A Project Report on

Shelf Track: Intelligent Empty Shelf and Low-Stock Monitoring System

***submitted in the partial fulfillment of the requirements for the award of the degree of***

**BACHELOR OF TECHNOLOGY**

**in**

**ARTIFICIAL INTELLIGENCE & MACHINE LEARNING**

**By**

**UDYAVARA UDAY SHANKAR 212G1A3953 G NIJAMUDDIN 212G1A3912**

**KALINGA GURUKIRAN 212G1A3929**

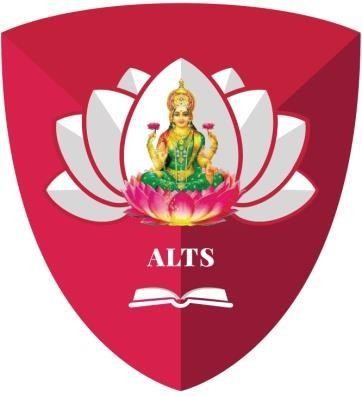
**BOLLINENI YASWANTH 212G1A3908**

**M KAVITHA 212G1A3935**

### Under Supervision of

**Dr. K Bhargavi M.Tech, Ph.D**

**Associate Professor & HOD Department of Computer science and Engineering**

****

**DEPARTMENT OF ARTIFICIAL INTELLIGENCE & MACHINE LEARNING**

**ANANTHA LAKSHMI INSTITUTE OF TECHNOLOGY AND SCIENCES**

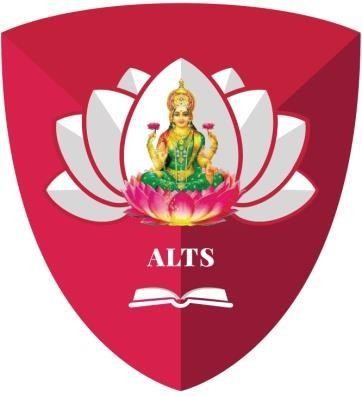
**Approved by AICTE, New Delhi & Affiliated to J.N.T.U. Ananthapuram, Accredited by NAAC Near S.K. University, Itikalapally(V), Anantapur (Dt) – 515 721.A.P.**

**(2021-2025)**

**ANANTHA LAKSHMI INSTITUTE OF TECHNOLOGY AND SCIENCES**

**Approved by AICTE, New Delhi & Affiliated to J.N.T.U. Ananthapuram, Accredited by NAAC Near S.K. University, Itikalapally(v), Anantapur (v) – 515 721.A.P.**

**DEPARTMENT OF ARTIFICIAL INTELLIGENCE & MACHINE LEARNING**

****

**CERTIFICATE**

This is to certify that the **“Shelf Track: Intelligent Empty Shelf and Low-Stock Monitoring System”** is bonafied work carried out by following students of this institute under guidance of **Dr. K Bhargavi M.Tech, Ph.D.** for the partial fulfilment of the requirements for the award of the degree of Bachelor of Technology in **Artificial Intelligence & Machine Learning** from ANANTHA LAKSHMI INSTITUTE OF TECHNOLOGY AND SCIENCES, Itikalapalli (V), Anantapur in the academic year of 2024- 2025**.**

**UDYAVARA UDAY SHANKAR 212G1A3953 G NIJAMUDDIN 212G1A3912**

**KALINGA GURUKIRAN 212G1A3929**

**BOLLINENI YASWANTH 212G1A3908**

**M KAVITHA 212G1A3935**

|  |  |
| --- | --- |
| **Name of the Supervisor**  **Dr. K Bhargavi M.Tech, Ph.D**  Associate Professor & HOD, Department of Computer science and Engineering,  Anantha Lakshmi Institute of Technology  and Sciences, Anantapur. | **Head of the Department**  **Dr. K Bhargavi M.Tech, Ph.D**  Associate Professor & HOD, Department of Computer science and Engineering,  Anantha Lakshmi Institute of Technology  and Sciences, Anantapur. |

**Viva Voce Conducted on :**

**Internal Examiner External Examiner**

# DECLARATION

We here by declare that the work which is being presented in this dissertation entitled **“Shelf Track: Intelligent Empty Shelf and Low-Stock Monitoring System”** submitted towards the partial fulfilment of the requirements for the award of the degree of **BACHELOR OF TECHNOLOGY** in the **ARTIFICIAL INTELLIGENCE & MACHINE LEARNING, ANANTHA LAKSHMI INSTITUTE OF TECHNOLOGY AND SCIENCES** is an authentic work carried out by us during 2024-2025 under the supervision of **Dr. K Bhargavi M.Tech, Ph.D. Associate Professor & HOD, Department of Computer Science and Engineering**, Anantha Lakshmi Institute of Technology and Sciences, Itikalapalli(V), Anantapur.

The matter embodied in this dissertation report has not been submitted by us for the award of any other degree or diploma. Further, the technical details furnished in the various chapters in this thesis are purely relevant to the above project.

**UDYAVARA UDAY SHANKAR 212G1A3953 G NIJAMUDDIN 212G1A3912**

**KALINGA GURUKIRAN 212G1A3929**

**BOLLINENI YASWANTH 212G1A3908**

**M KAVITHA 212G1A3935**

# ACKNOWLEDGEMENTS

The satisfaction and euphoria that accompany the successful completion of any task would be incomplete without the mention of people who made it possible, whose constant guidance and encouragement crowned our efforts with success. It is a pleasant aspect that we have now the opportunity to express our gratitude for all of them.

It is with immense pleasure that we would like to express my indebted gratitude to

Dr. K Bhargavi M.Tech, Ph.D., Associate Professor & HOD, **DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**, who has guided us a lot and encouraged us in every step of Internship. We thank him/her for the stimulating guidance, constant encouragement and constructive criticism which have made possible to bring out this project work.

It is with immense pleasure that we would like to express my indebted gratitude to **Dr. K Bhargavi M.Tech, Ph.D., Associate Professor & HOD**, **DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**, for giving me an opportunity to work on a project that was so challenging and interesting for me. we remember with great emotion. the constant encouragement and help extended to me by her that went even beyond the realm of academics.

We wish to convey our special thanks to **Dr. B.M.G.PRASAD M.Tech,Ph.D , DEAN OF CSE, ANANTHA LAKSHMI INSTITUTE OF TECHNOLOGY AND SCIENCES** for

giving the required information in doing my project work.

I wish to convey my special thanks to **Dr. Ramamurthy MTech., Ph.D., Principal** of **Anantha Lakshmi Institute of Technology and Sciences** for giving the required information in doing my project work. Not to forget, we thank all other faculty and non-teaching staff, and my friends who had directly or indirectly helped and supported us in completing my project in time.

We also express our sincere thanks to **Sri M. Anantha Ramudu, Chairman** and **Sri M. Ramesh Naidu, Vice Chairman** of **Anantha Lakshmi Institute of Technology and Sciences** for providing excellent facilities**.**

**UDYAVARA UDAY SHANKAR 212G1A3953 G NIJAMUDDIN 212G1A3912**

**KALINGA GURUKIRAN 212G1A3929**

**BOLLINENI YASWANTH 212G1A3908**

**M KAVITHA 212G1A3935**

**TABLE OF CONTENTS**

**Contents Page.no**

**CHAPTER 1: INTRODUCTION** 1

* 1. Faster R-CNN for Shelf Monitoring 2
  2. Machine Learning (ML) for Inventory Insights 2

**1.2.1 Convolutional Neural Networks (CNNs) 2**

1.2.2 Optical Character Recognition (OCR) 2

1.2.3 Data Analytics for Restocking and Reporting 3

* 1. Privacy-Centric Design 3

**CHAPTER 2: LITERATURE SURVEY**  4

2.1 Overview of Shelf Monitoring in Retail 4  
2.2 Real-Time Inventory Management 4

2.3 Machine Learning for Shelf Detection 5

2.4 Integrated Low-Stock Detection and Predictive Analytics 5

2.5 Limitations of Existing Solutions 5

**CHAPTER 3: SYSTEM ANALYSIS** 6

3.1 System Study 6

3.1.1 Economical Feasibility 6

3.1.2 Technical Feasibility 6

3.1.3 Social Feasibility 6

3.2 Existing Systems 7

3.3 Disadvantages of Existing Systems 7

3.4 Proposed System 7

3.5 System Requirements 8

**CHAPTER 4: PROPOSED SYSTEM** 10

4.1 Proposed System 10

4.2 Advantages of the Proposed System 10

4.3 System Architecture 11

4.3.1 Overview 11

4.3.2 Workflow Explanation 11

4.4 Objectives 12

4.4.1 Fulfilling the Objectives 13

4.4.2 OCR Process Flow 13

4.4.3 Integrated Algorithms 14

4.5 Factors Influencing Shelf Availability 16

4.5.1 Store Environment Parameters 17

4.5.2 Camera Setup Parameters 17

4.5.3 Model Performance Metrics 17

**CHAPTER 5: SYSTEM DESIGN** 18

5.1 Data Flow Diagram 18

5.2 UML diagrams 19

5.3 Use case diagram 20

5.4 Class diagram 22

5.5 Sequence diagram 23

5.6 Activity diagram 24

**CHAPTER 6: IMPLEMENTATION** 27

6.1 Modules 27

6.2 Modules Description 27

6.2.1 Home Module 27

6.2.2 Store Manager Module 28

6.2.3 Owner Module 28

6.2.4 Inventory Management Module 29

6.2.5 Notification Module 29

6.2.6 Reporting Module 30

6.3 Python 31

6.4 Django 42

**CHAPTER 7: TESTING AND EVALUATION** 51

7.1 System test 51

7.2 Types of tests 51

7.2.1 Unit testing 51

7.2.2 Integration testing 51

7.2.3 Functional test 51

7.2.4 System Test 52

7.2.5 White Box Testing 52

7.2.6 Black Box Testing 52

7.2.7 Integration Testing 53

**CHAPTER 8: RESULTS** 55

**CHAPTER 9: CONCLUSION & FUTURE ENHANCEMENT** 61

**CHAPTER 10: REFERENCES** 62

**CHAPTER 11: PAPER PUBLICATION** 64

**LIST OF FIGURES**

|  |  |  |
| --- | --- | --- |
| **Fig. No** | **Name of Figure** | **Page. No** |
| 4.1 | System architecture | 11 |
| 4.2 | OCR Process Flow | 14 |
| 5.1 | Data flow diagram for Shelf Track | 18 |
| 5.2 | Roles of actors in the system | 21 |
| 5.3 | Class Diagram for the relationship among the classes | 22 |
| 5.4 | Sequence of the store management | 23 |
| 5.5 | Work Flow of Activities in the Store management | 25 |
| 6.1 | Interaction between Web Browser and Database using Django | 43 |
| 6.2 | Communication between User and Client | 43 |
| 8.1 | Home Page | 55 |
| 8.2 | Owner login page | 55 |
| 8.3 | Store manager login | 56 |
| 8.4 | Owner home page | 56 |
| 8.5 | Employee Enroll form | 57 |
| 8.6 | Employee view form | 57 |
| 8.7 | Employee update form | 58 |
| 8.8 | Store manager home page | 58 |
| 8.9 | Image upload form | 59 |
| 8.10 | Inventory list | 59 |
| 8.11 | Low stock notification | 60 |
| 8.12 | Empty shelf detection | 60 |

**LIST OF ALGORITHMS**

|  |  |  |
| --- | --- | --- |
| **Alg. No** | **Name of Algorithm** | **Page. No** |
| 4.1 | Empty Shelf Detection Using Faster R-CNN | 14 |

4.2 Shelf Inventory Recognition Algorithm 15

4.3 Automated Inventory Management System 16

**LIST OF TABLES**

|  |  |  |
| --- | --- | --- |
| **Fig. No** | **Name of Table** | **Page. No** |
| 7.1 | Test Cases | 54 |

**LIST OF ABBREVATIONS**

|  |  |  |
| --- | --- | --- |
| **S.No** | **Name** | **Abbreviation** |
| 1 | R-CNN | Region-based Convolutional Neural Network |
| 2 | RPNs | Region Proposal Networks |
| 3 | CNN | Convolutional Neural Networks |
| 4 | OCR | Optical Character Recognition |
| 5 | RFID | Radio Frequency Identification |
| 6 | CCTV | Closed Circuit Television |
| 7 | HTML | Hypertext Markup Language |
| 8 | CSS | Cascading Style Sheets |
| 9 | DFD | Data Flow Diagram |
| 10 | UML | Unified Modeling Language |
| 11 | SMTP | Simple Mail Transfer Protocol |
| 12 | SMS | Short message service |
| 14 | DB | Data Base |

Shelf Track: Intelligent Empty Shelf and Low-Stock Monitoring System

# ABSTRACT

Efficient inventory management is crucial for success in the fast-paced, marketing-driven world. Traditional remote inventory management relies on image processing to identify missing objects on shelves, often raising concerns about customer privacy. Unlike previous approaches that focus solely on detecting specific missing items, this work emphasizes identifying empty shelf regions without compromising customer identity. Additionally, it locates the exact position of the empty shelves in a retail store. Expanding on this, the system incorporates a feature to detect low inventory levels and identify specific items that require replenishment.

This is achieved by integrating advanced technologies such as Optical Character Recognition (OCR) for item labeling, convolutional neural networks (CNNs) for stock-level detection, and a database-driven stock management system for real-time inventory tracking and analysis. The system combines the pre-trained Faster R-CNN model for empty shelf detection with a hybrid OCR-CNN model for recognizing item labels and quantities.

In addition, the system features a monthly report generation module that aggregates data on shelf occupancy, low-stock incidents, and replenishment activities. These comprehensive reports provide actionable insights, trend analyses, and performance metrics, which can be easily shared with management to support strategic decision-making. This innovative solution offers a privacy-centric, comprehensive method for shelf monitoring, low-stock detection, inventory replenishment, and performance reporting, addressing critical needs in the retail sector.

**Keywords**—inventory management, Faster R-CNN, OCR, CNN, empty shelf detection, stock monitoring, inventory replenishment, retail automation, monthly reporting.

# CHAPTER-01

# INTRODUCTION

The rapidly evolving retail industry faces mounting challenges in managing stock and inventory efficiently. Traditional methods of inventory monitoring, primarily reliant on manual efforts, are labor-intensive, prone to delays, and susceptible to inaccuracies. Retail stores often grapple with high labor costs and operational inefficiencies, as shelf monitoring is typically performed during off-peak hours. This delay can result in missed sales opportunities when items are unavailable on shelves, even though additional stock may exist in the warehouse.

