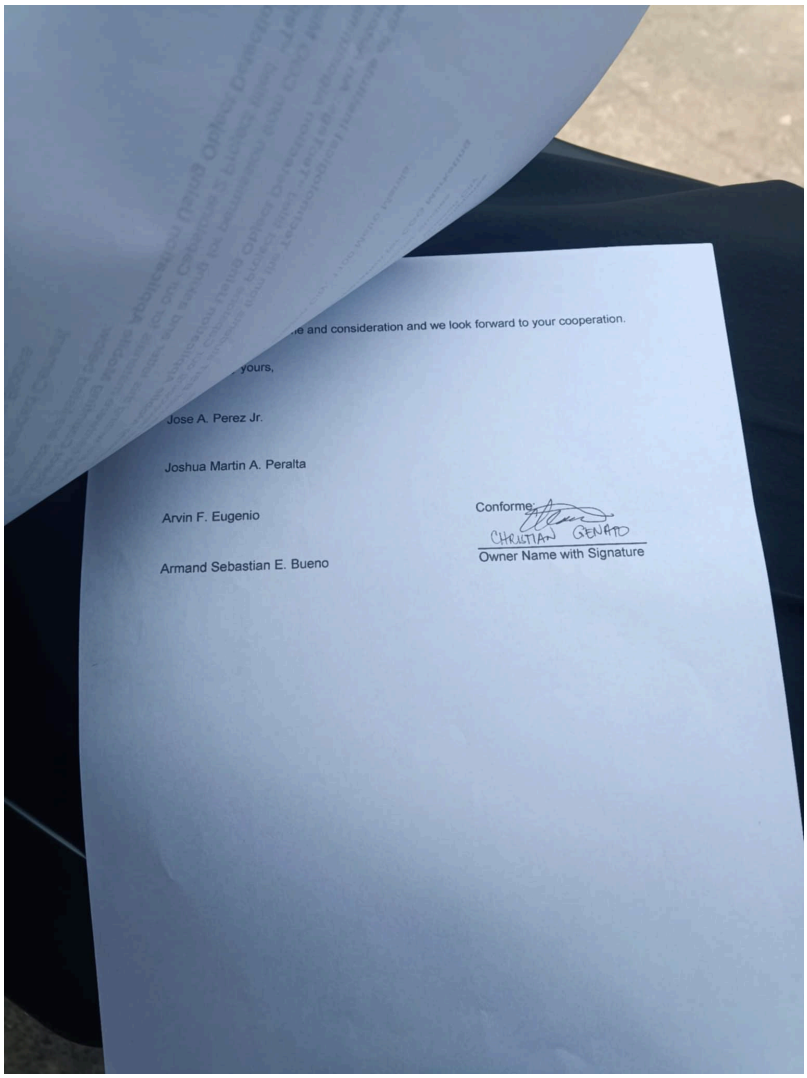
RISKS	LIKELIHOOD (Low/Medium/High)	IMPACT (Low/Medium/High)	MITIGATION STRATEGIES
Errors in object counting leading to inaccurate inventory records	Medium	High	• Improve the object detection model using high-quality datasets. • Conduct extensive testing in various warehouse environments. • Allow users to manually adjust results for verification.
App crashes or lags on certain mobile devices	Low	High	• Perform extensive compatibility testing across different Android devices. • Optimize PyTorch model for better performance. • Implement crash reporting tools to track and fix issues.
Difficulty detecting overlapping or cluttered objects	Medium	High	• Enhance image preprocessing techniques such as contrast adjustment and contour detection. • Train the AI model with diverse datasets, including cluttered object arrangements. Allow manual corrections for users.
Unauthorized access or data breaches	Low	High	• Encrypt sensitive user data and implement secure authentication methods. • Conduct regular security audits. • Comply with data protection regulations to safeguard inventory records.

