PROJECT REPORT

ON

“TracelQ”

Intel Unnati Industrial Training 2025

WORLD COLLEGE OF TECHNOLOGY AND MANAGEMENT
GURGAON (HARYANA), INDIA

Team Name : InnoVision

PROJECT MENTOR :

Dr. Preeti
(Assistant Professor)
CSE DEPARTMENT

TEAM MEMBER :

Nikita (MCA)
Sarveshwari (MCA)
Raushan Kumar (MCA)

PROBLEM STATEMENT

Design and implement a smart, automated system for product labelling and traceability, capable of verifying product quality parameters and applying or validating labels that include:

- Device ID
- Batch ID
- Manufacturing Date
- RoHS Compliance
- Serial Number (QR or barcode)
- The system will leverage:
- Mechatronic systems for sensor-based inspection, actuation, and real-time control.
- AI models for dynamic verification and validation of label data and product compliance.
- Automated labelling or data printing mechanisms (simulated or real).
- Data logging for traceability and audit purposes

TEAM MEMBERS WORK

Nikita: Mechatronics & Embedded System Simulation

Responsibilities :

- Design the conveyor sensor-actuator system.
- Simulate sensors (e.g., camera, IR sensors) and actuators (label printer, reject mechanism).
- Program microcontroller logic using Tinkercad Circuits or Proteus.
- Implement control logic to detect product, verify data, trigger labeling or rejection.
- Simulate real-time hardware interactions (sensor input → processing → actuator output).

Tools :

- Tinkercad Circuits
- Proteus (optional)
- Arduino IDE (simulated code)

Sarveshwari Singh: AI / Machine Learning Integration, Label Verification & Backend

Responsibilities:

- Implement AI-based label inspection using OCR or image recognition.
- Use OpenCV + EasyOCR in Google Colab for scanning QR codes/barcodes and verifying print quality.
- Train or apply ML models to detect defective or invalid labels.
- Develop algorithms to match product metadata (Device ID, Batch ID, RoHS compliance).
- Generate and validate simulated labels (QR/barcode) in Python.
- Connect frontend UI with backend logs or simulated data for demo.

Tools:

- Google Colab + OpenCV + EasyOCR
- Python (for label generation and AI model integration)
- Optional: LabelImg + YOLOv5 for object detection training
- Python + Firebase

Raushan Kumar: UI/UX Design & Traceability Database

Responsibilities:

- Design UI/UX for control panel, inspection dashboard, and traceability log.
- Build frontend interface showing real-time product status, label verification results, and rejection alerts.
- Implement traceability logging system using SQLite or CSV files.
- Enable search/filter/export features for product inspection data.

Tools:

- Flutter (for UI/UX prototyping)
- React.js / HTML + CSS + JS (for frontend development)
- SQLite (for Product database)

TABLE OF CONTENTS

Chapter-1

INTRODUTION

In modern manufacturing environments, ensuring product authenticity, quality, and regulatory compliance is crucial. To address these needs, a smart, automated system for product labelling and traceability is proposed, integrating advanced mechatronic systems, AI technologies, and data-driven controls. This system is designed to verify critical product parameters and apply or validate labels that include vital information such as Device ID, Batch ID, Manufacturing Date, RoHS Compliance, and a unique Serial Number encoded as a QR code or barcode. Mechatronic components—including sensors, actuators, and real-time controllers—will detect and inspect product characteristics to confirm quality standards are met. AI models will enhance the process by dynamically verifying label data and ensuring regulatory compliance before the labelling process proceeds. Once validated, the system either prints the label directly or confirms pre-applied labels using optical recognition systems. Additionally, all product and inspection data will be logged into a central database for traceability and audit purposes, ensuring transparency across the supply chain. This solution not only improves efficiency and accuracy in manufacturing lines but also reduces the potential for human error, supports compliance with global standards, and enhances consumer confidence through verified product information.

Chapter-2

SRS Report (Software Requirement Specifications) Report

This Software Requirements Specification (SRS) document outlines the design and functional expectations of a smart, automated system for product labelling and traceability. The primary goal of this system is to ensure accurate, efficient, and regulatory-compliant labelling of manufactured products while maintaining traceability and supporting quality verification using advanced technologies.