The absence of a product on the shelf often leads to undesirable consumer behavior, such as switching to competing brands, purchasing from alternate stores, or abandoning the purchase altogether. Retailers lose not only immediate revenue but also long-term customer loyalty. In this context, automated inventory monitoring and restocking have emerged as critical needs, providing insights into stock levels, customer preferences, staff performance, and overall sales efficiency.

Traditional inventory monitoring techniques rely on image processing to detect missing items or empty shelves. While effective, these methods demand real-time image capture—often via CCTV systems—which raises concerns about cost, feasibility, and customer privacy. Machine learning (ML) and advanced data analytics offer a transformative solution, enabling automated detection of empty shelves and low stock levels while safeguarding customer privacy. The proposed system leverages the pre-trained Faster R-CNN model for detecting empty shelves and incorporates advanced features for identifying low inventory, recommending restocking actions, and generating comprehensive monthly reports.

This dynamic approach promises to revolutionize inventory management, reducing labor costs, enhancing operational efficiency, and driving customer satisfaction. The system is versatile, catering not only to retail stores but also to diverse industries requiring efficient stock management solutions.

### ****Faster R-CNN for Shelf Monitoring****

### Faster R-CNN (Region-based Convolutional Neural Network) is a state-of-the-art object detection algorithm that combines high accuracy with efficient computation. It is particularly suited for binary classification tasks, such as detecting empty shelves in a retail environment. By integrating convolutional neural networks (CNNs) with region proposal networks (RPNs), Faster R-CNN achieves precise object localization and classification.

### In the proposed system, Faster R-CNN is employed to identify empty regions on retail shelves. This model uses ResNet-50 as its backbone architecture, enabling robust feature extraction while maintaining computational efficiency. The outputs from the Faster R-CNN model include the location of empty shelves, facilitating targeted restocking and optimizing store operations.

### ****Machine Learning (ML) for Inventory Insights****

### Machine learning plays a pivotal role in enhancing the functionality of the proposed system. Beyond detecting empty shelves, ML algorithms analyze stock levels, predict low inventory, and generate actionable recommendations for restocking.

#### **Convolutional Neural Networks (CNNs):**

#### CNNs are employed to analyze product images and determine stock quantities on shelves. By comparing detected stock levels with predefined thresholds, the system identifies low inventory items. This approach eliminates the need for real-time human intervention, providing a scalable and automated solution for stock monitoring.

#### **Optical Character Recognition (OCR):**

#### OCR technology is integrated to read product labels, barcodes, and other textual information on shelves. This functionality allows the system to accurately identify specific products with low stock levels, ensuring precise and targeted restocking.

#### **Data Analytics for Restocking and Reporting:**

#### The system utilizes advanced data analytics to track stock trends over time, offer insights into product popularity and sales performance, and generate monthly reports. These comprehensive reports aggregate data on shelf occupancy, low-stock incidents, and restocking actions. They provide actionable insights, trend analyses, and performance metrics, enabling retailers to evaluate the effectiveness of inventory management strategies and support strategic decision-making.

### ****Privacy-Centric Design****

### Customer privacy is a critical consideration in retail environments. The proposed system is designed to operate without capturing or storing identifiable customer information, ensuring compliance with

### data protection regulations. By focusing solely on shelf monitoring, inventory management, and performance reporting—including monthly analytics—the system addresses privacy concerns while delivering actionable insights.

# CHAPTER-02

# LITERATURE SURVEY

## Overview of Shelf Monitoring in Retail

### Michael, K.; McCathie, L. The Pros and Cons of RFID in Supply Chain Management. In Proceedings of the 4th Annual International Conference on Mobile Business, ICMB, Sydney, NSW, Australia, 11–13 July 2015; pp. 623–629

**Moorthy, R.; Behera, S.; Verma**

Traditional retail inventory management has long relied on manual shelf inspections and periodic image analysis through CCTV. Although such methods have served basic monitoring needs, they are labor-intensive and often result in delayed restocking, leading to missed sales opportunities. Recent advancements have introduced automated shelf monitoring using machine learning models, which enhance on-shelf availability by accurately detecting empty spaces.

## Real-Time Inventory Management

### Michael, K.; McCathie, L. The Pros and Cons of RFID in Supply Chain Management. In Proceedings of the 4th Annual International Conference on Mobile Business, ICMB, Sydney, NSW, Australia, 11–13 July 2015; pp. 623–629

**Moorthy, R.; Behera, S.; Verma**

Inventory management by providing real-time data on shelf stock levels through sensors such as RFID and weight sensors. These systems enable continuous monitoring and timely alerts for replenishment needs. Its traditional implementation often requires extensive infrastructure and does not inherently integrate advanced image processing techniques necessary for dynamic decision- making in retail.

**2.3 Machine Learning for Shelf Detection**

### Michael, K.; McCathie, L. The Pros and Cons of RFID in Supply Chain Management. In Proceedings of the 4th Annual International Conference on Mobile Business, ICMB, Sydney, NSW, Australia, 11–13 July 2015; pp. 623–629

**Moorthy, R.; Behera, S.; Verma**

Recent developments in machine learning, particularly the use of convolutional neural networks (CNNs) and region-based detectors like Faster R-CNN, have shown considerable promise in automating the detection of empty shelves. By leveraging pre-trained models and deep learning architectures, these systems can precisely localize empty shelf regions, enabling prompt and effective restocking decisions without relying on continuous human supervision.

## Integrated Low-Stock Detection and Predictive Analytics

### Michael, K.; McCathie, L. The Pros and Cons of RFID in Supply Chain Management. In Proceedings of the 4th Annual International Conference on Mobile Business, ICMB, Sydney, NSW, Australia, 11–13 July 2015; pp. 623–629

**Moorthy, R.; Behera, S.; Verma**

Modern inventory systems are evolving to not only detect empty shelves but also to assess low- stock conditions. By integrating Optical Character Recognition (OCR) with CNN-based models, the system is capable of reading product labels and estimating remaining stock levels. When combined with predictive analytics that leverage historical and real-time data, the approach generates actionable restocking recommendations and optimizes inventory management.

## Limitations of Existing Solutions

### Michael, K.; McCathie, L. The Pros and Cons of RFID in Supply Chain Management. In Proceedings of the 4th Annual International Conference on Mobile Business, ICMB, Sydney, NSW, Australia, 11–13 July 2015; pp. 623–629

**Moorthy, R.; Behera, S.; Verma**

Many existing shelf monitoring solutions depend exclusively on either manual inspections or isolated technologies such as basic image processing or RFID. Such approaches often lack a comprehensive, real-time view of inventory levels and can raise privacy concerns associated with continuous video monitoring. Moreover, without an integrated predictive component, these systems are unable to effectively forecast restocking needs, resulting in suboptimal inventory management.

# CHAPTER-03

# SYSTEM ANALYSIS

## System Study

### Feasibility Study

The feasibility of the project is evaluated in this phase, and a business proposal is put forth with a general plan and cost estimates. The proposed system is examined to ensure that it is not a burden to the company. For this study, an understanding of the major system requirements is essential. Three key considerations are involved in the feasibility analysis:

* **Economical Feasibility**
* **Technical Feasibility**
* **Social Feasibility**

### 3.1.1 Economical Feasibility

This study assesses the financial impact that the system will have on the organization. The proposed shelf-monitoring solution leverages freely available open-source frameworks and pre- trained models, significantly reducing development costs compared to traditional RFID tagging or manual inspections. The overall expenditure is kept minimal by focusing on essential customized components only.

### 3.1.2 Technical Feasibility

The technical feasibility evaluates whether the system’s requirements are manageable within the existing technical resources. The project employs a pre-trained Faster R-CNN model, integrated with CNN and ResNet-50 architectures, to detect empty shelves. Additionally, it incorporates OCR and further CNN modules for recognizing product labels and assessing low-stock conditions. This integration, ensures that the system can be implemented without imposing high demands on current infrastructure.

### 3.1.3 Social Feasibility

Social feasibility examines the system’s acceptance among its users and stakeholders. The proposed solution is designed to operate in public retail environments while preserving customer privacy by focusing solely on shelf and inventory data. Minimal training is required for staff operate the system, and its user-friendly interface, including actionable insights and automated restocking alerts, ensures a smooth transition from traditional methods.

## Existing Systems

Traditional inventory management methods rely on manual inspections, RFID tagging, and basic image processing techniques to monitor shelf availability in retail stores. In these systems, images of the shelves are captured—either manually or through CCTV—and processed to detect missing products. While these methods have been in use for years, they often suffer from high labor costs, delayed updates, and privacy concerns.

## Disadvantages of Existing Systems

**Labor Costs and Inefficiency:** Manual inspections are labor-intensive and often conducted during off-peak hours, leading to delays and increased operational costs.

**Sales Losses Due to Delays:** A delay in inventory monitoring can cause significant sales losses when products run out on shelves, despite additional stock being available in storage.

**Privacy Concerns:** Continuous image capture for inventory monitoring may inadvertently record customer images, raising privacy issues.

**Limited Scalability:** Technologies like RFID tagging and basic image processing require substantial investment and are often not scalable for larger or more dynamic retail environments.

**Inconsistent Performance:** Traditional image processing methods may fail in complex retail environments with varying lighting and clutter, resulting in inaccurate detection.

## 3.4 Proposed System

The proposed system introduces a comprehensive solution for retail shelf monitoring by combining advanced computer vision and machine learning techniques. Key components of the system include:

**Empty Shelf Detection:** A pre-trained Faster R-CNN model, integrated with a ResNet-50 backbone, is used to accurately detect empty shelf regions. This method minimizes the need for continuous real-time data capture, reducing both processing time and privacy concerns.

**Low-Stock Detection:** In addition to detecting empty shelves, the system employs Optical Character Recognition (OCR) and additional CNN modules to read product labels and assess current stock levels. By comparing these readings with predefined thresholds, the system can identify which items are low in stock and require replenishment.

**Predictive Analytics:** Historical data and real-time insights are combined using predictive analytics to forecast inventory trends, allowing for proactive restocking and improved decision- making.

Together, these technologies ensure a scalable, cost-effective, and privacy-centric solution that enhances overall retail efficiency.

**3.5 System Requirements:**

**3.5.1 Hardware Requirements:**

* System : Intel Core i7.
* Hard Disk : 1TB.
* Monitor : 14-inch Colour Monitor
* Input Devices : Optical Mouse
* Ram : 16GB.

**3.5.2 Software Requirements:**

* Operating system : Windows 10.
* Coding Language : Python
* Front-End : HTML, CSS
* Designing : HTML, CSS, JavaScript

**Database:** SQLite (or other lightweight database for storing inventory data)

**Machine Learning Framework:** TensorFlow (with TensorFlow Lite for deployment)

**OCR Framework:** Tesseract (or similar OCR engine)

By addressing these requirements, the proposed shelf-monitoring system aims to offer an efficient, scalable, and privacy-conscious solution that enhances retail inventory management through real- time detection, low-stock ident-ification, and predictive restocking insights.

# ****CHAPTER-04****

# ****PROPOSED SYSTEM****

### ****4.1 Proposed System****

The proposed system utilizes a **pre-trained Faster R-CNN model**, integrating **CNN and ResNet-50 architecture**, to detect **empty shelves** in retail stores. Unlike traditional methods that require continuous real-time monitoring, this approach effectively identifies shelf shortages while ensuring **customer privacy** is not compromised.

By leveraging **deep learning techniques**, the system processes images from store shelves to recognize empty spaces and alert store managers. This enhances inventory management efficiency, reduces manual labor, and optimizes restocking strategies.

### ****4.2 Advantages of the Proposed System****

* **High Accuracy**: Achieves **99% accuracy**, outperforming models like **YOLO and standard CNNs**.
* **Cost-Effective**: Eliminates the need for **manual shelf inspections** and **expensive RFID tagging**.
* **Enhanced Privacy**: Focuses **solely on empty shelves**, ensuring that **customer identities remain untracked**.
* **Real-Time Insights**: Helps analyze **inventory status, staff efficiency, and product demand** for better decision-making.
* **Scalable Solution**: Adaptable to various **store sizes** and retail environments.
* **Efficient Processing**:
  + **Training Time**: ~6 hours
  + **Prediction Speed**: As fast as **24 seconds per analysis**
* **Optimized Workflow**: Utilizes **Region Proposal Networks (RPNs)** for faster and more accurate shelf detection.

**4.3 System Architecture**

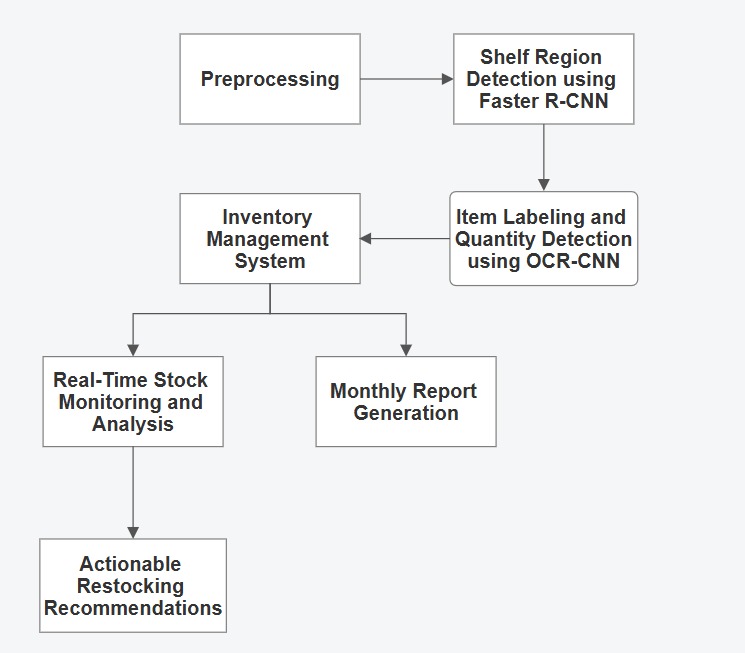
****

Fig-4.1: System Architecture**.**

**4.3.1 Overview**

The system is designed to detect empty shelves in retail stores using image processing and deep learning techniques. It follows a structured approach that includes data collection, preprocessing, shelf detection, inventory management, and restocking recommendations. Refer to Fig. 4.1 for an overview of the system's architecture

### ****4.3.2 Workflow Explanation****

The system design consists of the following steps:

#### **1. Image Data Collection**

* Cameras capture images of store shelves in real-time.
* The images include both fully stocked and partially empty shelves for analysis.

