Major Project [4th Semester]

(DealDepot – Inventory Management System)

Project Report

Submitted by:

Project Group – 1 (COE-UA 23-26 Batch)

Under Guidance of:

Ms. Jahanvi (Assistant Professor, Department of Computer Engineering – SFC)

In partial fulfillment of the requirements for the award of the diploma in

COMPUTER ENGINEERING

Seth Jai Parkash Polytechnic

Address: Damla-135001 (Distt. Yamuna Nagar) Haryana

Website: https://www.sjpdamla.ac.in/

May, 2025

Part A:
FOUNDATION

Certificate from Project Mentor

This is to certify that the project titled "DealDepot" has been successfully carried out by Adarsh Dhiman (Roll No. 230150800102), serving as the Team Leader, along with his project team, bonafide students of the Diploma IV Semester, Department of Computer Engineering - SFC, Seth Jai Parkash Polytechnic, Damla.

The project work presented here is a genuine record of the research, development, and implementation efforts undertaken by the team during the fourth semester as a partial fulfillment of the requirements for the award of the Diploma in Computer Engineering. The project was conducted under my guidance and reflects the team's technical proficiency, problem-solving capabilities, and commitment to excellence.

Throughout the project, the team demonstrated commendable coordination, innovative thinking, and professional approach towards accomplishing the project objectives. Their work showcases a solid understanding of core computer engineering principles and their effective application to real-world scenarios.

I hereby affirm that the project meets the academic and professional standards expected at this level. I extend my best wishes to the Team for continued success and outstanding achievements in their future careers.

Dated: _____

Ms. Jahanvi
(Project Mentor)

Assistant Professor
Department of Computer Engineering - SFC,
Seth Jai Parkash Polytechnic, Damla.

Declaration

I, Adarsh Dhiman, Bearing Roll No. 230150800102, a student of Diploma VIth Semester, Department of Computer Engineering - SFC, Seth Jai Parkash Polytechnic, Damla, Serving as Team Leader of my Project Team.

Hereby declare that the project entitled "DealDepot", submitted to Seth Jai Parkash Polytechnic, Damla, is a genuine and original work completed by me and my team under the esteemed guidance of Ms. Jahanvi.

This project has been carried out as a part of the curriculum requirements for the partial fulfillment of the award of the Diploma in Computer Engineering. To the best of my knowledge and belief, this work is original, not copied from any source, and has been developed with sincere dedication and technical understanding.

Dated: _____

Adarsh Dhiman
(Team Leader)

Roll No: 230150800102
Class: COE – UA (VI Semester)
Department of Computer Engineering - SFC
Seth Jai Parkash Polytechnic, Damla

Acknowledgement

We, the project team of "DealDepot", wish to express our sincere gratitude to our project mentor, Ms. Jahanvi, for her invaluable guidance, encouragement, and continuous support throughout the development of this project. Her insightful feedback and mentorship were instrumental in shaping our ideas into a successful outcome.

We are also deeply thankful to the Head of Department and the entire faculty members of the Department of Computer Engineering - SFC, Seth Jai Parkash Polytechnic, Damla, for their constant support, motivation, and for providing a conducive environment for learning and growth.

We extend our heartfelt appreciation to our team members for their excellent coordination, patience, understanding, and dedication, which made this collaborative effort smooth and successful.

We would also like to acknowledge our classmates for their active involvement in testing the project, providing valuable feedback, and assisting us with external modules and resources, which greatly enhanced the quality of our work.

Lastly, we are grateful to Seth Jai Parkash Polytechnic, Damla for offering us this opportunity to undertake a project that helped us apply our learning practically and sharpen our technical and professional skills.

(Adarsh Dhiman and Team)
Roll No.: 230150800102
Class: COE – UA (VI Semester)
Department of Computer Engineering - SFC
Seth Jai Parkash Polytechnic, Damla

Abstract

The project titled "DealDepot" is designed to provide a complete, efficient, and user-friendly solution for small and medium-sized enterprises (SMEs) to manage store operations. It focuses on streamlining sales, inventory management, employee attendance, and billing, all within a lightweight FastAPI and web-based system using HTML, CSS, JavaScript, Bootstrap, and CSV for offline data handling.

The system is split into two major panels: an Admin Panel for management and reporting, and an Employee Panel for day-to-day sales and attendance punching. Employees can easily mark attendance, manage product sales via a smart POS screen, handle customer billing, and work within a responsive and intuitive interface. Admins have access to advanced features like employee management, sales reporting, and inventory control.