1. Introduction

The proposed system is designed for automated identification, inspection, and labelling of products on a production line. It integrates mechatronic components for sensing and actuation, artificial intelligence (AI) models for validation and decision-making, and automated printing mechanisms for applying product labels. Each label contains critical information, including Device ID, Batch ID, Manufacturing Date, RoHS compliance status, and a unique Serial Number encoded as a QR code or barcode. This information is essential for product traceability, regulatory compliance, and quality assurance.

2. System Overview

The system operates as an intelligent inspection and labelling module that can be embedded into a larger production workflow. Products moving along a conveyor belt will be inspected by various sensors to ensure they meet pre-defined quality parameters. These may include weight, shape, color, dimensions, surface finish, and other physical or electronic properties. Mechatronic systems are used to trigger the inspection process, manage product positioning, and activate labelling or rejection actuators based on the results.

AI models process data from sensors and camera inputs to detect compliance with quality standards and regulatory requirements like RoHS (Restriction of Hazardous Substances). These models also verify that the data to be printed or already printed on the label is correct, dynamically adjusting in case of mismatches or anomalies.

3. Functional Requirements

- **Product Identification and Verification**: Each product is uniquely identified using a Device ID and Batch ID retrieved from the internal database or sensor input.
- **Sensor-Based Inspection**: Sensors inspect the physical and/or functional quality of the product. Criteria may include shape, size, surface integrity, or embedded electronic features.
- **AI-Based Validation**: An AI engine checks whether the collected data aligns with expected quality and compliance parameters (including RoHS compliance).
- **Label Generation**: Upon successful validation, the system generates a label containing Device ID, Batch ID, Manufacturing Date, RoHS status, and a QR/barcode representing a unique Serial Number.

- **Label Application or Validation**: The system either applies the label via a printing mechanism or scans a pre-applied label to verify its accuracy.
- **Rejection Mechanism**: Products failing inspection or validation are automatically rejected via actuators.
- **Data Logging and Traceability**: All operations, inspection results, and product details are logged into a secure database for traceability, auditing, and future analytics.

4. Non-Functional Requirements

- **Scalability**: The system should support high-speed processing for large-scale manufacturing lines.
- **Reliability**: Accuracy in inspection and label application must be above 99%.
- **Security**: Product and batch data must be securely stored and protected from unauthorized access.
- **Usability**: The interface should be intuitive for operators and engineers to configure, monitor, and troubleshoot.
- **Maintainability**: Modular design allows for easy updates to sensors, AI models, or printing mechanisms.

5. Hardware & Software Components

- **Sensors**: IR sensors, cameras, weight sensors, or proximity sensors for quality inspection.
- **Actuators**: Label printer mechanism, pneumatic rejectors, and conveyor belt motors.
- **Microcontroller/PLC**: For real-time control of mechatronic components.
- **AI Engine**: Deployed locally or on the cloud to verify compliance and label integrity.
- **Database**: SQL/NoSQL-based logging of product information, inspection results, and label data.
- **User Interface**: Dashboard to configure product parameters, view real-time process status, and generate reports.

6. Conclusion

This SRS describes a robust, smart automated labelling and traceability system capable of enhancing product quality assurance and compliance tracking in modern manufacturing. The integration of mechatronic inspection, AI-based validation, automated labelling, and comprehensive data logging ensures operational efficiency, reduces human error, and supports global regulatory standards.

Chapter 3

Mechatronics and Embedded System Simulation

1. Introduction

This chapter outlines the simulation of mechatronics and embedded control logic used in the Smart Automated Product Labeling and Traceability System. It focuses on the design and implementation of a virtual hardware setup using Tinkercad Circuits and the Arduino platform. The system simulates an industrial conveyor process where products are detected, evaluated by an external AI module, and subsequently labeled or rejected based on the outcome.The simulation bridges the gap between software-based AI decisions and hardware actions by mimicking the real-world behavior of sensors, actuators, and communication systems. This forms a vital subsystem of the overall project, enabling real-time, decision-driven mechanical responses in a safe and testable environment.

2. Objectives

The main goals of the mechatronics and embedded system simulation are:

- To simulate a real-time conveyor system that detects the presence of products.
- To build a control logic system using Arduino that can respond to AI-based validation.
- To simulate hardware actuators such as labeling and rejection mechanisms.
- To establish and test serial communication between the embedded system and the external software module.
- To model how physical machines can interact with smart decisions to achieve automation.