#### **2. Preprocessing**

* The collected images undergo cleaning, resizing, and enhancement.
* Contrast adjustment and noise reduction are applied to improve clarity.

#### **3. Shelf Region Detection using Faster R-CNN**

* A pre-trained Faster R-CNN model detects shelf regions in the processed images.
* Region Proposal Networks (RPNs) are used to identify potential empty spaces. See Fig. 4.1 for the flow of data through the Faster R-CNN module.

#### **4. Item Labeling and Quantity Detection using OCR-CNN**

* Optical Character Recognition (OCR) and CNN-based models label detected items.
* The system estimates the quantity of products on the shelves.

#### **5. Inventory Management System**

* The detected shelf data is processed to monitor stock availability.
* The system tracks product counts and identifies restocking needs.

#### **6. Real-Time Stock Monitoring and Analysis**

* The system continuously monitors stock levels and updates inventory data.
* Empty shelf detections trigger alerts for store managers.

#### **7. Monthly Report Generation**

* The system generates reports summarizing stock availability and sales trends.
* Reports help in analyzing demand fluctuations and optimizing inventory.

#### **8. Actionable Restocking Recommendations**

* Based on detected shortages, the system suggests restocking actions.
* Recommendations help store managers optimize shelf replenishment strategies.

**4.4 Objectives**

The proposed system is designed with two primary objectives.

**Objective 1: Accurate Empty Shelf Detection ensures** that empty or near-empty shelf spaces are accurately identified in real time, without compromising customer privacy.

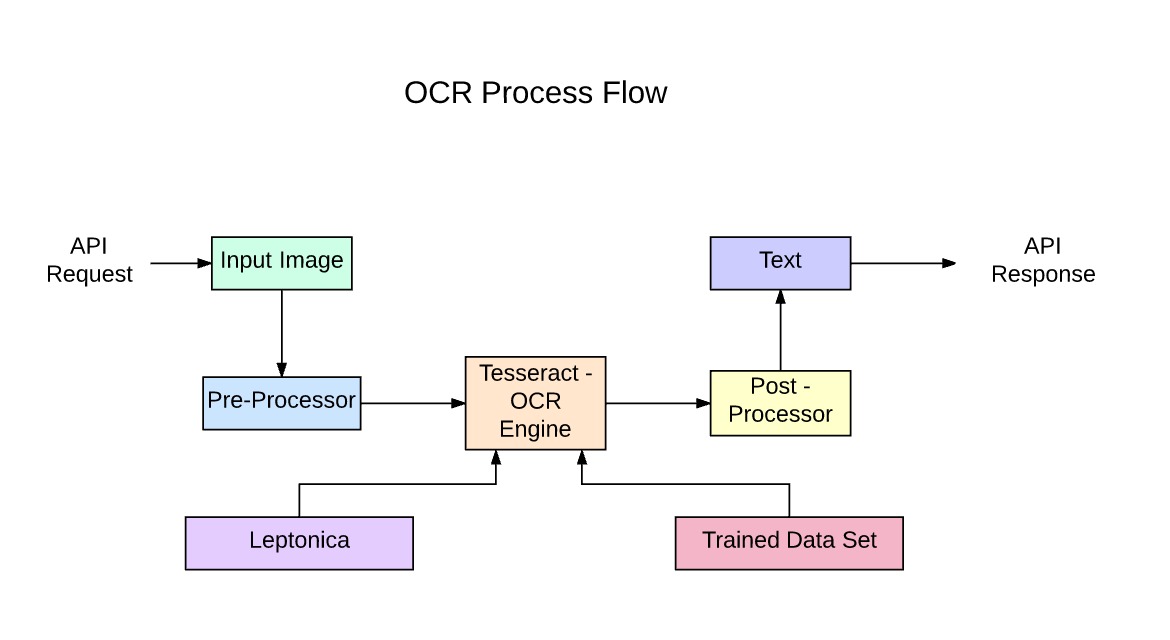
**Objective 2: Automated Inventory Monitoring and Restocking** provides a comprehensive inventory management solution that tracks stock levels, identifies low-stock items, and recommends restocking actions promptly.

**4.4.1 Fulfilling the Objectives**

To achieve Objective 1, the system leverages the Faster R-CNN model with ResNet-50 as the backbone to maintain high accuracy (up to 99%) in detecting empty shelf regions. The images undergo cleaning, resizing, and contrast enhancement to improve detection rates under various lighting conditions and shelf arrangements. By focusing solely on shelf regions, the model preserves customer anonymity and avoids privacy concerns. For Objective 2, an OCR-CNN module identifies product labels and estimates quantities. These data points are stored in a cloud-based inventory management system, enabling real-time stock monitoring, monthly reporting, and automated alerts when stock levels fall below defined thresholds. This approach allows store managers to make informed decisions and receive timely replenishment suggestions. Refer to Fig. 4.1 for how these objectives are achieved within the system architecture.

**4.4.2 OCR Process Flow**

Following shelf region detection, the system uses OCR-based item labeling and quantity detection to determine which specific products need replenishment. Figure 4.2 shows the OCR process flow, in which the detected shelf region is first passed through a pre-processor to enhance image quality for more accurate text extraction. The Tesseract OCR engine then retrieves textual information which is subsequently cleaned, validated, and fed into the inventory system as recognized labels.

Fig-4.2: OCR Process Flow

**4.4.3 Integrated Algorithms**

The following paragraphs illustrate how the system detects empty shelves, labels products, and updates inventory in a cohesive manner.

For empty shelf detection, the input shelf image is first resized (for example, to 512×512) and enhanced to reduce noise. It is then fed into the Faster R-CNN model, which uses a Region Proposal Network (RPN) to identify potential empty regions before classifying each region as empty or not empty. If any empty regions are detected, an alert is generated and the coordinates of these regions are recorded for further processing. Refer to Algorithm 4.1 for pseudo-code implementation.

|  |
| --- |
| **Algorithm 4.1: Empty Shelf Detection Using Faster R-CNN** |
| **Input:** ShelfImage  **Output:** DetectedEmptyRegions  1Preprocess(ShelfImage):  Resize image to a standard resolution (e.g., 512x512).  Enhance contrast and reduce noise.  Return ProcessedImage.  2. Regions ← FasterRCNN(ProcessedImage):  Use Region Proposal Network (RPN) to propose potential empty regions.  Classify each region as empty or not empty.  Return Regions containing empty shelf areas.  3. If Regions is not empty:  Alert = "Empty Shelf Detected"  Store empty region coordinates for further analysis  Else:  Alert = "No Empty Shelf" |

Once the system identifies empty shelf regions, it proceeds to label items and estimate stock levels. For each detected region, a RegionOfInterest (ROI) is extracted and passed through OCR. The recognized text is mapped to product labels, and a CNN-based model estimates the quantity of each item. Refer to Algorithm 4.2 for implementation details.

|  |
| --- |
| **Algorithm 4.2: Shelf Inventory Recognition Algorithm** |
| **Input**: ShelfImage, Regions  **Output**: LabeledItems, StockLevels  1. For each region in Regions:  Extract RegionOfInterest (ROI) from ShelfImage.  TextData = OCR(ROI):  i. Preprocess ROI for OCR.  ii. Apply Tesseract or OCR-CNN for text extraction.  iii. Post-process extracted text.  ItemLabel = IdentifyProduct(TextData)  Quantity = EstimateQuantity(ROI, CNNModel)  LabeledItems.append(ItemLabel)  StockLevels[ItemLabel] = Quantity  2. Return LabeledItems, StockLevels |

Finally, the inventory management system updates the stock levels and recommends restocking actions. For each labeled item, the system retrieves the current stock from the database, adds the detected quantity, and updates the database accordingly. If any item’s stock falls below its threshold, the system automatically adds it to a restock list, which is then sent to the store manager.  
Refer to Algorithm 4.3 for implementation details.

|  |
| --- |
| **Algorithm 4.3: Automated Inventory Management System** |
| **Input:** LabeledItems, StockLevels  **Output:** UpdatedInventory, RestockRecommendations  1. For each item in LabeledItems:  current\_stock = Database.getCurrentStock(item)  new\_stock = current\_stock + StockLevels[item]  Database.updateStock(item, new\_stock)  2. For each item in Database:  if item.stock < item.threshold:  RestockRecommendations.append(item)  3. Send RestockRecommendations to StoreManager  4. Return UpdatedInventory, RestockRecommendations |

### Overall, this proposed system integrates empty shelf detection, OCR-based product labeling, and real-time inventory management to deliver a robust, privacy-centric solution for retail stores. By combining Faster R-CNN for detecting empty regions and OCR-CNN for product identification, it ensures timely restocking, minimizes stockouts, and reduces manual labor. The architecture, objectives, and integrated algorithms described here demonstrate a comprehensive approach to real-time shelf monitoring and efficient inventory control.

### ****4.5 Factors Influencing Shelf Availability****

Shelf detection accuracy depends on various **store-specific** and **technical factors**:

#### **4.5.1 Store Environment Parameters**

* **Lighting Conditions** – Poor lighting can affect image clarity.
* **Shelf Arrangement** – Complex layouts may require additional training data.
* **Product Variety** – Diverse packaging colors and shapes can impact detection accuracy.

#### **4.5.2 Camera Setup Parameters**

* **Camera Resolution** – Higher resolution improves detection performance.
* **Viewing Angle** – A well-placed camera ensures better shelf coverage.
* **Frame Rate** – Determines how frequently the shelves are scanned.

#### **4.5.3 Model Performance Metrics**

* **False Positives** – Incorrectly marking a full shelf as empty.
* **False Negatives** – Failing to detect an actual empty shelf.
* **Inference Speed** – The time taken to analyze a shelf image and generate results.

**CHAPTER-05**

**SYSTEM DESIGN**

**5.1 Data Flow Diagram:**

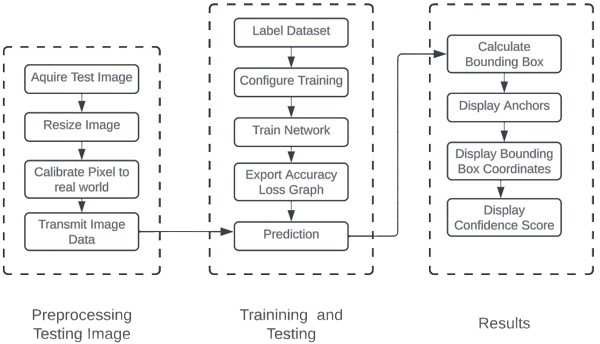
1. The DFD is also called as bubble chart. It is a simple graphical formalism that can be used to represent a system in terms of input data to the system, various processing carried out on this data, and the output data is generated by this system.
2. The data flow diagram (DFD) is one of the most important modeling tools. It is used to model the system components. These components are the system process, the data used by the process, an external entity that interacts with the system and the information flows in the system.
3. DFD shows how the information moves through the system and how it is modified by a series of transformations. It is a graphical technique that depicts information flow and the transformations that are applied as data moves from input to output.
4. DFD is also known as bubble chart. A DFD may be used to represent a system at any level of abstraction. DFD may be partitioned into levels that represent increasing information flow and functional detail.

Fig-5.1: Data flow diagram for Shelf Track

**5.1.1 Overview**

The system is designed for image-based object detection, following a structured approach that includes image preprocessing, model training, prediction, and result visualization.

**Workflow Explanation**

The system design consists of the following steps:

1. **Preprocessing Testing Image**
   * The system begins by acquiring a test image for analysis.
   * The image is resized to ensure uniformity in input dimensions.
   * Pixel calibration is performed to map image data to real-world values.
   * The processed image data is then transmitted for further computation.
2. **Training and Testing**
   * The dataset is labeled to provide ground truth for model learning.
   * Training configurations, such as hyperparameters and batch sizes, are set.
   * The neural network is trained using the labeled dataset.
   * Accuracy and loss graphs are generated to monitor training progress.
   * The trained model is used to make predictions on test images.
3. **Results**
   * The system calculates bounding boxes to identify detected objects.
   * Anchor points are displayed to assist in object localization.
   * The coordinates of detected bounding boxes are extracted and shown.
   * A confidence score is displayed to indicate the accuracy of the detection.

This workflow ensures an efficient and structured approach to image-based object detection, optimizing accuracy and performance.

**5.2 UML Diagrams**

UML stands for Unified Modeling Language. UML is a standardized g eneral-purpose modeling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group.

The goal is for UML to become a common language for creating models of object oriented computer software. In its current form UML is comprised of two major components: a Meta-model and a notation. In the future, some form of method or process may also be added to; or associated with, UML.

The Unified Modeling Language is a standard language for specifying, Visualization, Constructing and documenting the artifacts of software system, as well as for business modeling and other non-software systems.

The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems.

The UML is a very important part of developing objects oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

**Goals**

The Primary goals in the design of the UML are as follows:

1. Provide users a ready-to-use, expressive visual modeling Language so that they can develop and exchange meaningful models.
2. Provide extendibility and specialization mechanisms to extend the core concepts.
3. Be independent of particular programming languages and development process.
4. Provide a formal basis for understanding the modeling language.
5. Encourage the growth of OO tools market.
6. Support higher level development concepts such as collaborations, frameworks, patterns and components.
7. Integrate best practices.

**5.3 Use Case Diagram**

A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.

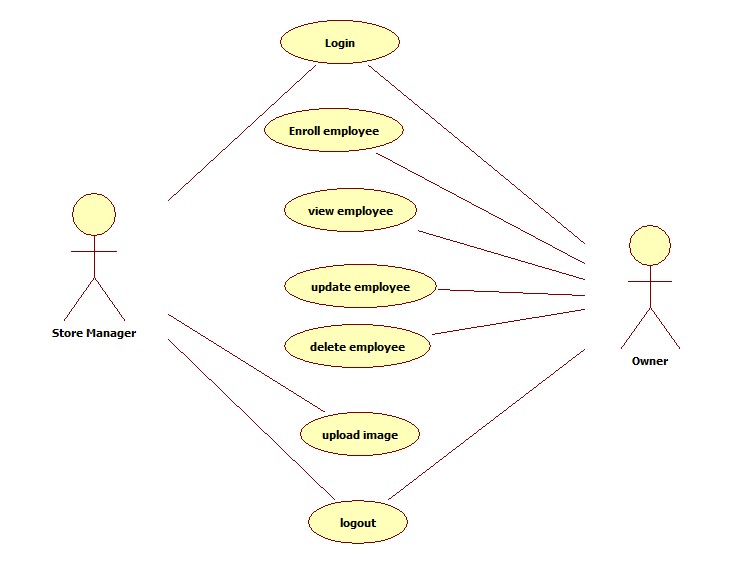


Fig-5.2: Roles of actors in the system

**5.3.1 Overview**

The system is designed for employee management, allowing users to perform various operations related to employee records. It follows a structured approach where different roles (Store Manager and Owner) have access to specific functionalities.