User dissatisfaction due to complex UI or hard-to-understand results	Low	Medium	• Conduct usability testing and gather feedback from target users. • Simplify the app's navigation and visualization of results. • Provide user tutorials and help sections.
Slow processing time when analyzing large quantities of objects	Medium	High	• Optimize the object detection algorithm for speed without sacrificing accuracy. • Use efficient PyTorch models and limit unnecessary computations. • Perform load testing to identify performance bottlenecks.
Poor adoption rate due to lack of awareness or perceived usefulness	Low	Medium	• Run targeted marketing campaigns to demonstrate TecTags' benefits. • Offer training sessions for construction and hardware store staff. • Encourage early adopters to share feedback and success stories.
Accidental deletion or loss of scanned inventory records	Low	High	• Implement automatic data backups and cloud storage options. • Include a recovery option for accidentally deleted records. • Provide warnings before data deletion actions.
Changes in construction industry standards affecting the app's usefulness	Low	Medium	• Stay updated with industry trends and modify features as needed. • Maintain close communication with construction professionals for feedback. Keep the app adaptable for future enhancements.

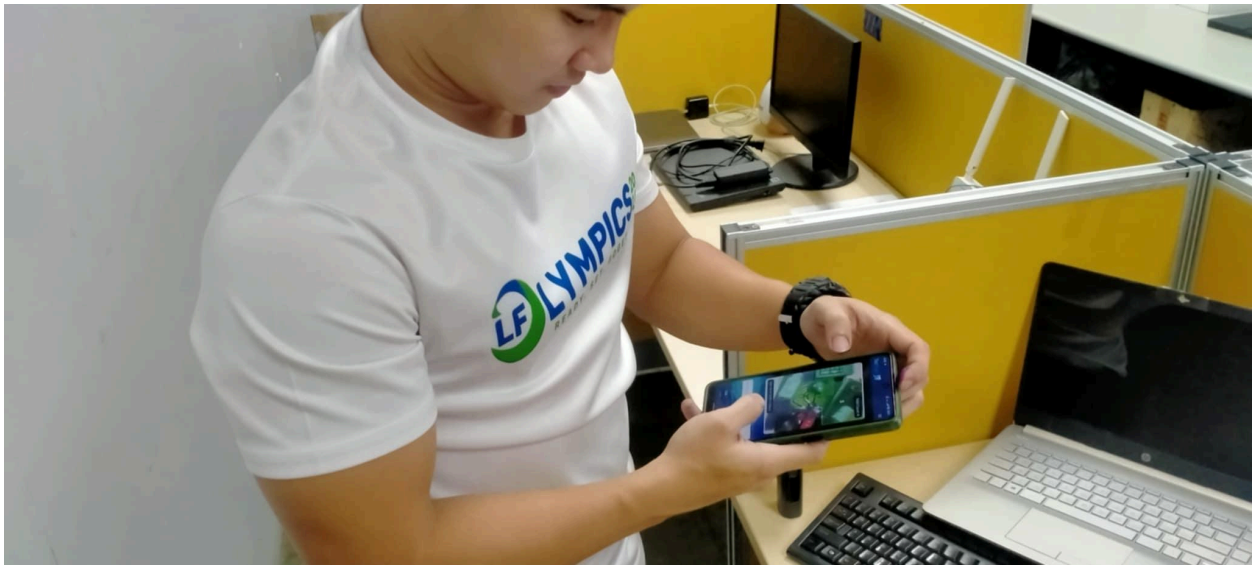
Appendix C

Proof of Evaluation

CGG Marketing Staff

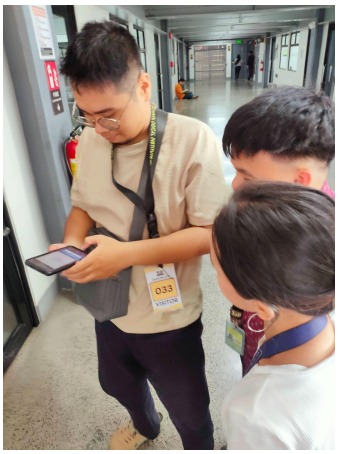

and consideration and we look forward to your cooperation.
yours,
Jose A. Perez Jr.
Joshua Martin A. Peralta
Arvin F. Eugenio
Armand Sebastian E. Bueno
Conforme:
CHRISTIAN GENATO
Owner Name with Signature