3. Tools and Technologies

Tinkercad Circuits: Virtual environment to simulate Arduino-based circuits

Arduino IDE: Platform used to write, upload, and simulate embedded control logic

IR Sensor (Simulated): Detects product presence on the conveyor

Servo Motors (2x): Act as actuators: one for labeling, one for rejecting products

Arduino Uno: Core microcontroller for processing inputs and controlling outputs

Serial Communication: Interface for connecting with external Python-based AI module

4. System Design and Architecture

The simulated system replicates an automated labeling unit where products arrive periodically, are evaluated for correctness, and then either receive a label or are rejected.

4.1 Components Involved

Simulated IR Sensor:

Functions as a product detector on the conveyor belt. It triggers an event each time a product is assumed to arrive.

Arduino Uno Microcontroller:

Receives input from the IR sensor, sends data to the AI module, and activates the respective actuator based on the returned decision.

Labeling Servo Motor:

Activates when the product is accepted by the AI module. Simulates the action of label application.

Rejection Servo Motor:

Activates when the product is rejected, simulating a mechanical arm that removes the faulty product from the belt.

Serial Communication Interface:

Enables command-based data transfer between the Arduino and a Python-based AI module.

4.2 Functional Workflow:

A product is simulated to arrive using a time-based trigger instead of a physical IR sensor.

The microcontroller sends a predefined command to the AI model (e.g., "CAPTURE_IMAGE").

The AI system evaluates the simulated product's label or condition.

The AI sends a response back via serial (e.g., "ACCEPT" or "REJECT").

Based on the response:

If "ACCEPT" → Labeling motor is activated.

If "REJECT" → Reject arm motor is triggered.

System resets and waits for the next product cycle.

5. Simulation Behavior and Observations:

The system was developed and tested in Tinkercad Circuits, where both servos and the IR sensor were represented virtually. The simulation followed a 10-second product cycle, where every 10 seconds, the system assumed a new product had arrived.

Key behaviors observed:

The labeling servo only responded when the AI model returned an "ACCEPT" response.

The system continuously looped, handling each new simulated product without interruption.

Serial communication functioned effectively, simulating real-world communication between embedded hardware and smart decision engines.

6. Outcome and Evaluation

The embedded simulation successfully demonstrated a fully functional prototype of a smart labeling and rejection system. The Arduino logic was able to:

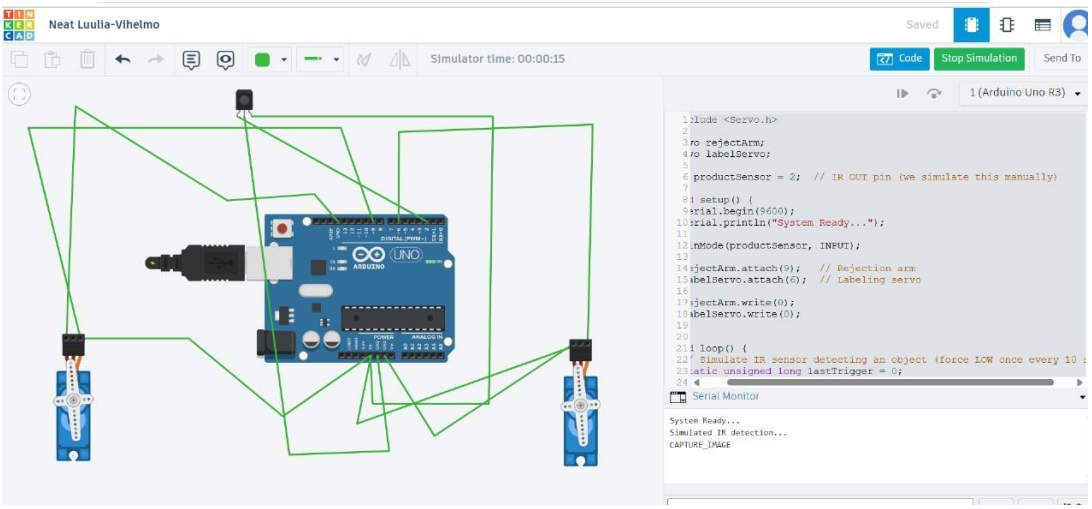
- Detect incoming products
- Interact with a software AI model
- Execute mechanical responses based on real-time decisions
- Maintain a consistent product cycle with reset and repeat behavior
- This system forms a critical part of the larger project by representing the physical execution layer — proving that smart software can directly control hardware actions.