**Workflow Explanation**

The system consists of the following steps:

1. **User Authentication**
   * Users (Store Manager and Owner) must log in to access the system.
2. **Employee Management**
   * **Enroll Employee**: Users can add new employees to the system.
   * **View Employee**: Users can retrieve and view employee details.
   * **Update Employee**: Users can modify employee records.
   * **Delete Employee**: Users can remove employee records from the system.
3. **Image Upload**
   * Users can upload employee-related images for record-keeping.
4. **Logout**
   * Users can log out of the system to terminate their session.

This system ensures efficient employee management by providing role-based access to functionalities, with both the Store Manager and Owner having control over employee records.

**5.4 Class Diagram**

In software engineering, a class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among the classes. It explains which class contains information.

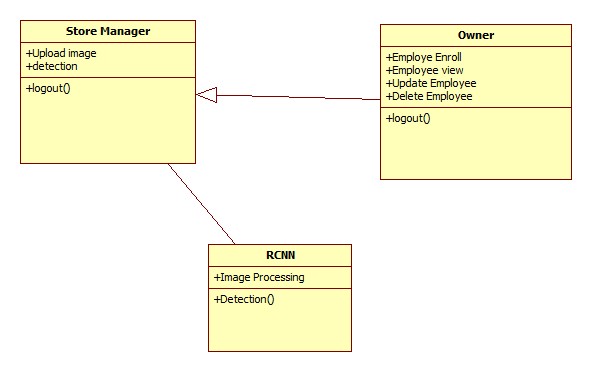


Fig-5.3: Class Diagram for the relationship among the classes

**5.4.1 Overview**

The system is designed for employee management with an additional image processing and detection functionality. It consists of three main entities: Store Manager, Owner, and RCNN (Region-based Convolutional Neural Network), each with distinct roles and responsibilities.

**Workflow Explanation**

1. **Store Manager**
   * **Upload Image**: Allows the Store Manager to upload images related to employees.
   * **Detection**: Uses image processing techniques for analysis.
   * **Logout()**: Ends the session for the Store Manager.
2. **Owner**
   * **Employee Enroll**: Adds new employees to the system.
   * **Employee View**: Retrieves and displays employee details.
   * **Update Employee**: Modifies employee records.
   * **Delete Employee**: Removes employee records from the system.
   * **Logout()**: Ends the session for the Owner.
3. **RCNN (Region-based Convolutional Neural Network)**
   * **Image Processing**: Processes uploaded images for further analysis.
   * **Detection()**: Performs object detection on images, assisting in employee-related image recognition.

This system ensures efficient employee management while integrating image detection capabilities for enhanced automation and accuracy.

**5.5 Sequence Diagram**

A sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. Sequence diagrams are sometimes called event diagrams, event scenarios, and timing diagrams.

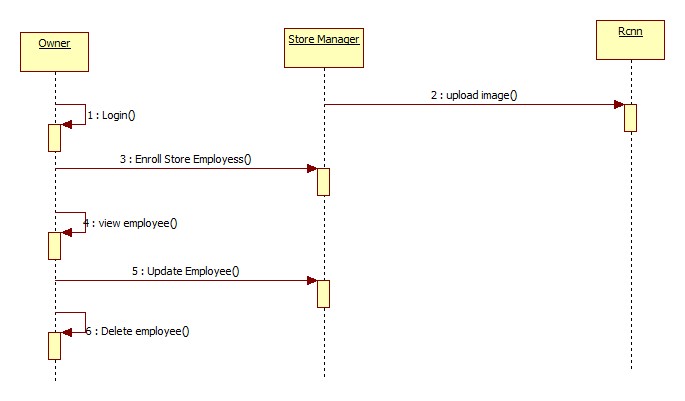


Fig-5.4: Sequence of the store management

**5.5.1 Overview**

The system represents an employee management workflow that integrates image processing capabilities. It consists of three main entities: **Owner, Store Manager, and RCNN**, each handling specific operations.

**Workflow Explanation**

1. **Owner Actions:**
   * **Login()**: The Owner logs into the system.
   * **Enroll Store Employees()**: The Owner registers new employees.
   * **View Employee()**: Retrieves employee details.
   * **Update Employee()**: Modifies employee records.
   * **Delete Employee()**: Removes an employee from the system.
2. **Store Manager Actions:**
   * **Upload Image()**: The Store Manager uploads an image for processing.
3. **RCNN (Region-based Convolutional Neural Network) Actions:**
   * **Receives Uploaded Image**: The system processes the uploaded image for detection and recognition.

This system ensures a structured approach to employee management while integrating image processing for enhanced automation and efficiency.

**5.6 Activity Diagram**

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.

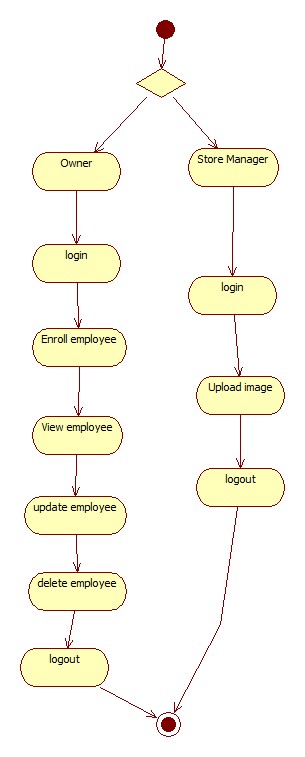


Fig-5.5: Work Flow of Activities in the Store management

**5.6.1 Overview**

This is an **Activity Diagram** that represents the workflow of an employee management system. It shows the processes followed by two different user roles: **Owner** and **Store Manager**, from login to logout.

**Workflow Breakdown**

1. **Decision Node (Initial Choice):**
   * The system starts by determining whether the user is an **Owner** or a **Store Manager**.
2. **Owner Workflow:**
   * **Login** → The Owner logs into the system.
   * **Enroll Employee** → The Owner adds new employees to the system.
   * **View Employee** → Retrieves employee details.
   * **Update Employee** → Modifies employee details.
   * **Delete Employee** → Removes an employee from the system.
   * **Logout** → Ends the Owner's session.
3. **Store Manager Workflow:**
   * **Login** → The Store Manager logs into the system.
   * **Upload Image** → Uploads an image (likely for processing or identification purposes).
   * **Logout** → Ends the Store Manager's session.
4. **Final State:**
   * Both roles eventually lead to a common **end node**, signifying the conclusion of their processes.

**Key Takeaways:**

* The Owner has **employee management** capabilities, including adding, viewing, updating, and deleting employee records.
* The Store Manager is **limited to uploading images** and logging in/out.
* The system ensures that users follow a structured path based on their roles before logging out.

This diagram provides a **clear visual representation** of how the employee management system operates, ensuring a streamlined workflow.

**CHAPTER-06**

**IMPLEMENTATION**

* 1. **Modules:**
* Home Module
* Store Manager Module
* Owner Module
* Inventory Management Module
* Notification Module
* Reporting Module

### 6.2 Modules Description

### Home Module

**6.2.1.1Purpose**

The Home Module serves as the main entry point for users accessing the system. It is designed to provide an immediate overview of the system’s operational status through a user-friendly dashboard that displays key metrics such as inventory statistics and alerts. This module directs users to the appropriate subsequent modules based on their roles, ensuring a seamless navigation experience right from the first interaction.

### 6.2.1.2 Features

The Home Module is engineered with a clear and intuitive interface that allows users to quickly comprehend the current state of the system. It dynamically presents essential information through visual summaries and metrics, making it easy for both owners and store managers to monitor performance indicators. Additionally, the module includes navigation elements that guide users to authentication pages and other functionalities, effectively setting stage for the operations to follow.

### 6.2.1.3Workflow & Dependencies

When a user accesses the system, the Home Module immediately loads a dashboard that aggregates the latest operational data. The module determines the user’s role upon login and directs them to either the Owner or Store Manager module accordingly. Its smooth operation depends on the Django framework for the backend and modern front-end libraries for rendering the dashboard and UI components, ensuring that data is both timely and accurately displayed.

### Store Manager Module

* + - 1. **Purpose**

The Store Manager Module is tailored to assist store managers in handling inventory data effectively. Its primary role is to facilitate the upload and management of inventory-related information, ensuring that data is processed accurately for subsequent analysis. This module is central to the day-to-day operations, as it allows store managers to maintain and update datasets critical for inventory management.

### Features

Store managers, once enrolled by the Owner, can log in to this module using their verified credentials. The module supports the upload of images that correspond to specific dataset columns, thereby integrating seamlessly. Furthermore, it includes functionality for initiating a data cleaning process, which converts raw data into a standardized format ready for analysis. This continuous data update capability helps maintain an accurate and up-to-date inventory record.

### Workflow & Dependencies

The workflow begins with the Owner enrolling a store manager, after which the manager logs into the system. Once authenticated, the store manager uploads images that are then mapped to the relevant columns in the inventory dataset. Upon initiating the data preparation process, the module cleans the data and displays the processed information along with visual graphs. The module’s effective operation relies on Django’s robust authentication system, specialized image processing libraries, and data visualization tools that work together to ensure accuracy and responsiveness.

### Owner Module

* + - 1. **Purpose**

The Owner Module is designed to provide comprehensive administrative oversight and control over the system. It allows the Owner to manage the roster of store managers, ensuring that all personnel details are accurate and current. Through this module, the Owner gains the ability to enroll new store managers, monitor system usage, and maintain employee records, thereby ensuring that the operational side of the business is well-organized.

### Features

This module emphasizes secure access through a robust login process, guaranteeing that only authorized users can perform administrative tasks. Once logged in, the Owner can enroll new store managers into the system and view a detailed list of current employees. Additionally, the module allows for updates to employee information and the removal of records for resigned employees, ensuring that the database remains current and reflective of the actual workforce.

### Workflow & Dependencies

The Owner logs into the system through a secure authentication process and gains access to a dedicated administrative dashboard. From here, the Owner can perform enrollment, update, or deletion operations on store manager records. The workflow is streamlined to support efficient management of personnel data, relying on Django’s role-based access control and user management systems along with a robust database backend that securely stores and manages employee records.

### Inventory Management Module

* + - 1. **Purpose**

The Inventory Management Module is responsible for monitoring stock levels and identifying empty shelf regions within the retail environment. Its main goal is to provide real-time insights into inventory status, enabling prompt detection of low-stock situations and empty shelves. This proactive approach helps in maintaining optimal inventory levels and supports timely restocking decisions.

### Features

This module utilizes a pre-trained Faster R-CNN model to detect vacant shelf areas, ensuring that only the relevant parts of the image are processed without compromising privacy. It further integrates advanced techniques such as Optical Character Recognition (OCR) and Convolutional Neural Networks (CNN) to recognize item labels and accurately assess stock levels. By processing images in real time, the module generates actionable recommendations for restocking, which can be immediately reviewed by relevant personnel.

### Workflow & Dependencies

The operational workflow of the Inventory Management Module begins with capturing inventory images, which are then processed using sophisticated image analysis algorithms. The system detects empty shelf regions and evaluates stock levels based on numerical data extracted from the images. Upon analysis, the module updates the inventory status and triggers restocking alerts if necessary. This module relies heavily on machine learning models like Faster R-CNN, OCR and CNN libraries, as well as a cloud-based database system for real-time data storage and retrieval.

### Notification Module

* + - 1. **Purpose**

The Notification Module is engineered to ensure that all stakeholders remain informed about critical inventory and system events. Its purpose is to provide real-time alerts to both store managers and owners, enabling them to react swiftly to changes in inventory levels or process updates. By automating the alerting process, this module plays a crucial role in maintaining operational efficiency.

### Features

Designed to work seamlessly in the background, the Notification Module automatically dispatches alerts via email or SMS. It monitors system events such as low inventory thresholds or the completion of data processing tasks, ensuring that users receive timely notifications. This integration of messaging services into the system ensures that critical information is delivered without delay, supporting prompt decision-making and action.

### Workflow & Dependencies

In practice, the Notification Module continuously monitors inventory and processing events. When a defined threshold or significant event is detected, the module triggers an alert that is automatically sent to the relevant users. The process is supported by robust messaging services, including SMTP for email notifications and SMS gateways for text alerts. The module's effectiveness is further enhanced by its integration with other modules, such as Inventory Management and data processing, ensuring that alerts are generated based on real-time system conditions.

### Reporting Module

* + - 1. **Purpose**

The Reporting Module is designed to deliver comprehensive insights into the system’s performance, particularly regarding inventory status and sales trends. It serves as a critical tool for decision-makers by providing detailed reports that inform strategic planning and operational adjustments. The module’s analytical capabilities support a data-driven approach to managing inventory and monitoring business performance.

### Features

This module offers customizable dashboards that present data in the form of graphs and charts, making complex information easier to understand and act upon. Users have the flexibility to export reports in various formats, such as PDF and Excel, to suit different analytical and documentation needs. Additionally, the module leverages historical data to perform trend analysis, thereby enabling forecasts and proactive management of future inventory requirements.

### Workflow & Dependencies

The Reporting Module aggregates data from various parts of the system, including the Inventory Management and operational logs, to generate detailed reports. Once the data is compiled, the module dynamically creates visual dashboards that accurately represent both current and historical performance metrics. Users can then review these reports, export them for further analysis, or use the insights for strategic planning. The smooth operation of this module depends on advanced data visualization libraries and integrated reporting engines within Django, as well as reliable database connectivity to ensure that data is continuously updated.