This project was successfully developed and implemented by a dedicated team:

- Adarsh Dhiman – 230150800102 (Team Leader & Backend Developer) efficiently handled the core FastAPI development, logic building, and backend connectivity.
- Arsh Chaudhary – 230150800113 (Graphics and JavaScript Developer) worked on designing dynamic frontend interactions, cart functionalities, and API integrations.
- Bhumi - 230150800116 (HTML Developer) focused on building the HTML structure of various pages like login, dashboard, and POS.
- Abhinav Kamboj - 230150800101 (Styling Expert) ensured a professional and responsive look across devices using CSS and Bootstrap.
- Himanshu – 230150800122 (Database Developer) managed the CSV data structures, ensuring clean storage of attendance, sales, products, and users.

With a focus on simplicity, performance, and scalability, DealDepot stands out as an innovative, low-cost solution for small businesses aiming to modernize their operations without heavy infrastructure requirements.

Table of Contents

Part B:

CORE

Chapter 1: Introduction

1.1 Introduction to Project

The increasing need for efficient and accessible store management systems has made digital solutions essential for small and medium-sized enterprises (SMEs). Traditional methods of maintaining sales records, managing stock, and tracking employee attendance are time-consuming and error-prone. Many existing solutions are either costly, overly complex, or dependent on constant internet connectivity, which may not be suitable for all environments.

To address these challenges, DealDepot has been developed as a centralized, lightweight inventory management and point-of-sale (POS) system. It is specifically designed for SMEs that need a simple yet powerful platform to handle daily operations such as product management, employee attendance, customer billing, and sales reporting.

The system is divided into two main roles: Admin and Employee. The Admin panel allows authorized users to manage employees, add or update product inventory, and generate reports on attendance and sales. On the other hand, employees are provided with an intuitive POS screen to mark attendance, sell products, manage the cart, and generate bills for customers. This role-based access ensures secure and streamlined operations.

DealDepot is built using FastAPI for the backend and standard web technologies including HTML, CSS, JavaScript, and Bootstrap for the frontend. All data is stored in CSV files, making the system easy to deploy without the need for a full database setup. The application can be hosted on a local network using Python sockets, enabling multiple users to operate from different devices over LAN or Wi-Fi.

With its offline-friendly design, clean interface, and essential store management features, DealDepot offers a practical and cost-effective solution for small businesses looking to digitize their workflow without depending on third-party software or internet access.

DealDepot
OUR ONE-STOP SHOP FOR UNBEATABLE DEALS

1.2 Existing System

Currently, many small and medium-scale retail businesses rely on either manual processes or limited-purpose tools to manage their daily operations. Inventory management, billing, employee attendance, and report generation are often handled separately, leading to inefficiencies, errors, and data redundancy.

Some commonly used systems or methods include:

- Manual Registers / Excel Sheets:
  Used for stock updates, attendance, and sales tracking, but prone to human error and lack real-time updates.
- Basic POS Machines:
  Only support billing operations; inventory and employee management must be handled independently.
- Commercial POS Software (costly/complex):
  - Tally ERP – Expensive, accounting-focused, not designed for store-level POS operations.
  - Zoho POS – Cloud-based, requires internet access and paid subscription.
  - Lightspeed – Feature-rich but designed for large enterprises with high infrastructure needs.
- Mobile-based Apps:
  Limited control, data security issues, and not suitable for multi-employee use over LAN.
- Third-party Attendance Systems:
  Often separate from sales platforms, requiring data to be compiled manually for reports.

These systems often:

- Lack role-based access (admin vs. employee separation),
- Require constant internet connectivity,
- Have little to no customization flexibility,
- Do not support offline/local network usage,
- And fail to combine all store operations into a single platform.

This fragmented workflow creates a clear need for a cost-effective, integrated, and locally-hosted solution that is easy to use, scalable, and specifically built for small businesses — which is the aim of DealDepot.

1.3 Problem Definition

In small and medium-sized retail businesses, managing day-to-day operations like billing, inventory updates, and employee attendance is often challenging due to the lack of an integrated and affordable system. Most available solutions either focus on individual tasks or are built for larger enterprises, making them too complex, expensive, or dependent on external infrastructure like cloud hosting or internet connectivity.

The absence of a unified platform forces store managers to rely on multiple tools or manual methods, which results in:

- Delays in data recording and retrieval,
- Increased chances of human error,
- Difficulty in monitoring employee attendance and sales,
- No real-time synchronization across users,
- Poor visibility into overall business performance.

Additionally, commonly used POS systems do not offer:

- Offline or LAN-based access,
- Custom role separation (admin vs. employee),
- Simple attendance tracking,
- Easy report generation without database integration.