The approach also ensures that the same embedded logic can be transferred to physical hardware (using real IR sensors, servos, and motor drivers) with minimal changes.

7. Conclusion

This simulation proved that mechatronics systems and embedded logic can be efficiently controlled and tested in a virtual environment. Using free tools like Tinkercad Circuits and Arduino IDE, a realistic prototype of a factory automation system was built, enabling safe and cost-effective testing of automation workflows.By integrating with software decisions via serial communication, this system lays the groundwork for fully automated industrial processes, combining AI intelligence with mechanical precision.

Circuit Configuration:



Chapter 4

AI / Machine Learning Integration, Label Verification & Backend

Introduction

This project focuses on implementing an intelligent system for automated label inspection and verification in manufacturing using AI and Machine Learning. It ensures that product labels — containing key data such as Device ID, Batch ID, Manufacturing Date, Serial Number (QR/barcode), and RoHS compliance — are accurate, readable, and valid. The system uses Python tools and cloud services to scan, validate, and log label data in real-time. Integration of image processing, OCR, and backend connectivity allows seamless validation and traceability for production lines aiming at Industry 4.0 compliance.

1. AI-Based Label Inspection Using OCR/Image Recognition

- Uses EasyOCR to extract printed text such as device ID, batch number, and RoHS compliance from label images.
- Combined with OpenCV to pre-process images (e.g., grayscale, thresholding, noise reduction) for enhanced OCR accuracy.
- Detects and reads both English and multi-language labels.

2. Scanning QR Codes/Barcodes & Print Quality Verification

- OpenCV handles detection and decoding of QR codes and barcodes.
- EasyOCR or ZBar library can verify print quality by checking clarity, alignment, and completeness of the code.
- Ensures codes are machine-readable and visually intact.

3. ML Models for Defective/Invalid Label Detection

- Optionally uses YOLOv5 for object detection to spot print defects such as smudging, skew, or missing elements.
- LabelImg helps annotate custom datasets for training.
- Classifies labels as valid or defective using pre-trained or custom-trained models.

4. Metadata Matching Algorithms

- Python algorithms verify that extracted label content matches expected metadata like:
    1. Device ID
    2. Batch ID
    3. Manufacturing Date
    4. RoHS compliance
- Ensures each product is properly identified and compliant.

5. Simulated Label Generation (QR/Barcode)

- Generates sample QR codes and barcodes using Python libraries like qrcode or python-barcode.
- Embeds simulated serial numbers and product metadata for testing and validation.

- Supports both 1D and 2D barcode formats.

6. Frontend-Backend Connectivity for Demo

- A simple UI (e.g., Streamlit or HTML-based) shows:
    1. Live OCR results
    2. Validation status (pass/fail)
    3. Label image preview
- Connects with backend to fetch and display logs and product info**.**

7. Tools & Technologies Used

Google Colab:

- Cloud-based platform for running Python notebooks.
- Useful for training/testing ML models and image processing without local setup.

OpenCV + EasyOCR:

- For real-time image preprocessing and text recognition.
- Detects and extracts printed data from labels.

Python:

- Core language for label generation, OCR processing, model integration, and backend logic.
- Libraries include qrcode, opencv-python, easyocr, firebase-admin.

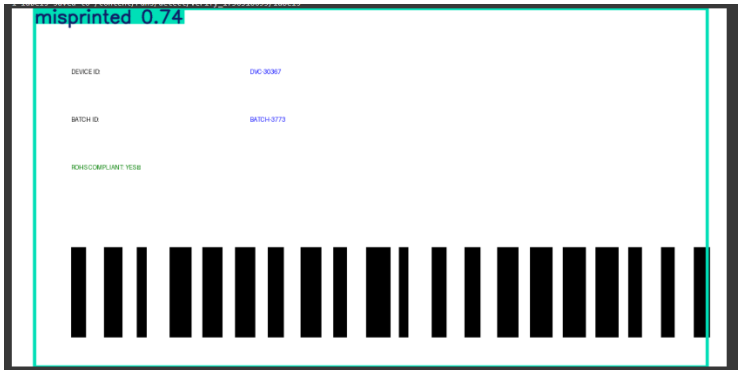
YOLOv5 + LabelImg (Optional):

- For training ML models to detect defective labels or missing components.
- LabelImg used to annotate custom datasets for object detection.