**6.3 Python**

Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language. An [interpreted language,](https://en.wikipedia.org/wiki/Interpreted_language) Python has a design philosophy that emphasizes code [readability](https://en.wikipedia.org/wiki/Readability) (notably using [whitespace](https://en.wikipedia.org/wiki/Whitespace_character) indentation to delimit [code](https://en.wikipedia.org/wiki/Code_block) [blocks](https://en.wikipedia.org/wiki/Code_block) rather than curly brackets or keywords), and a syntax that allows programmers to express concepts in fewer [lines of code](https://en.wikipedia.org/wiki/Source_lines_of_code) than might be used in languages such as [C++](https://en.wikipedia.org/wiki/C%2B%2B)or [Java.](https://en.wikipedia.org/wiki/Java_(programming_language)) It provides constructs that enable clear programming on both small and large scales. Python interpreters are available for many [operating systems.](https://en.wikipedia.org/wiki/Operating_system) [C Python,](https://en.wikipedia.org/wiki/CPython) the [reference implementation](https://en.wikipedia.org/wiki/Reference_implementation) of Python, is [open](https://en.wikipedia.org/wiki/Open_source) [source](https://en.wikipedia.org/wiki/Open_source) software and has a community-based development model, as do nearly all of its variant implementations. C Python is managed by the non-profit [Python Software Foundation.](https://en.wikipedia.org/wiki/Python_Software_Foundation) Python features a [dynamic type](https://en.wikipedia.org/wiki/Dynamic_type) system and automatic [memory management](https://en.wikipedia.org/wiki/Memory_management). It supports multiple [programming paradigms,](https://en.wikipedia.org/wiki/Programming_paradigm) including [object-](https://en.wikipedia.org/wiki/Object-oriented_programming) [oriented](https://en.wikipedia.org/wiki/Object-oriented_programming), [imperative,](https://en.wikipedia.org/wiki/Imperative_programming) [functional](https://en.wikipedia.org/wiki/Functional_programming) and [procedural,](https://en.wikipedia.org/wiki/Procedural_programming) and has a large and comprehensive [standard](https://en.wikipedia.org/wiki/Standard_library) [library.](https://en.wikipedia.org/wiki/Standard_library)

## Interactive Mode Programming

Invoking the interpreter without passing a script file as a parameter brings up the following prompt −

$ python

Python 2.4.3 (#1, Nov 11 2010, 13:34:43)

[GCC 4.1.2 20080704 (Red Hat 4.1.2-48)] on linux2

Type "help", "copyright", "credits" or "license" for more information.

>>>

Type the following text at the Python prompt and press the Enter −

>>> print "Hello, Python!"

If you are running new version of Python, then you would need to use print statement with parenthesis as in print ("Hello, Python!");. However in Python version 2.4.3, this produces the following result −

Hello, Python!

## Script Mode Programming

Invoking the interpreter with a script parameter begins execution of the script and continues until the script is finished. When the script is finished, the interpreter is no longer active.

Let us write a simple Python program in a script. Python files have extension .py. Type the following source code in a test.py file −

Live Demo

print "Hello, Python!"

We assume that you have Python interpreter set in PATH variable. Now, try to run this program as follows −

$ python test.py

This produces the following result −

Hello, Python!

Let us try another way to execute a Python script. Here is the modified test.py file −

Live Demo #!/usr/bin/python

print "Hello, Python!"

We assume that you have Python interpreter available in /usr/bin directory. Now, try to run this program as follows −

$ chmod +x test.py # This is to make file executable

$./test.py

This produces the following result −

Hello, Python!

## Python Identifiers

A Python identifier is a name used to identify a variable, function, class, module or other object. An identifier starts with a letter A to Z or a to z or an underscore (\_) followed by zero or more letters, underscores and digits (0 to 9).

Python does not allow punctuation characters such as @, $, and % within identifiers. Python is a case sensitive programming language. Thus, Manpower and manpower are two different identifiers in Python.

Here are naming conventions for Python identifiers −

Class names start with an uppercase letter. All other identifiers start with a lowercase letter. Starting an identifier with a single leading underscore indicates that the identifier is private.

Starting an identifier with two leading underscores indicates a strongly private identifier.

If the identifier also ends with two trailing underscores, the identifier is a language-defined special name.

## Reserved Words

The following list shows the Python keywords. These are reserved words and you cannot use them as constant or variable or any other identifier names. All the Python keywords contain lowercase letters only.

|  |  |  |
| --- | --- | --- |
| and | exec | not |
| assert | finally | or |
| break | for | pass |
| class | from | print |

continue global raise

def if return

del import try

elif in while

else is with except lambdayield

## Lines and Indentation

Python provides no braces to indicate blocks of code for class and function definitions or flow

control. Blocks of code are denoted by line indentation, which is rigidly enforced.

The number of spaces in the indentation is variable, but all statements within the block must be indented the same amount. For example −

if True:

print "True" else:

print "False"

However, the following block generates an error −

if True:

print "Answer" print "True" else:

print "Answer" print "False"

Thus, in Python all the continuous lines indented with same number of spaces would form a block. The following example has various statement blocks −

Note − Do not try to understand the logic at this point of time. Just make sure you understood various blocks even if they are without braces.

#!/usr/bin/python

import sys

try:

# open file stream

file = open(file\_name, "w") except IOError:

print "There was an error writing to", file\_name sys.exit()

print "Enter '", file\_finish, print "' When finished" while file\_text != file\_finish:

file\_text = raw\_input("Enter text: ") if file\_text == file\_finish:

# close the file file.close break

file.write(file\_text) file.write("\n")

file.close()

file\_name = raw\_input("Enter filename: ") if len(file\_name) == 0:

print "Next time please enter something" sys.exit()

try:

file = open(file\_name, "r") except IOError:

print "There was an error reading file" sys.exit()

file\_text = file.read() file.close()

print file\_text

Multi-Line Statements

Statements in Python typically end with a new line. Python does, however, allow the use of the line continuation character (\) to denote that the line should continue. For example −

total = item\_one + \ item\_two + \ item\_three

Statements contained within the [], {}, or () brackets do not need to use the line continuation character. For example −

days = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']

Quotation in Python

Python accepts single ('), double (") and triple (''' or """) quotes to denote string literals, as long as the same type of quote starts and ends the string.

The triple quotes are used to span the string across multiple lines. For example, all the following are legal −

word = 'word'

sentence = "This is a sentence." paragraph = """This is a paragraph. It is

made up of multiple lines and sentences.""" Comments in Python

A hash sign (#) that is not inside a string literal begins a comment. All characters after the # and up to the end of the physical line are part of the comment and the Python interpreter ignores them.

Live Demo

#!/usr/bin/python

# First comment

print "Hello, Python!" # second comment This produces the following result −

Hello, Python!

You can type a comment on the same line after a statement or expression −

name = "Madisetti" # This is again comment You can comment multiple lines as follows −

# This is a comment.

# This is a comment, too.

# This is a comment, too. # I said that already.

Following triple-quoted string is also ignored by Python interpreter and can be used as a multiline comments:

'''

This is a multiline comment.

'''

Using Blank Lines

A line containing only whitespace, possibly with a comment, is known as a blank line and Python totally ignores it.

In an interactive interpreter session, you must enter an empty physical line to terminate a multiline statement.

Waiting for the User

The following line of the program displays the prompt, the statement saying “Press the enter key to exit”, and waits for the user to take action −

#!/usr/bin/python

raw\_input("\n\nPress the enter key to exit.")

Here, "\n\n" is used to create two new lines before displaying the actual line. Once the user presses the key, the program ends. This is a nice trick to keep a console window open until the user is done with an application.

Multiple Statements on a Single Line

The semicolon ( ; ) allows multiple statements on the single line given that neither statement starts a new code block. Here is a sample snip using the semicolon.

import sys; x = 'foo'; sys.stdout.write(x + '\n') Multiple Statement Groups as Suites

A group of individual statements, which make a single code block are called suites in Python. Compound or complex statements, such as if, while, def, and class require a header line and a suite.

Header lines begin the statement (with the keyword) and terminate with a colon ( : ) and are followed by one or more lines which make up the suite. For example −

if expression :

suite

elif expression :

suite else :

suite

## Command Line Arguments

Many programs can be run to provide you with some basic information about how they should be run. Python enables you to do this with -h −

$ python -h

usage: python [option] ... [-c cmd | -m mod | file | -] [arg] ... Options and arguments (and corresponding environment variables):

-c cmd : program passed in as string (terminates option list)

-d : debug output from parser (also PYTHONDEBUG=x)

-E : ignore environment variables (such as PYTHONPATH)

-h : print this help message and exit

You can also program your script in such a way that it should accept various options. Command Line Arguments is an advanced topic and should be studied a bit later once you have gone through rest of the Python concepts.

## Python Lists

The list is a most versatile datatype available in Python which can be written as a list of comma- separated values (items) between square brackets. Important thing about a list is that items in a list need not be of the same type.

Creating a list is as simple as putting different comma-separated values between square brackets. For example −

list1 = ['physics', 'chemistry', 1997, 2000];

list2 = [1, 2, 3, 4, 5 ];

list3 = ["a", "b", "c", "d"]

Similar to string indices, list indices start at 0, and lists can be sliced, concatenated and so on. A tuple is a sequence of immutable Python objects. Tuples are sequences, just like lists. The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets.

Creating a tuple is as simple as putting different comma-separated values. Optionally you can put these comma-separated values between parentheses also. For example −

tup1 = ('physics', 'chemistry', 1997, 2000);

tup2 = (1, 2, 3, 4, 5 );

tup3 = "a", "b", "c", "d";

The empty tuple is written as two parentheses containing nothing −

tup1 = ();

To write a tuple containing a single value you have to include a comma, even though there is only one value −

tup1 = (50,);

Like string indices, tuple indices start at 0, and they can be sliced, concatenated, and so on.

Accessing Values in Tuples

To access values in tuple, use the square brackets for slicing along with the index or indices to obtain value available at that index. For example −

Live Demo #!/usr/bin/python

tup1 = ('physics', 'chemistry', 1997, 2000);

tup2 = (1, 2, 3, 4, 5, 6, 7 );

print "tup1[0]: ", tup1[0];

print "tup2[1:5]: ", tup2[1:5];

When the above code is executed, it produces the following result −

tup1[0]: physics tup2[1:5]: [2, 3, 4, 5]

Updating Tuples

Accessing Values in Dictionary

To access dictionary elements, you can use the familiar square brackets along with the key to obtain its value. Following is a simple example −

Live Demo #!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}

print "dict['Name']: ", dict['Name']

print "dict['Age']: ", dict['Age']

When the above code is executed, it produces the following result −

dict['Name']: Zara dict['Age']: 7

If we attempt to access a data item with a key, which is not part of the dictionary, we get an error as follows −

Live Demo #!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}

print "dict['Alice']: ", dict['Alice']

When the above code is executed, it produces the following result −

dict['Alice']:

Traceback (most recent call last): File "test.py", line 4, in <module>

print "dict['Alice']: ", dict['Alice'];

KeyError: 'Alice' Updating Dictionary

You can update a dictionary by adding a new entry or a key-value pair, modifying an existing entry, or deleting an existing entry as shown below in the simple example −

Live Demo #!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'} dict['Age'] = 8; # update existing entry dict['School'] = "DPS School"; # Add new entry

print "dict['Age']: ", dict['Age']

print "dict['School']: ", dict['School']

When the above code is executed, it produces the following result −

dict['Age']: 8 dict['School']: DPS School Delete Dictionary Elements

You can either remove individual dictionary elements or clear the entire contents of a dictionary. You can also delete entire dictionary in a single operation.

To explicitly remove an entire dictionary, just use the del statement. Following is a simple example −

Live Demo #!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'} del dict['Name']; # remove entry with key 'Name' dict.clear(); # remove all entries in dict

del dict ; # delete entire dictionary

print "dict['Age']: ", dict['Age']

print "dict['School']: ", dict['School']

This produces the following result. Note that an exception is raised because after del dict dictionary does not exist any more −

dict['Age']:

Traceback (most recent call last):

File "test.py", line 8, in <module> print "dict['Age']: ", dict['Age'];

TypeError: 'type' object is unsubscriptable

Note − del() method is discussed in subsequent section.

## Properties of Dictionary Keys

Dictionary values have no restrictions. They can be any arbitrary Python object, either standard objects or user-defined objects. However, same is not true for the keys.

There are two important points to remember about dictionary keys −

(a) More than one entry per key not allowed. Which means no duplicate key is allowed. When duplicate keys encountered during assignment, the last assignment wins. For example −

Live Demo

#!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7, 'Name': 'Manni'}

print "dict['Name']: ", dict['Name']

When the above code is executed, it produces the following result −

dict['Name']: Manni

1. Keys must be immutable. Which means you can use strings, numbers or tuples as dictionary keys but something like ['key'] is not allowed. Following is a simple example −

Live Demo

#!/usr/bin/python

dict = {['Name']: 'Zara', 'Age': 7}

print "dict['Name']: ", dict['Name']

When the above code is executed, it produces the following result −

Traceback (most recent call last):

File "test.py", line 3, in <module> dict = {['Name']: 'Zara', 'Age': 7};

TypeError: unhashable type: 'list'

Tuples are immutable which means you cannot update or change the values of tuple elements. You are able to take portions of existing tuples to create new tuples as the following example demonstrates −

Live Demo #!/usr/bin/python

tup1 = (12, 34.56);

tup2 = ('abc', 'xyz');

# Following action is not valid for tuples # tup1[0] = 100;

# So let's create a new tuple as follows

tup3 = tup1 + tup2;

print tup3;

When the above code is executed, it produces the following result −

(12, 34.56, 'abc', 'xyz')

Delete Tuple Elements

Removing individual tuple elements is not possible. There is, of course, nothing wrong with putting together another tuple with the undesired elements discarded.

To explicitly remove an entire tuple, just use the del statement. For example −

Live Demo #!/usr/bin/python

tup = ('physics', 'chemistry', 1997, 2000); print tup;

del tup;

print "After deleting tup : "; print tup;

This produces the following result. Note an exception raised, this is because after del tup tuple does not exist any more −

('physics', 'chemistry', 1997, 2000) After deleting tup :

Traceback (most recent call last):

File "test.py", line 9, in <module> print tup;

NameError: name 'tup' is not defined

* 1. **DJANGO**

Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design. Built by experienced developers, it takes care of much of the hassle of Web development, so you can focus on writing your app without needing to reinvent the wheel. It’s free and open source.

Django's primary goal is to ease the creation of complex, database-driven websites. Django emphasizes [reusability](https://en.wikipedia.org/wiki/Reusability)and "pluggability" of components, rapid development, and the principle of [don't repeat yourself.](https://en.wikipedia.org/wiki/Don%27t_repeat_yourself) Python is used throughout, even for settings files and data models.