For small retail setups, there is a need for a system that is:

- Simple to use and deploy,
- Cost-effective and free from subscription dependencies,
- Capable of handling sales, inventory, attendance, and reporting in one place,
- Functional even without internet access.

DealDepot aims to solve these issues by offering a lightweight, LAN-hosted, FastAPI-based solution that fulfills all essential store management needs with minimal hardware or software requirements.

1.4 Proposed System

To address the challenges faced by small and medium retail stores, the proposed solution — DealDepot — is a centralized, web-based inventory and sales management system built with simplicity, flexibility, and offline accessibility in mind. It integrates all major store functions such as product management, employee attendance, sales processing, and report generation into a single platform that runs on a local area network (LAN), making it ideal for environments without stable internet connectivity.

The system is divided into two role-based panels:

- Admin Panel: Allows administrators to manage products, monitor sales records, track employee attendance, and add or remove employee details. Admins can also generate sales and attendance reports in CSV or PDF format, filtered by month or employee.
- Employee Panel: Employees can log in, mark their attendance with a single click (ensuring no duplication based on date and emp_code), and operate a full-fledged POS interface. The POS includes category-based product filtering, a dynamic cart, quantity adjustments, and an integrated checkout system with customer detail capture and payment gateway redirection.

Key features of DealDepot include:

- Role-based login system with user and admin access.
- POS interface with real-time cart management.
- CSV-based data handling for ease of deployment and portability.
- Attendance marking with validation for each day.
- Monthly and employee-wise report generation using jsPDF.
- Category filtering and live product loading via JavaScript.
- LAN-based hosting using Python sockets, eliminating reliance on external servers.

The system uses FastAPI as the backend framework due to its performance and modular structure, while the frontend is built using HTML, CSS, JavaScript, and Bootstrap for responsive design. All data is stored in CSV format to keep the system lightweight and database-free, ideal for local deployments in retail environments.

DealDepot thus offers a low-cost, customizable, and efficient solution for stores looking to digitize their workflow without the complexity or cost of enterprise-level systems.

1.5 Hardware and Software Requirements

The DealDepot system is designed to operate efficiently within a local network setup, where the application is hosted on an admin (server) system and accessed by employees (clients) through web browsers. The hardware and software requirements for both ends are minimal and economical, making the system suitable for small and medium-scale retail stores.

For the admin side, a moderately configured desktop or laptop is required to host the FastAPI-based backend. The system should have at least an Intel i3 processor, 4 GB of RAM, and 250 GB of storage to run the server and manage CSV-based data files. It must be connected to a LAN network, either through Ethernet or Wi-Fi, which will serve the client devices. On the software side, the admin system must run a modern operating system such as Windows 10/11 or a Linux distribution. Python 3.10 or later is used along with FastAPI, Uvicorn, and libraries like pandas for data handling. Code editors such as Visual Studio Code or PyCharm can be used for backend development. A browser like Chrome or Firefox is needed for interface access and testing.

On the employee or client side, the requirements are significantly lighter. Any system capable of running a modern browser (such as Chrome, Firefox, or Edge) is sufficient. This includes desktops, laptops, or tablets connected to the same Wi-Fi or LAN as the admin system. No additional installations are needed on the client side since the application is browser-based. The frontend is built using HTML, CSS, Bootstrap, and JavaScript, and integrates external libraries like jsPDF for bill generation and QRCode.js for potential future features like digital receipts or item scanning.

This distributed structure ensures that the admin has full control over the data and system operations while employees experience a lightweight and user-friendly interface. It reduces the need for expensive infrastructure, making DealDepot a practical solution for offline, LAN-based inventory and sales management.

1.6 Back-end Tools

The backend of the DealDepot system is developed using FastAPI, a modern, high-performance web framework for building APIs with Python. FastAPI was chosen for its speed, simplicity, and automatic generation of interactive documentation using Swagger UI and ReDoc, which greatly aids in development and testing. The backend handles all the core functionalities of the system such as employee login, attendance management, product listing and categorization, sales tracking, and customer detail handling.

Data is stored using lightweight CSV files instead of complex databases, making the system easier to manage, especially for small and medium businesses without dedicated IT support. FastAPI reads and writes to these CSV files using the pandas library, which enables efficient handling and manipulation of tabular data. This structure allows for faster development and avoids the overhead of setting up and managing a full-scale database.

The backend also manages the logic for validating attendance (e.g., ensuring one punch per day), computing the bill total, generating sales reports, and updating the inventory. Uvicorn is used as the ASGI server to run FastAPI locally on the admin machine, accessible over LAN.