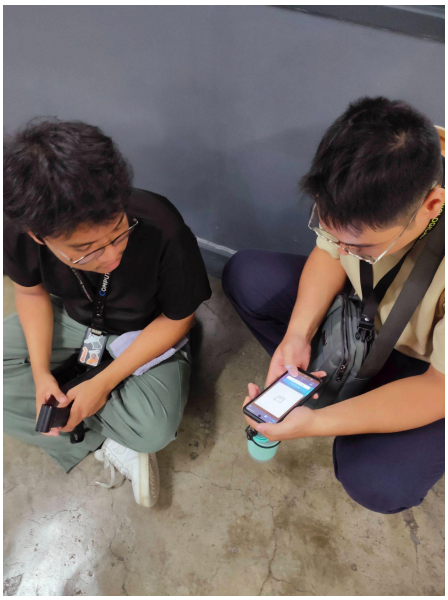
IT Professional



Construction Professional

Customers





Appendix D

Transcript of Interview

About the supplies:

Arvin: 1 minute or 2 minutes lang po. Icoconfirm lang po namin yung sa seven na mabentang products po sa inyo.

Arvin: Cement, Bistay Sand, Hollow blocks, Skim Coat, Gravel, Deformed bar. Tama po ba na eto yung mga mabebenta?

Gemma: Oo

Arvin: Tapos back then po kasi nagtanong kami din po kami bakit di po like namin mapicturan yung ibang product. Dahil po ba kaya siyang mabilang like onti lang po yung stocks nya ganun po ba ma'am

Gemma: Yung ibang items?

Arvin: Opo

Gemma: Dahil hindi nyo kaya mabilang?

Arvin: I mean kayo po na parang sabi po kasi ni boss Jayson dati

Gemma: Ano?

Arvin: Wag na daw po idagdag kasi kayang kaya naman po bilangin

Gemma: Oo, yung mga ganyan, mga pintura kaya nyang bilangin

Arvin: Opo ma'am. Yun lang po ma'am. Thank you po sa time ma'am

About the process:

Arvin: Sa inyo po ba yung mga binaba or binibilang?

Jayson: Pag baba diba dinedeliver na naka box. Ibaba yan tapos pinapa separate ko para pag dedeliver sya madali lang yung pagbilang

Arvin: Tapos ayun after po nung sa kunwari bibili po kami like paano po kaya yung process, kunwari bumili po kami ng limang cement bibilangin nyo pa po ba yung stock nyo after?

Jayson: Di na kami nagbibilang. Out in out lang kami

Arvin: In out lang po talaga yung sa process niyo

Jayson: Oo

Arvin: Sige po

Jayson: Katulad nyan yung mga nagkakarga na yan. Yung mga pumasok na yan. May monthly inventory kasi yan

Arvin: So pagpasok alam nyo kunwari example pag pasok 200 pieces na ganito alam nyo na po yung gagawin

Jayson: Oo

Arvin: Pero pag bibili na po kami

Jayson: Pag inaa out. Di pinapakita yung mga nakukuha. Wala kami kinikwenta kung ilan pa yung ibebenta kasi araw araw din

Arvin: Parang straightforward po na bibili kami. Di niyo na po ichecheck yung stocks

Jayson: Maiikot lahat to ng checker. Ako checker iikot lang ako dito. Pag wala ng stock, may bumili nasainyo kung oorder pa kayo tapos to follow na lang para dun sa darating na order

Arvin: Ang question ko po na next, paano po kapag wala na kayo stock boss? Tyaka na lang kayo magrere order?

Jayson: Oo, kahit konti na lang yung sa cement. 200 yung stock, pag bente na lang oorder na lang ulit

Arvin: Okay po. Thank you po boss