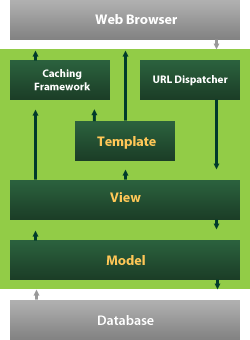


Fig-6.1: Interaction between Web Browser and Database using Django

Django also provides an optional administrative [create, read, update and delete](https://en.wikipedia.org/wiki/Create%2C_read%2C_update_and_delete) interface that is generated dynamically through [introspection](https://en.wikipedia.org/wiki/Introspection_(computer_science)) and configured via admin models



Fig-6.2: Communication between User and Client

## Create a Project

Whether you are on Windows or Linux, just get a terminal or a cmd prompt and navigate to the place you want your project to be created, then use this code −

$ django-admin startproject myproject

This will create a "myproject" folder with the following structure −

myproject/ manage.py myproject/

init .py settings.py urls.py wsgi.py

The Project Structure

The “myproject” folder is just your project container, it actually contains two elements −

manage.py − This file is kind of your project local django-admin for interacting with your project via command line (start the development server, sync db...). To get a full list of command accessible via manage.py you can use the code −

$ python manage.py help

The “myproject” subfolder − This folder is the actual python package of your project. It contains four files −

init .py − Just for python, treat this folder as package. settings.py − As the name indicates, your project settings.

urls.py − All links of your project and the function to call. A kind of ToC of your project.

wsgi.py − If you need to deploy your project over WSGI.

Setting Up Your Project

Your project is set up in the subfolder myproject/settings.py. Following are some important options you might need to set −

DEBUG = True

This option lets you set if your project is in debug mode or not. Debug mode lets you get more

information about your project's error. Never set it to ‘True’ for a live project. However, this has to be set to ‘True’ if you want the Django light server to serve static files. Do it only in the development mode.

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.sqlite3', 'NAME': 'database.sql',

'USER': '',

'PASSWORD': '',

'HOST': '',

'PORT': '',

}

}

Database is set in the ‘Database’ dictionary. The example above is for SQLite engine. As stated earlier, Django also supports −

MySQL (django.db.backends.mysql)

PostGreSQL (django.db.backends.postgresql\_psycopg2) Oracle (django.db.backends.oracle) and NoSQL DB MongoDB (django\_mongodb\_engine)

Before setting any new engine, make sure you have the correct db driver installed.

You can also set others options like: TIME\_ZONE, LANGUAGE\_CODE, TEMPLATE…

Now that your project is created and configured make sure it's working −

$ python manage.py runserver

You will get something like the following on running the above code − Validating models...

0 errors found

September 03, 2015 - 11:41:50

Django version 1.6.11, using settings 'myproject.settings' Starting development server at <http://127.0.0.1:8000/> Quit the server with CONTROL-C.

A project is a sum of many applications. Every application has an objective and can be reused into another project, like the contact form on a website can be an application, and can be reused for others. See it as a module of your project.

## Create an Application

We assume you are in your project folder. In our main “myproject” folder, the same folder then manage.py −

$ python manage.py startapp myapp

You just created myapp application and like project, Django create a “myapp” folder with the application structure −

myapp/

init .py admin.py models.py tests.py views.py

init .py − Just to make sure python handles this folder as a package.

admin.py − This file helps you make the app modifiable in the admin interface. models.py − This is where all the application models are stored.

tests.py − This is where your unit tests are.

views.py − This is where your application views are.

Get the Project to Know About Your Application

At this stage we have our "myapp" application, now we need to register it with our Django project "myproject". To do so, update INSTALLED\_APPS tuple in the settings.py file of your project (add your app name) −

INSTALLED\_APPS = (

'django.contrib.admin', 'django.contrib.auth', 'django.contrib.contenttypes', 'django.contrib.sessions', 'django.contrib.messages', 'django.contrib.staticfiles', 'myapp',

)

Creating forms in Django, is really similar to creating a model. Here again, we just need to inherit from Django class and the class attributes will be the form fields. Let's add a forms.py file in myapp folder to contain our app forms. We will create a login form.

myapp/forms.py

#-\*- coding: utf-8 -\*- from django import forms

class LoginForm(forms.Form):

user = forms.CharField(max\_length = 100)

password = forms.CharField(widget = forms.PasswordInput())

As seen above, the field type can take "widget" argument for html rendering; in our case, we want the password to be hidden, not displayed. Many others widget are present in Django: DateInput for dates, CheckboxInput for checkboxes, etc.

Using Form in a View

There are two kinds of HTTP requests, GET and POST. In Django, the request object passed as parameter to your view has an attribute called "method" where the type of the request is set, and

all data passed via POST can be accessed via the request.POST dictionary. Let's create a login view in our myapp/views.py –

#-\*- coding: utf-8 -\*-

from myapp.forms import LoginForm

def login(request):

username = "not logged in"

if request.method == "POST":

#Get the posted form

MyLoginForm = LoginForm(request.POST)

if MyLoginForm.is\_valid():

username = MyLoginForm.cleaned\_data['username'] else:

MyLoginForm = Loginform()

return render(request, 'loggedin.html', {"username" : username})

The view will display the result of the login form posted through the loggedin.html. To test it, we will first need the login form template. Let's call it login.html.

<html>

<body>

<form name = "form" action = "{% url "myapp.views.login" %}" method = "POST" >{% csrf\_token %}

<div style = "max-width:470px;">

<center>

<input type = "text" style = "margin-left:20%;" placeholder = "Identifiant" name = "username" />

</center>

</div>

<br>

<div style = "max-width:470px;">

<center>

<input type = "password" style = "margin-left:20%;" placeholder = "password" name = "password" />

</center>

</div>

<br>

<div style = "max-width:470px;">

<center>

<button style = "border:0px; background-color:#4285F4; margin-top:8%; height:35px; width:80%;margin-left:19%;" type = "submit"

value = "Login" >

<strong>Login</strong>

</button>

</center>

</div>

</form>

</body>

</html>

The template will display a login form and post the result to our login view above. You have probably noticed the tag in the template, which is just to prevent Cross-site Request Forgery (CSRF) attack on your site.

{% csrf\_token %}

Once we have the login template, we need the loggedin.html template that will be rendered after form treatment.

<html>

<body>

You are : <strong>{{username}}</strong>

</body>

</html>

Now, we just need our pair of URLs to get started: myapp/urls.py

from django.conf.urls import patterns, url

from django.views.generic import TemplateView

urlpatterns = patterns('myapp.views', url(r'^connection/',TemplateView.as\_view(template\_name = 'login.html')), url(r'^login/', 'login', name = 'login'))

When accessing "/myapp/connection", we will get the following login.html template rendered − Setting Up Sessions

In Django, enabling session is done in your project settings.py, by adding some lines to the MIDDLEWARE\_CLASSES and the INSTALLED\_APPS options. This should be done while creating the project, but it's always good to know, so MIDDLEWARE\_CLASSES should have

'django.contrib.sessions.middleware.SessionMiddleware' And INSTALLED\_APPS should have −

'django.contrib.sessions'

By default, Django saves session information in database (django\_session table or collection), but you can configure the engine to store information using other ways like: in file or in cache.

When session is enabled, every request (first argument of any view in Django) has a session (dict) attribute.

Let's create a simple sample to see how to create and save sessions. We have built a simple login system before (see Django form processing chapter and Django Cookies Handling chapter). Let us save the username in a cookie so, if not signed out, when accessing our login page you won’t see the login form. Basically, let's make our login system we used in Django Cookies handling more secure, by saving cookies server side.

For this, first lets change our login view to save our username cookie server side −

def login(request):

username = 'not logged in'

if request.method == 'POST':

MyLoginForm = LoginForm(request.POST)

if MyLoginForm.is\_valid():

username = MyLoginForm.cleaned\_data['username'] request.session['username'] = username

else:

MyLoginForm = LoginForm()

return render(request, 'loggedin.html', {"username" : username}

Then let us create formView view for the login form, where we won’t display the form if cookie is set −

def formView(request):

if request.session.has\_key('username'):

username = request.session['username']

return render(request, 'loggedin.html', {"username" : username}) else:

return render(request, 'login.html', {})

Now let us change the url.py file to change the url so it pairs with our new view −

from django.conf.urls import patterns, urlfrom django.views.generic import TemplateView

urlpatterns = patterns('myapp.views', url(r'^connection/','formView', name = 'loginform'), url(r'^login/', 'login', name = 'login'))

When accessing /myapp/connection, you will get to see the following page

# CHAPTER-07

**TESTING AND EVALUATION**

# 7.1 System Test

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub assemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

* 1. **Type of Tests**

**7.2.1 Unit Testing**

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

**7.2.2 Integration Tests**

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

**7.2.3 Functional Tests**

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

**Valid Input** : identified classes of valid input must be accepted.

**Invalid Input** : identified classes of invalid input must be rejected.

**Functions** : identified functions must be exercised.

**Output** : identified classes of application outputs must be exercised.

**Systems/Procedures** : interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

### 7.2.4 System Test

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

### 7.2.5 White Box Testing

White Box Testing is a testing in which in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is purpose. It is used to test areas that cannot be reached from a black box level.

**7.2.6 Black Box Testing**

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box .you cannot “see” into it. The test provides inputs and responds to outputs without considering how the software works.

## Test strategy and approach

* Field testing will be performed manually and functional tests will be written in detail.

### 7.2.6.2 Test objectives

* All field entries must work properly.
* Pages must be activated from the identified link.
* The entry screen, messages and responses must not be delayed.

### 7.2.6.3 Features to be tested

* Verify that the entries are of the correct format
* No duplicate entries should be allowed
* All links should take the user to the correct page.

### 7.2.7 Integration Testing

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.

The task of the integration test is to check that components or software applications, e.g. components in a software system or – one step up – software applications at the company level – interact without error.

**Test Results:** All the test cases mentioned above passed successfully. No defects encountered.

* + 1. **User Acceptance Testing**

Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

**Test Results:** All the test cases mentioned above passed successfully. No defects encountered.

**Test Cases**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S.no** | **Test Case** | **Excepted Result** | **Result** | **Remarks(IF Fails)** |
| 1. | User Register | If User registration successfully. | Pass | If already user email exist then it fails. |
| 2. | User Login | If Username and password is correct then it will getting valid page. | Pass | Un Register Users will not logged in. |
| 3. | Admin Add the Data | A new record will added to our dataset. | Pass | Data types not match then this case failed |
| 4. | Data Cleaning | Data will be cleaned. | Pass | The data will be in in or float format, otherwise algorithm will not work.. |
| 5. | Mean Square error calculates | Means Square error calculation | Pass | If noisy data there then the result will not accurate |
| 6. | Mean Absolute Error calculation | Selected Model Mean Absolute Error calculated and sent to browser. | Pass | If noisy data there then the result will not accurate. |
| 7. | Explained  Variance  Score calculated | Selected Model Explained  VarianceScore calculatedand sent to browser | Pass | If noisy data there then the result will not accurate. |
| 8. | Median  Absolute  Error calculated | Median Absolute Error Calculated and sent to controller | Pass | If noisy data there then the result will not accurate |
| 9. | R2\_Score Calculated | R2\_Score Calculated and send to browser | Pass | If noisy data there then the result will not accurate |
| 10. | Admin can activate the register users | Admin can activate the register user id | Pass | If user id not found then it won’t login. |