In summary, FastAPI combined with Python libraries like pandas provides a fast, flexible, and scalable backend environment that supports DealDepot's goals of being lightweight yet functionally complete. It also leaves room for future integration with databases or external services if the system needs to scale.

1.7 Front-end Tools

The front-end of the DealDepot system is crafted using standard HTML, CSS, and JavaScript, enhanced with the Bootstrap framework to ensure a clean, responsive, and mobile-friendly user interface. The use of Bootstrap provides a consistent visual structure and simplifies the layout of components such as navigation tabs, product cards, modals, and forms. It also ensures that the interface remains intuitive for both employees and administrators across various screen sizes.

JavaScript handles dynamic elements such as category filtering, cart management, and form validations. Libraries like jsPDF and qrcode.js are integrated for generating printable PDF bills and embedding scannable QR codes, offering convenience to both sellers and customers. These features make the POS (Point of Sale) experience seamless and tech-enabled.

The front-end communicates with the FastAPI backend using AJAX fetch calls, allowing real-time interaction without full-page reloads. Employee details are stored in localStorage to persist session data like employee code during the billing and attendance process. Pop-ups and confirmation modals enhance interactivity, while clean alert messages keep the user informed during each operation.

Overall, the front-end stack is lightweight yet powerful, focusing on user experience and functionality. It ensures that employees with minimal technical expertise can operate the system efficiently, and that administrative tasks like inventory management and sales tracking are handled smoothly.

Chapter 2: Software Requirements Specifications

DealDepot is a centralized inventory management and POS system tailored for malls, aiming to streamline the interaction between mall owners (admins) and their employees (billing staff). The software's purpose is to simplify inventory tracking, automate attendance, and enable seamless billing operations in a local area network environment using Python sockets for hosting. The system allows administrators to manage products, categorize them, generate reports, and monitor employee performance, while employees can log in, punch attendance, manage the point-of-sale interface, and generate customer bills with jsPDF and QR code integration. Designed for use within malls where the main system runs over LAN (typically via a router-connected main PC and Wi-Fi-connected employee devices), the application's core features include employee authentication, single-click attendance, category-based product selection, cart and checkout functionality, and automated PDF billing with customer data capture. All actions are logged into CSV-based records (such as sales.csv and attendance.csv) ensuring lightweight and portable data storage. The system supports a basic but reliable front end powered by Bootstrap, with JavaScript handling interactivity and FastAPI serving the backend logic. Although it is hosted locally, internet access is needed for libraries like jsPDF and QRCode.js. Non-functional requirements include high usability with an intuitive interface, support for 10–20 concurrent users, maintainability through editable CSV files, and basic authentication for security. While scalable cloud-based infrastructure is not implemented at this stage, DealDepot is built to address immediate and practical operational gaps in current small to mid-scale mall systems, especially those relying on manual or semi-digital billing processes. This makes it ideal for establishments that want efficient, LAN-based, browser-accessible inventory and billing solutions without the complexity of large-scale ERP systems.