Firebase:

- Backend for storing and validating label metadata.
- Real-time database used to:
    1. Log inspection results
    2. Match scanned data
    3. Provide historical traceability

Barcode Image:



Chapter 5

UI/UX Design & Traceability Database

Overview

This Flutter application is designed to streamline and digitize the process of **product labeling, traceability, and inventory management** through a secure and efficient mobile interface. It features **user authentication**, **password recovery**, a comprehensive **product labeling dashboard**, and integrated **QR/barcode scanning** capabilities to enhance operational speed and accuracy.

The app targets industries like **retail**, **manufacturing**, **logistics**, and **warehousing**, where precise labeling and quick identification of products are crucial. The main objective is to eliminate manual errors, provide real-time tracking of labeled products, and simplify the data entry process using camera-based scanning features.

Built using the **Flutter framework**, the application supports both Android and iOS platforms with a consistent, responsive UI. It follows **Material Design** principles, ensuring a modern look and intuitive user experience. The layout is optimized for mobile and tablet screens, with scalable widgets and responsive layouts for different devices.

The **Login screen** offers a secure gateway using **Firebase Authentication**, where users can enter their credentials. New users can access the **Register screen** to create accounts by filling out a form with name, email, and password. In case of password loss, the **Forgot Password** screen enables recovery through email verification.

The backend is powered by **Firebase Firestore** (or optionally SQLite for local data), supporting real-time storage and retrieval of product data. Data security and access control are enforced through Firebase rules.

2. Objectives

- To develop a mobile application for **efficient product labeling and tracking**.
- Provide a **secure user authentication system** with login, registration, and password recovery.
- Simplify product data entry using **QR code/barcode scanning** through the device's camera.
- Create a **user-friendly dashboard** to manage product information (add, view, edit, delete).
- Ensure **responsive design** compatible with both Android and iOS devices.
- Minimize manual data entry errors by automating label scanning.
- Use **Firebase Authentication** and **Firestore/SQLite** for secure backend integration.

- Design intuitive and clean UI screens using Flutter's **Material Design** components.
- Maintain **data accuracy** by validating all user inputs and showing appropriate error messages.
- Enable **real-time feedback** through SnackBars and Toasts for user actions.
- Focus on **fast navigation** between screens using Navigator and named routes.

3. Target Users

- **Factory workers** who need to label products quickly and accurately.
- **Warehouse staff** responsible for tracking product batches and movements.
- **Retail store employees** managing inventory and shelf labeling.
- **Supervisors or managers** overseeing labeling and traceability processes.
- **Small business owners** who require a simple solution for product labeling.
- **Inventory managers** looking to reduce manual entry errors.
- **Logistics personnel** needing fast access to product information via scanning.
- **Technicians or field staff** managing mobile product verification.
- **Quality control teams** needing real-time label verification.
- **Anyone** involved in product packaging, labeling, or data entry tasks.

4. App Navigation Flow (Flutter Screens)

- Use a diagram or bullet points
- Login → Dashboard
- Register → Login → Dashboard → Login
- Forgot Password → New password → Login
- “Used Navigator.push and PageView for navigation between screens.”

5. Wireframes

Attach low-fidelity sketches or wireframes of main screens:

- Login Screen
- Dashboard
- Add QR or barcode
- Get result

6. UI Design

The UI design of this app follows **Material Design principles**, ensuring a clean, consistent, and responsive user experience across all devices. Each screen is thoughtfully laid out using Flutter's core widgets to provide clarity and ease of use.

Login Screen

- **Widgets Used**: TextFormField, ElevatedButton, GestureDetector
- **Features**: Email/password input, validation, password visibility toggle
- **Design Notes**: Minimalist layout with a brand-colored login button

Register Screen

- **Widgets Used**: Form, TextField, DropdownButton, Checkbox
- **Features**: Account creation with validation, confirm password
- **Design Notes**: Neatly spaced input fields with clear labels

Forgot Password Screen

- **Widgets Used**: TextFormField, ElevatedButton, SnackBar
- **Features**: Enter email for password reset link
- **Design Notes**: Simple layout with concise instructions

Product Labelling Dashboard

- **Widgets Used**: AppBar, Card, ListView, FloatingActionButton, Drawer
- **Features**: View/edit/delete product info, navigate to QR/barcode scanner
- **Design Notes**: Organized card layout, clean typography, intuitive FAB for adding products