Table 7.1: Test Cases

**CHAPTER-08**

**RESULTS**

**8.1 Homa Page:**

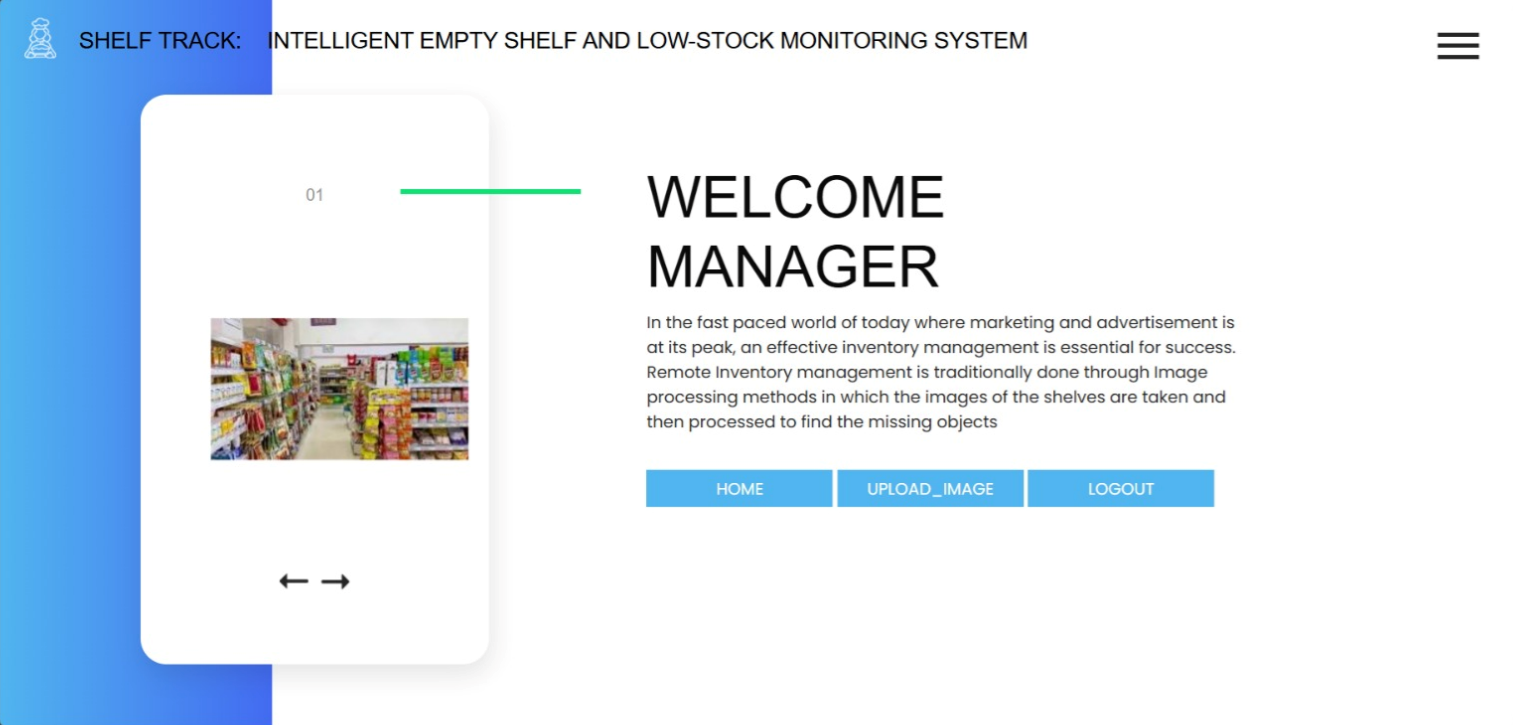
****

Fig-8.1: Home page

**8.2 Owner Login Page :**

****

Fig-8.2: Owner login page

**8.3 Store Manager Login :**

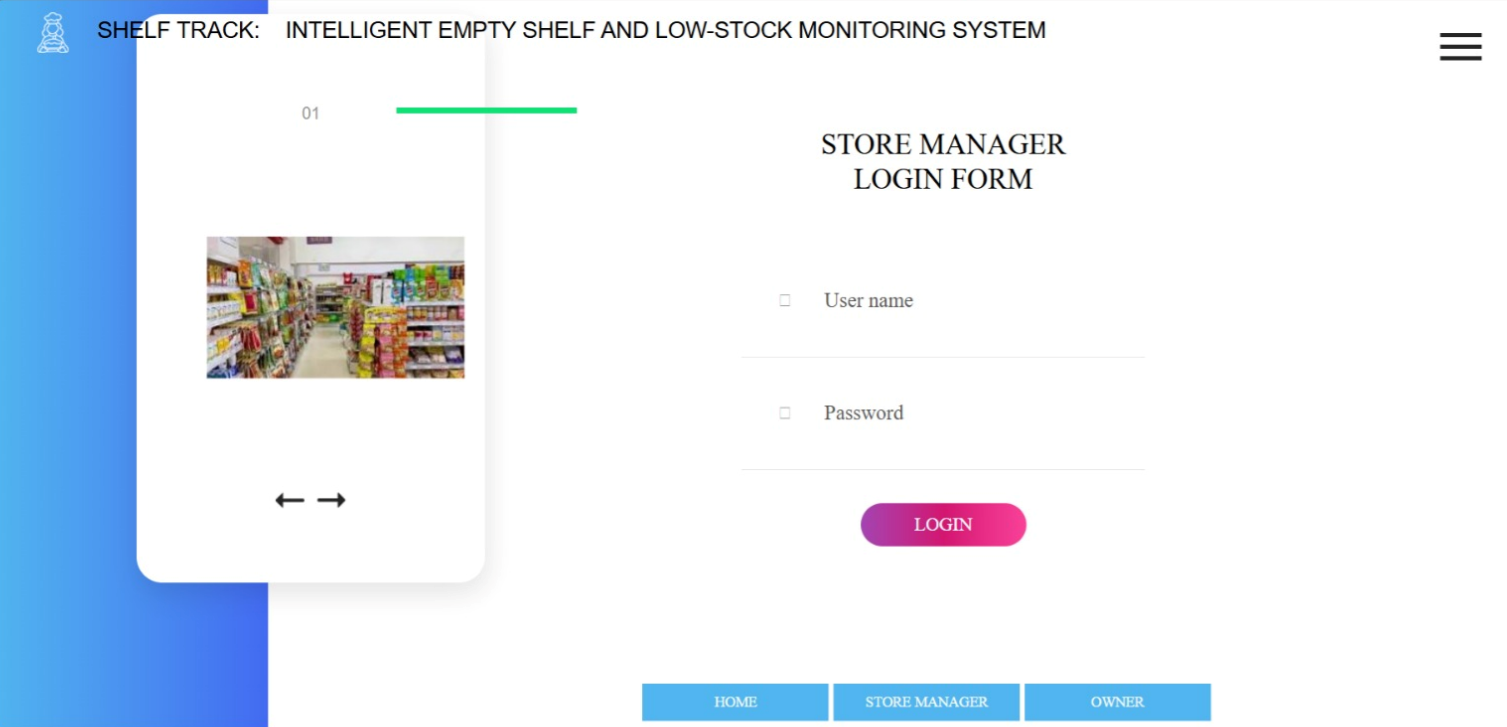
****

Fig-8.3: Store manager login

**8.4 Owner Home Page :**

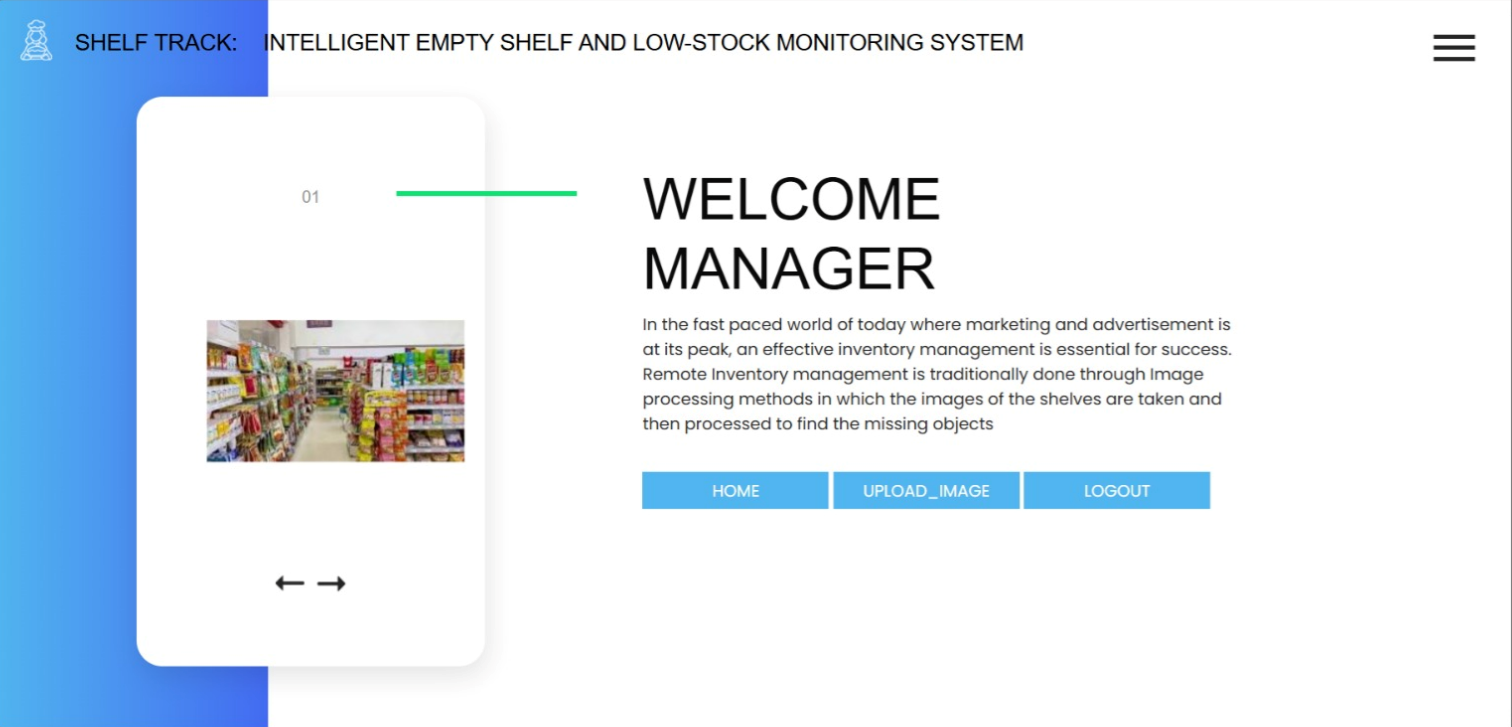
****

Fig-8.4: Owner home page

**8.5 Employee Enroll Form :**

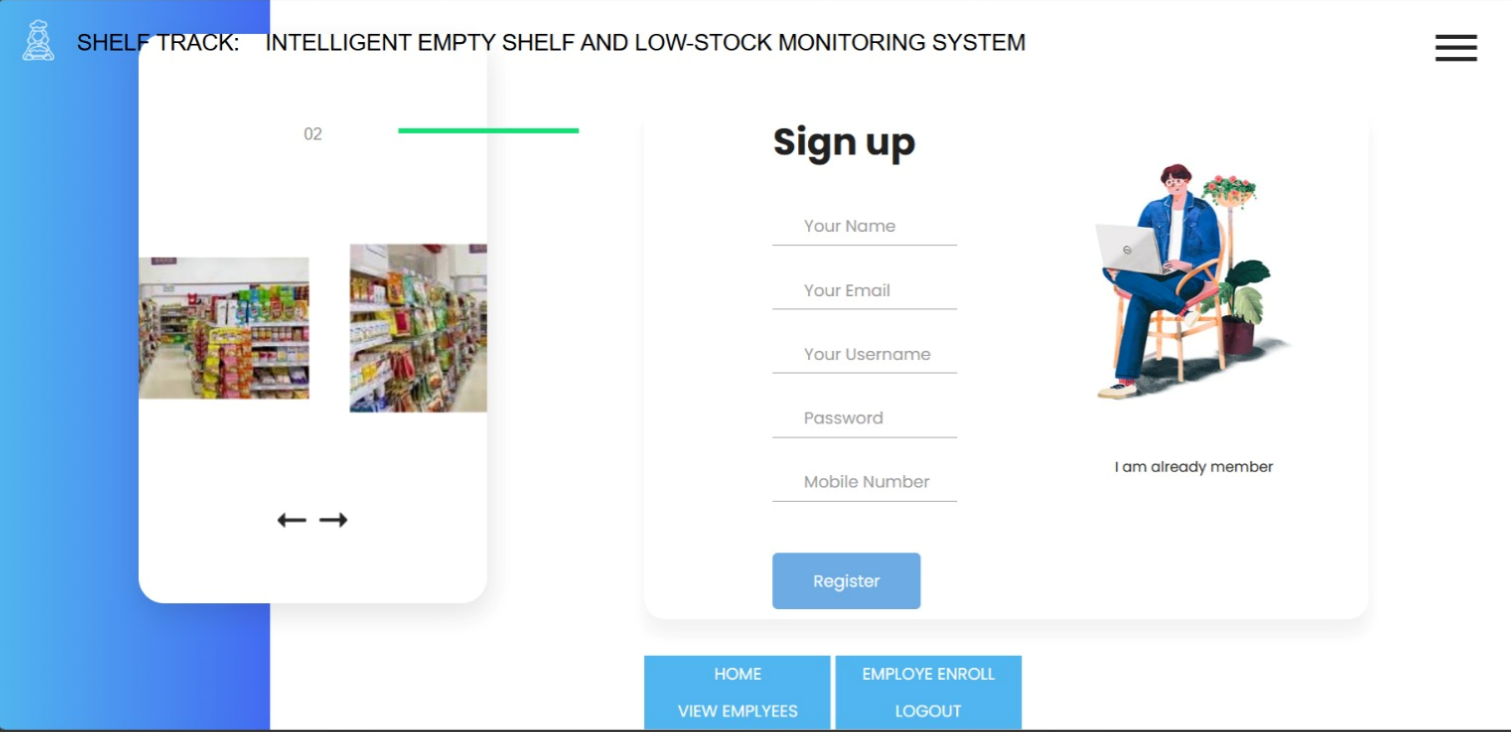
****

Fig-8.5: Employee Enroll form

**8.6 Employee View Form :**

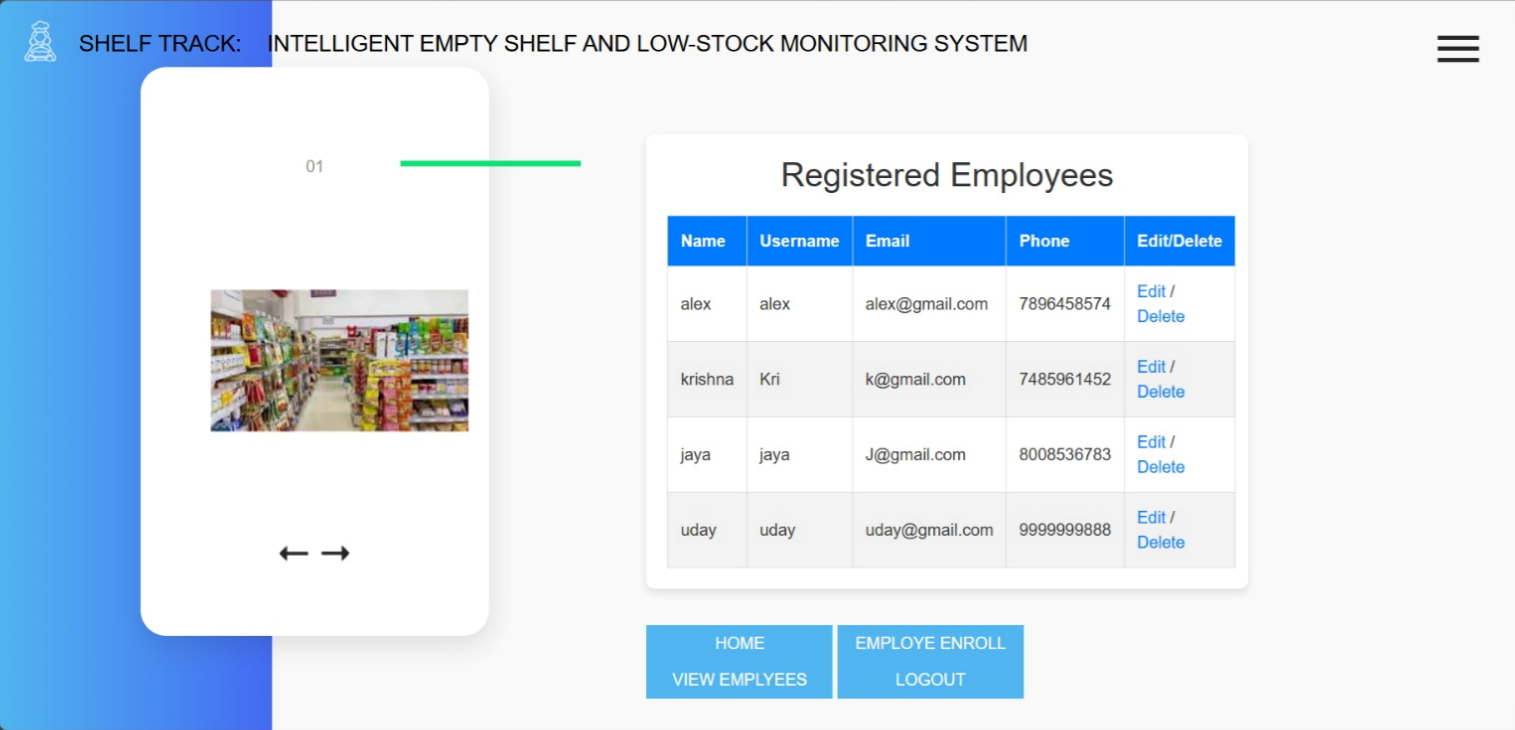
****

Fig-8.6: Employee view form

**8.7 Employee Update Form :**

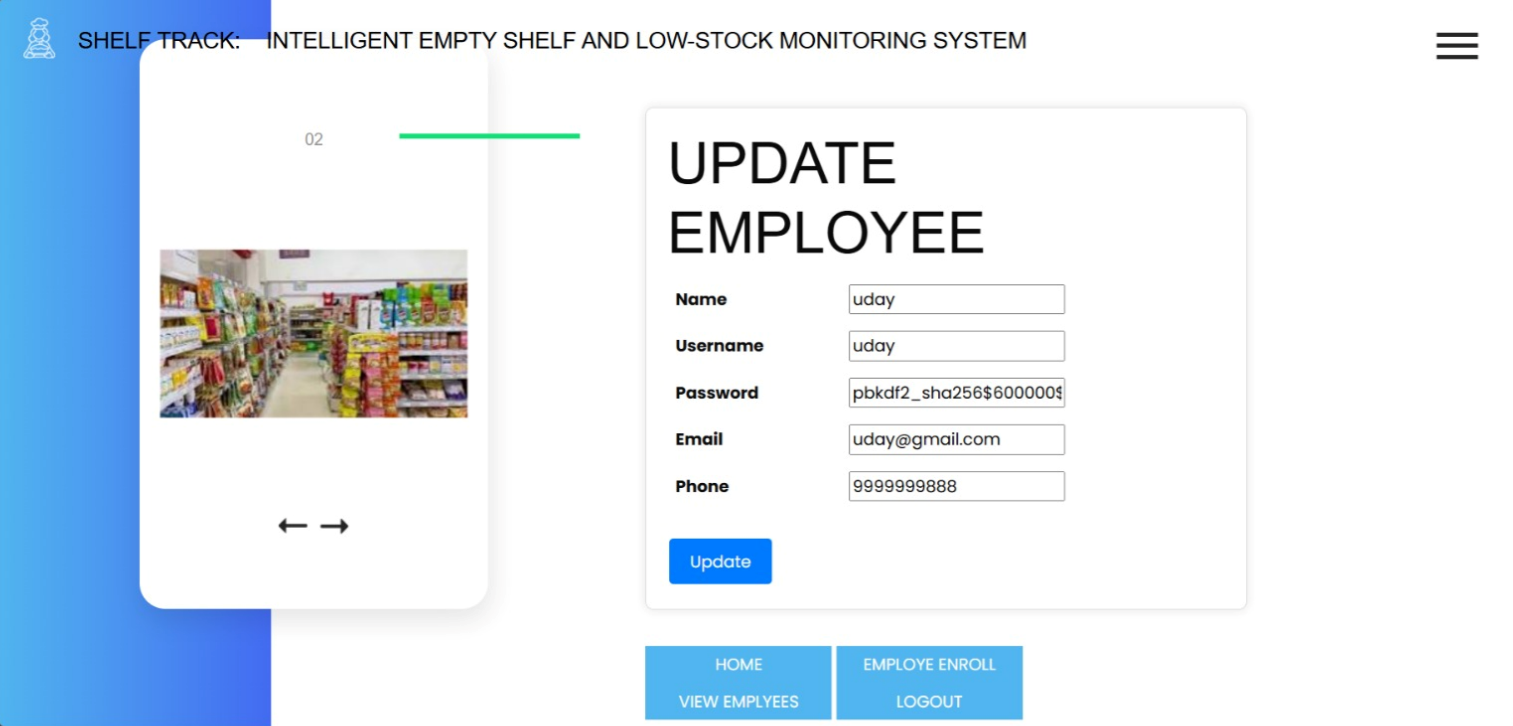
****

Fig-8.7: Employee update form

**8.8 Store Manager Home Page :**

****

Fig-8.8: Store manager home page

**8.9 Image Upload Form :**

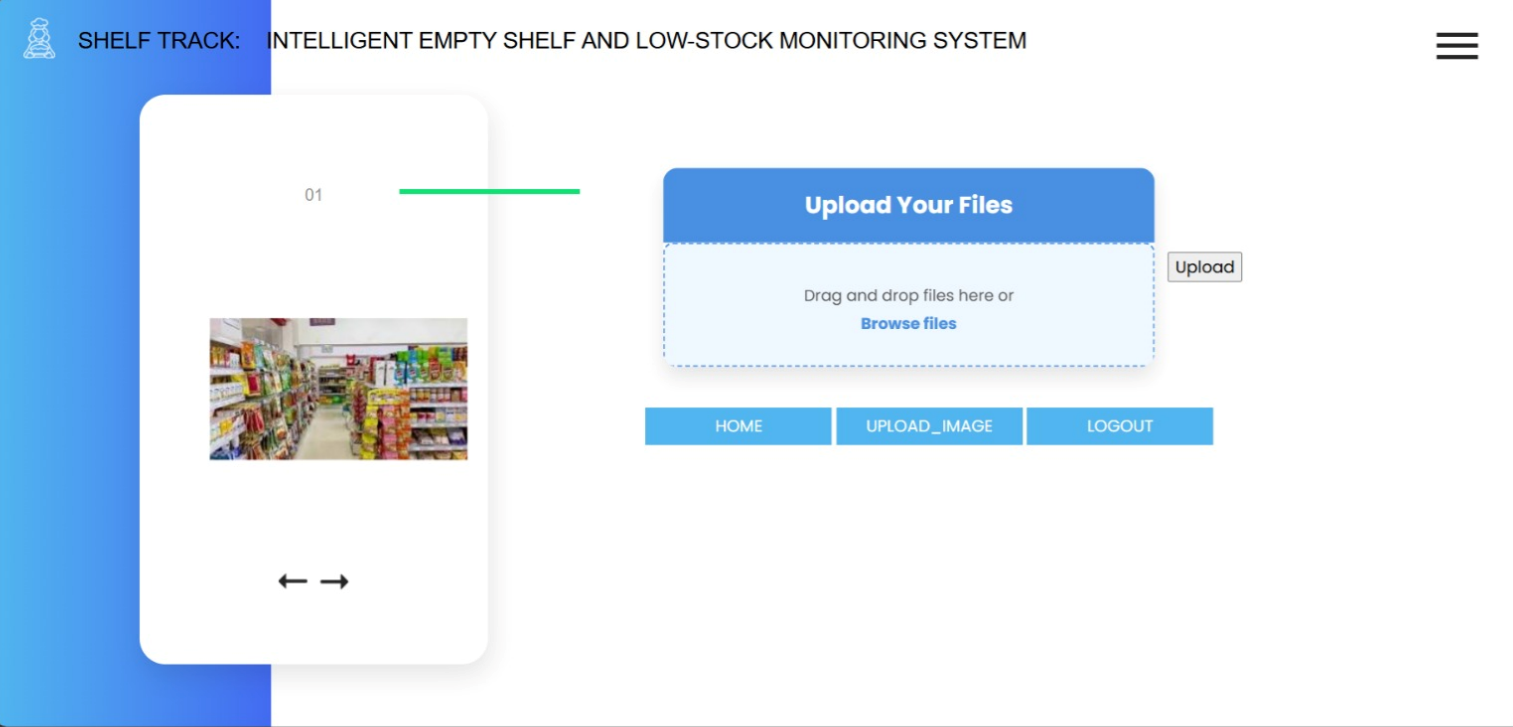
****

Fig-8.9: Image upload form

**8.10 Inventory List:**

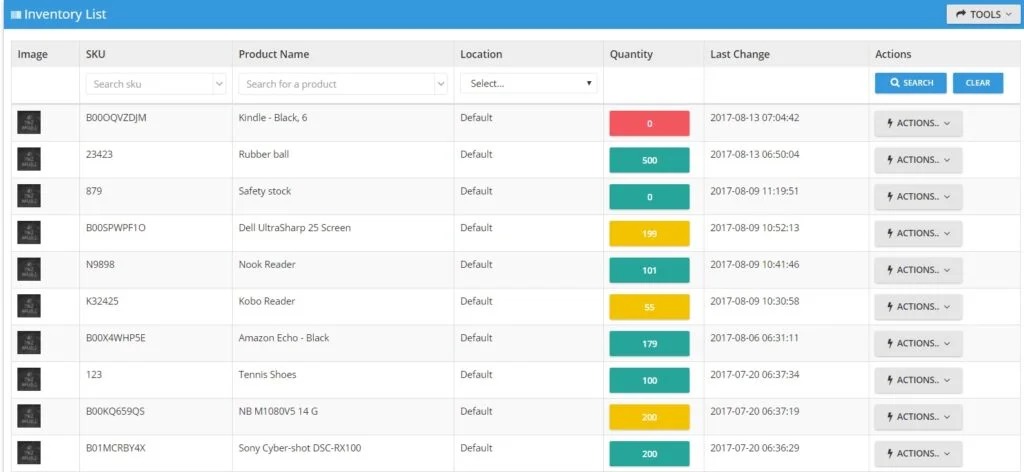
****

Fig-8.10: Inventory list

**8.11 Low Stock Notification**

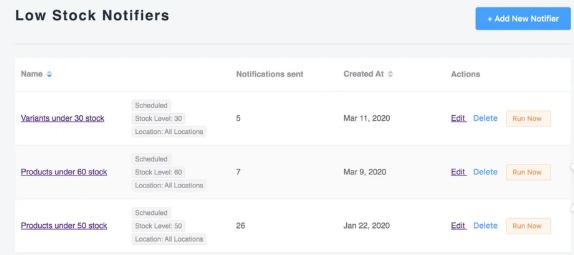
****

Fig-8.11: Low stock notification

**8.12 Empty shelf detection**

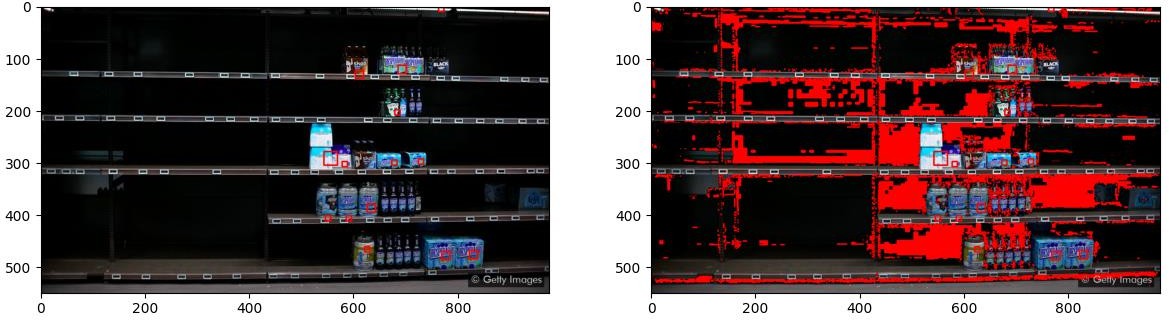
****

Fig-8.12: Empty shelf detection

# CHAPTER -09

**CONCLUSION & FUTURE ENHANCEMENT**

**9.1 CONCLUSION**

This work introduces an efficient, privacy-conscious inventory management solution that detects empty shelf regions, monitors stock levels, and automates replenishment in retail environments. Unlike traditional methods that focus solely on identifying missing items, this system employs a pre-trained Faster R-CNN model to accurately detect empty shelves with an achieved efficiency of 99%. Additionally, a hybrid OCR-CNN model is integrated to recognize item labels and quantities, ensuring precise stock tracking. By reducing manual effort and addressing privacy concerns, this approach offers a scalable and reliable solution for modern retail stores.

**9.2 FUTURE ENHANCEMENT**

* The system further enhances automation by connecting to a cloud-based inventory management platform, enabling real-time stock monitoring and generating actionable restocking recommendations.
* Future improvements can optimize detection algorithms and enhance real-time processing.