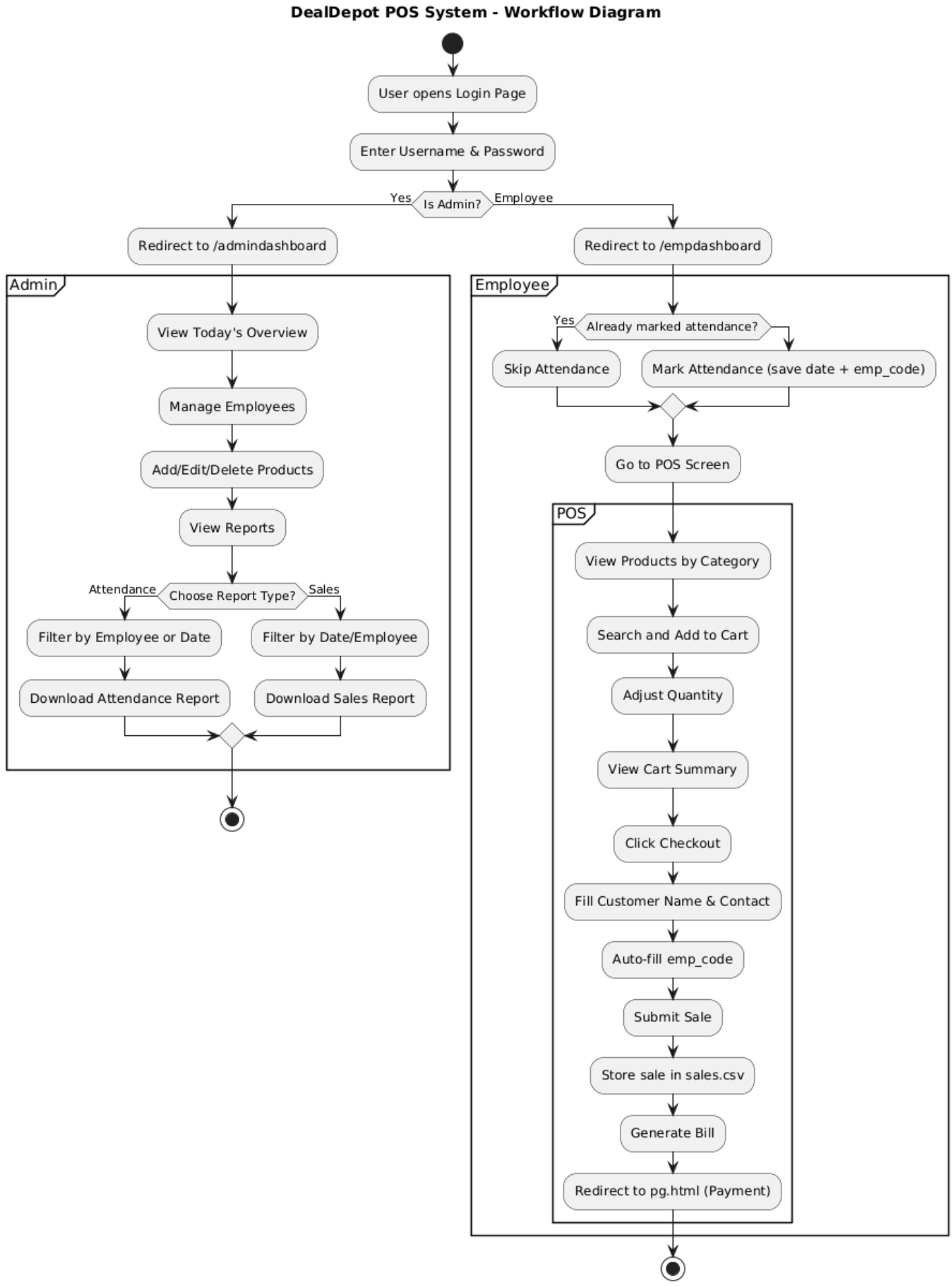
Chapter 3 - Project Design

3.1 Project Workflow

The workflow of the DealDepot system is designed to support efficient communication and task execution between two primary user roles: the Admin (mall owner or manager) and the Employee (billing counter staff). When an employee accesses the system, the process begins with authentication. Upon successful login, the employee is redirected to a personalized dashboard. The first interaction typically involves punching attendance, which the system records for the current date, avoiding duplicates for the same day. Once the attendance is marked, the employee proceeds to the Point-of-Sale (POS) module. Here, a categorized product interface is loaded dynamically using data fetched from the backend via the /api/products route. Employees can add items to the cart, adjust quantities, and when ready, initiate the checkout process. During checkout, a form captures customer name and contact number, and the system automatically fetches the employee's identity and calculates the total bill amount. Upon confirming the order, the system processes the payment, stores the billing data in sales.csv, and generates a PDF invoice using jsPDF and QRCode.js.

On the Admin side, the workflow allows for product management, including the addition of new products, modification of inventory details, and categorization for effective filtering. The admin can also monitor employee attendance, sales reports, and system usage via CSV logs. The admin interface includes utilities to view daily/weekly/monthly performance, making the system ideal for internal analysis. The backend, built with FastAPI, communicates through RESTful endpoints and serves JSON responses to the frontend, which is styled using Bootstrap and powered by JavaScript for dynamic interactions. The application, though hosted over a local area network (LAN) using Python sockets, depends on an active internet connection for client-side resources like jsPDF and QRCode libraries. This local deployment architecture ensures data privacy, faster internal performance, and reduced dependency on external servers—ideal for malls that prefer an internal, secure solution over cloud-hosted platforms.