QR/Barcode Scanner

- **Widgets Used**: CameraView, qr_code_scanner, IconButton
- **Features**: Live camera scanning, auto-detect codes
- **Design Notes**: Full-screen scanner with a focus box and flashlight toggle

Design Consistency

- **Colors**: Primary #3F51B5, Accent #FFC107, Background #FFFFFF
- **Fonts**: Google Fonts for modern typography (e.g., Roboto, Poppins)

- **Icons**: Material Icons for common actions (edit, delete, scan)

7. Flutter Widgets and Features Used

This Flutter application uses a variety of built-in and third-party widgets to deliver a smooth and functional user experience. Below is a breakdown of key widgets and features implemented across the app:

Authentication Screens (Login, Register, Forgot Password)

- TextFormField – For text input with validation
- Form – To group and manage input fields
- ElevatedButton – For actions like login and submit
- GestureDetector – For clickable text (e.g., “Forgot Password?”)
- SnackBar – To show user feedback
- Visibility – For password show/hide toggle

Dashboard

- Scaffold – Base layout with AppBar, Drawer, and FAB
- AppBar – Displays screen title and navigation
- Drawer – For side navigation (optional)
- Card – To display product entries in a clean format
- ListView.builder – To list dynamic product data
- FloatingActionButton – To add new product entries

Product Input & Display

- DropdownButtonFormField – For selecting predefined product types
- DatePicker – For selecting manufacturing/expiry dates
- AlertDialog – For confirming delete/edit actions
- IconButton – For inline actions like edit or delete

8.Tools & Plugins Used

A) Development Tools

Tool	Purpose
Flutter SDK	Cross-platform UI development

Dart	Programming language for Flutter
Android Studio / VS Code	Code editor and emulator testing
Firebase Console	Authentication and Firestore database
Git & GitHub	Version control and project collaboration

B) Flutter Plugins (Packages)

Package	Purpose
firebase_auth	User authentication (login/register/forgot password)
cloud_firestore	Store and manage product label data
qr_code_scanner or mobile_scanner	Camera-based QR/barcode scanning
provider	State management
google_fonts	Use custom fonts
fluttertoast	Display toast messages for quick feedback

9.Screenshorts

Screenshots provide a visual overview of the app's interface and help demonstrate the design layout, user flow, and key UI elements used across the different screens.

Below are descriptions of each screen, along with suggested screenshot types to include in your report or documentation:

1. Login Screen

- Shows the app's entry point with fields for email and password.
- Highlights: Logo, form fields, login button, and "Forgot Password?" link.

2. Register Screen

- Displays the user registration form with input validation.
- Highlights: Full name, email, password, confirm password, and submit button.

3. Forgot Password Screen

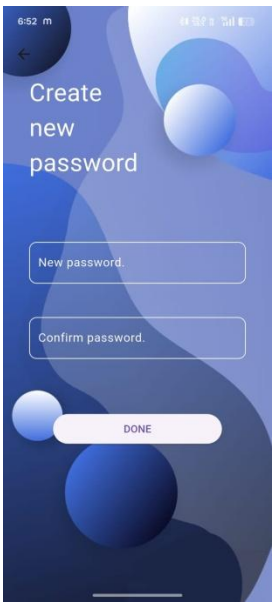
- Allows users to reset their password using their email.
- Highlights: Email field, reset button, confirmation SnackBar or Toast.

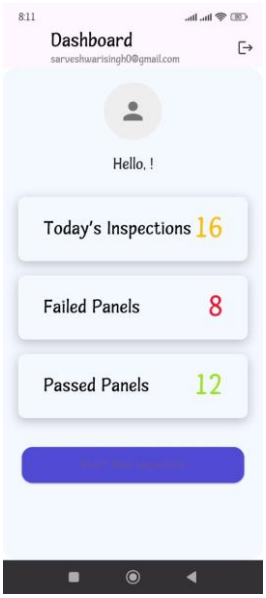
4. Product Labelling Dashboard

- Main screen for managing product entries.
- Highlights: Product list in cards or table format, AppBar, FloatingActionButton, and options for editing or deleting.


6:52
Welcome !
Email
Password
Sign in
Forgot password?
New user


6:52
Create
Account
Name
Email
Password
Sign Up
Sign In


6:52
Create
new
password
New password.
Confirm password.
DONE


8:11
Dashboard
Hello, !
Today's Inspections 16
Failed Panels 8
Passed Panels 12