* The approach can be extended to various retail environments for broader applicability.

**CHAPTER-10**

**REFERENCES**

1. Higa K, Iwamoto K. “Robust Shelf Monitoring Using Supervised Learning for Improving On-Shelf Availability in Retail Stores”. Sensors (Basel). 2019 Jun 17;19(12):2722. doi: 10.3390/s19122722. PMID: 31213015; PMCID: PMC6631981.
2. Bottani, E.; Bertolini, M.; Rizzi, A.; Romagnoli, G. Monitoring on- shelf availability, out-of-stock and product freshness through RFID in the fresh food supply chain. Int. J. RF Technol. Res. Appl. 2017, 8, 33– 55.
3. Michael, K.; McCathie, L. The Pros and Cons of RFID in Supply Chain Management. In Proceedings of the 4th Annual 4th International Conference on Mobile Business, ICMB,

Sydney, NSW, Australia, 11– 13 July 2015; pp. 623–629

1. Moorthy, R.; Behera, S.; Verma, S.; Bhargave, S.; Ramanathan, P. Applying image processing for detecting on-shelf availability and product positioning in retail stores. In Proceedings of the ACM International Conference Proceeding Ser., Kochi, India, 10–13 August 2015; pp. 451–457.
2. Higa, K.; Iwamoto, K. Robust estimation of product amount on store shelves from a surveillance camera for improving on-shelf availability. In Proceedings of the IST 2018—IEEE International Conference Imaging Systems and Techniques Proceeding, Kraków, Poland, 16– 18 October 2018; pp. 1–6.
3. Ramiz Yilmazer and Derya Birant, “Shelf Auditing Based on Image Classification”,Sensors2021,21(2),327;https://doi.org/10.3390/s21027
4. Zhu, X. Semi-Supervised Learning, Encyclopedia of Machine Learning and Data Mining; Springer: Berlin/Heidelberg, Germany, 2017; Volume 3, ISBN 9781489976871.
5. Satapathy, R.; Prahlad, S.; Kaulgud, V. Smart Shelfie-Internet of shelves: For higher on-shelf availability. In Proceedings of the 2015 IEEE Region 10 Symposium TENSYMP, Ahmedabad, India, 13–15 May 2015; pp. 70–73.

[9] Kejriwal, N.; Garg, S.; Kumar, S. Product counting using images with application to robot-based retail stock assessment. In Proceedings of the IEEE Conference on Technologies for Practical Robot Applications, Woburn, MA, USA, 11–12 May 2015; pp. 1–6.

**CHAPTER-11**

**PAPER PUBLICATION CERTIFICATIONS**