3.2 Flowchart



3.3 Database Structure

- Employee Attendance

date	emp_code
28/01/2025	emp001
28/01/2025	emp002
28/01/2025	emp003
28/01/2025	emp004
29/01/2025	emp001

- Employee Details

emp_code	name	doj
emp001	Arsh	15/06/2023
emp002	Bhumi	30/01/2025
emp003	Abhinav	30/01/2025
emp004	Himanshu	31/01/2025

- Products

product_name	image_address	product_category	product_price	product_quantity
Aashirvaad Whole Wheat Atta	https://m.media-amazon.com/images/I/91REmKyE84L._SL1500_.jpg	Groceries	250	5000g
Daawat Basmati Rice	https://www.daawat.com/cbss-html/img/sajakaar_biryani.png	Groceries	180	1000g
Tata Salt	https://m.media-amazon.com/images/I/614mm2hYHyL._AC_UF1000,1000_QL80_.jpg	Groceries	20	1000g
Fortune Sunflower Oil	https://m.media-amazon.com/images/I/71AgCOCoKYL.jpg	Groceries	120	1000ml

- Sales

date	emp_code	bill_no	total_cost
30/01/2025	emp001	1001	500
30/01/2025	emp002	1002	700
06/02/2025	emp003	1003	200
10/04/2025	emp003	1004	130
10/04/2025	emp001	1005	355

- Users

Username	password
admin	admin@123
emp001	emp001@123

3.4 Folder Structure

DealDepot - Inventory Management System/

```
├── main.py
├── data/
│   ├── attendance.csv
│   ├── employeedetails.csv
│   ├── sales.csv
│   ├── users.csv
│   └── products.csv
├── static/
│   ├── bootstrap/
│   ├── css/
│   ├── js/
│   └── images/
├── templates/
│   ├── admin.html
│   ├── index.html
│   ├── 404.html
│   ├── 500.html
│   ├── emp.html
│   ├── pos.html
│   └── pg.html
└── documents/
    ├── README.md
    ├── Project Synposis.pdf
    └── Project Report.pdf
```

Chapter 4: Development

Source Code (main.py)

```
from fastapi import FastAPI, Form, Request, HTTPException
from fastapi.responses import HTMLResponse, RedirectResponse, JSONResponse
from fastapi.staticfiles import StaticFiles
from fastapi.templating import Jinja2Templates
from fastapi.middleware.cors import CORSMiddleware
import os
import pandas as pd
from datetime import datetime
from datetime import date
from pydantic import BaseModel
import socket
import uvicorn

app = FastAPI()
templates = Jinja2Templates(directory="templates")
app.mount("/static", StaticFiles(directory="static"), name="static")
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)
EMPLOYEE_CSV = "data/employeedetails.csv"
USER_CSV = "data/users.csv"
SALES_CSV = "data/sales.csv"
ATTENDANCE_CSV = "data/attendance.csv"
PRODUCTS_CSV = "data/products.csv"
if not os.path.exists("data"):
    os.makedirs("data")
for file_path, columns in {
    EMPLOYEE_CSV: ["emp_code", "name", "doj"],
    USER_CSV: ["username", "password"],
    SALES_CSV: ["date", "total_cost"],
    ATTENDANCE_CSV: ["emp_code", "date"],
}.items():
    if not os.path.exists(file_path):
        pd.DataFrame(columns=columns).to_csv(file_path, index=False)
@app.get("/", response_class=HTMLResponse)
async def show_login_page(request: Request):
    return templates.TemplateResponse("index.html", {"request": request, "error": None})
@app.post("/login")
async def login_user(
    request: Request, username: str = Form(...), password: str = Form(...)
):
    users = pd.read_csv(USER_CSV)
    if any((users["username"] == username) & (users["password"] == password)):
```

```
        if username.lower() == "admin":
            return RedirectResponse(url="/admindashboard", status_code=303)
        else:
            return RedirectResponse(url="/empdashboard", status_code=303)
    return templates.TemplateResponse(
        "index.html", {"request": request, "error": "Invalid username or password!"}
    )
@app.get("/admindashboard", response_class=HTMLResponse)
async def show_admin_dashboard(request: Request):
    return templates.TemplateResponse("admindashboard.html", {"request": request})
@app.get("/empdashboard", response_class=HTMLResponse)
async def show_employee_dashboard(request: Request):
    return templates.TemplateResponse("empdashboard.html", {"request": request})
@app.get("/pos", response_class=HTMLResponse)
async def show_employee_dashboard(request: Request):
    return templates.TemplateResponse("pos.html", {"request": request})
@app.get("/pg", response_class=HTMLResponse)
async def show_employee_dashboard(request: Request):
    return templates.TemplateResponse("pg.html", {"request": request})
@app.exception_handler(404)
async def not_found_handler(request: Request, exc: HTTPException):
    return templates.TemplateResponse("404.html", {"request": request}, status_code=404)
@app.exception_handler(500)
async def server_error_handler(request: Request, exc: HTTPException):
    return templates.TemplateResponse("500.html", {"request": request}, status_code=500)
@app.get("/api/employees")
async def get_employees():
    df = pd.read_csv(EMPLOYEE_CSV)
    return df.to_dict(orient="records")
@app.post("/api/employees")
async def add_employee(
    emp_code: str = Form(...), name: str = Form(...), doj: str = Form(...)
):
    emp_df = pd.read_csv(EMPLOYEE_CSV)
    if emp_code in emp_df["emp_code"].astype(str).values:
        return JSONResponse(
            content={"message": "Employee Code already exists"}, status_code=400
        )
    emp_df = emp_df._append(
        {"emp_code": emp_code, "name": name, "doj": doj}, ignore_index=True
    )
    emp_df.to_csv(EMPLOYEE_CSV, index=False)
    user_df = pd.read_csv(USER_CSV)
    if emp_code not in user_df["username"].astype(str).values:
        new_user = pd.DataFrame([{"username": emp_code, "password": f"{emp_code}@123"}])
        user_df = pd.concat([user_df, new_user], ignore_index=True)
        user_df.to_csv(USER_CSV, index=False)
    return JSONResponse(
        content={"message": "Employee added successfully"}, status_code=201
    )
@app.delete("/api/employees/{emp_code}")
async def delete_employee(emp_code: str):
```

```
    df = pd.read_csv(EMPLOYEE_CSV)
    df = df[df["emp_code"] != emp_code]
    df.to_csv(EMPLOYEE_CSV, index=False)
    return {"message": "Employee deleted successfully"}
@app.get("/api/today")
def fetch_today_data():
    today_str = date.today().strftime("%Y-%m-%d")
    try:
        sales_df = pd.read_csv(SALES_CSV)
        sales_df["date"] = sales_df["date"].astype(str)
        today_sales = sales_df[sales_df["date"] == today_str]
        total_sales = today_sales["total_cost"].sum() if not today_sales.empty else 0
    except FileNotFoundError:
        total_sales = 0
    try:
        attendance_df = pd.read_csv(ATTENDANCE_CSV)
        attendance_df["date"] = attendance_df["date"].astype(str)
        total_attendance = attendance_df[attendance_df["date"] == today_str].shape[0]
    except FileNotFoundError:
        total_attendance = 0
    return {
        "date": today_str,
        "total_sales": int(total_sales),
        "total_attendance": total_attendance,
    }
class Product(BaseModel):
    product_name: str
    image_address: str
    product_category: str
    product_price: float
    product_quantity: int
@app.get("/api/products")
def get_products():
    try:
        df = pd.read_csv(PRODUCTS_CSV)
        if df.empty:
            return {
                "product": [],
                "img_add": [],
                "category": [],
                "price": [],
                "quantity": [],
            }
        return {
            "product": df["product_name"].tolist(),
            "img_add": df["image_address"].tolist(),
            "category": df["product_category"].tolist(),
            "price": df["product_price"].tolist(),
            "quantity": df["product_quantity"].tolist(),
        }
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))
```

```
@app.post("/api/add_product")
def add_product(product: Product):
    try:
        new_product = pd.DataFrame([product.dict()])
        new_product.to_csv(
            PRODUCTS_CSV, mode="a", header=not os.path.exists(PRODUCTS_CSV), index=False
        )
        return {"success": True, "message": "Product added successfully!"}
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))
@app.delete("/api/products/{product_name}")
async def delete_product(product_name: str):
    df = pd.read_csv(PRODUCTS_CSV)
    if product_name not in df["product_name"].values:
        return {"message": "Product not found"}
    df = df[df["product_name"] != product_name]
    df.to_csv(PRODUCTS_CSV, index=False)
    return {"message": "Product deleted successfully"}
@app.get("/api/sales")
def get_sales():
    df = pd.read_csv(SALES_CSV)
    return {
        "date": df["date"].to_list(),
        "emp_code": df["emp_code"].to_list(),
        "bill_no": df["bill_no"].to_list(),
        "total_cost": df["total_cost"].to_list(),
    }
class Sale(BaseModel):
    name: str
    amount: float
    emp_code: str
    date: str
    bill_no: int
@app.post("/api/save_sales")
async def save_sales(sale: Sale):
    if os.path.exists(SALES_CSV):
        df = pd.read_csv(SALES_CSV)
    else:
        df = pd.DataFrame(columns=["date", "emp_code", "bill_no", "total_cost"])
    new_row = {
        "date": sale.date,
        "emp_code": sale.emp_code,
        "bill_no": sale.bill_no,
        "total_cost": sale.amount,
    }
    df.loc[len(df)] = new_row
    df.to_csv(SALES_CSV, index=False)
    return {"message": "Sale saved successfully", "bill_no": sale.bill_no}
@app.get("/api/attendance")
def get_attendance():
    df = pd.read_csv(ATTENDANCE_CSV)
    return {
```

```
        "date": df["date"].to_list(),
        "emp_code": df["emp_code"].to_list(),
    }
@app.post("/mark_attendance")
async def mark_attendance(emp_code: str = Form(...)):
    today = datetime.now().strftime("%Y-%m-%d")
    if os.path.exists(ATTENDANCE_CSV):
        df = pd.read_csv(ATTENDANCE_CSV)
    else:
        df = pd.DataFrame(columns=["date", "emp_code"])
    if not df[(df["emp_code"] == emp_code) & (df["date"] == today)].empty:
        return JSONResponse(
            content={
                            "status": "fail",
                "message": f"Attendance already marked for {emp_code} on {today}.",
            }
        )
    new_row = {"date": today, "emp_code": emp_code}
    df.loc[len(df)] = new_row
    df.to_csv(ATTENDANCE_CSV, index=False)
    return JSONResponse(
        content={
            "status": "success",
            "message": f"Attendance marked successfully for {emp_code} on {today}.",
        }
    )
if __name__ == "__main__":
    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    s.connect(("8.8.8.8", 80))
    local_ip = s.getsockname()[0]
    s.close()
    port = 1607
    print(f"\n🚀 Project is Live!\n")
    print(f"🔗 Local:          http://127.0.0.1:{port}")
    print(f"🌍 Network:        http://{local_ip}:{port}\n")
    uvicorn.run("main:app", host="0.0.0.0", port=port, reload=True)
```

Source Code (admin.html)

```
<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width, initial-scale=1" />

<title>DealDepot - Admin Panel</title>

<link
```

```
rel="shortcut icon"
href="/static/images/logo favicon.png"
type="image/x-icon"
/>
<link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css"
rel="stylesheet"
/>
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"></script>
<style>
.sidebar {
width: 280px;
height: 100vh;
position: fixed;
top: 0;
left: 0;
background-color: #f8f9fa;
padding: 15px;
transition: all 0.3s ease-in-out;
}
.main-content {
margin-left: 280px;
padding: 20px;
transition: all 0.3s ease-in-out;
}
.content-section {
display: none;
}
.active-section {
display: block;
```

```
}
@media (max-width: 768px) {
.sidebar {
left: -280px;
}
.main-content {
margin-left: 0;
}
}
.sidebar.show {
left: 0;
}
.nav-link.active {
background-color: #008ab3 !important;
color: white !important;
}
#toggleBtn {
background-color: #008ab3;
border-color: #008ab3;
}
.sidebar .collapse-btn {
background-color: #008ab3;
color: white;
border: none;
margin-top: 20px;
cursor: pointer;
.card {
transition: 0.3s;
}
.card:hover {
```

```
transform: scale(1.05);
}
.employee-table {
width: 100%;
border-collapse: collapse;
margin-top: 20px;
}
.employee-table th,
.employee-table td {
border: 1px solid #ccc;
padding: 8px 12px;
text-align: left;
}
.employee-table th {
background-color: #f4f4f4;
}
.filter-section {
margin-bottom: 10px;
}
.download-buttons {
margin-top: 10px;
}
}
</style>
<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.15.4/css/all.min.css" />
</head>
<body class="bg-light">
<!-- Sidebar -->
<div class="sidebar d-flex flex-column flex-shrink-0" id="sidebar">
<a
```

```
href="#"
class="d-flex align-items-center mb-3 link-dark text-decoration-none"
>
<img
src="/static/images/logo png square.png"
alt="Logo"
class="me-2"
style="height: 150px; width: auto; margin-left: 20px"
/>
</a>
<hr />
<ul class="nav nav-pills flex-column mb-auto">
<li class="nav-item">
<a
href="#"
class="nav-link link-dark active"
onclick="showContent(event, 'home')"
>Home</a
>
</li>
<li>
<a
href="#"
class="nav-link link-dark"
onclick="showContent(event, 'dashboard')"
>Inventory Management</a
>
</li>
<li>
<a
```

```
href="#"
class="nav-link link-dark"
onclick="showContent(event, 'orders')"
>Sales Records</a
>
</li>
<li>
<a
href="#"
class="nav-link link-dark"
onclick="showContent(event, 'products')"
>Employee Attendance Records</a
>
</li>
<li>
<a
href="#"
class="nav-link link-dark"
onclick="showContent(event, 'customers')"
>Manage Employees</a
>
</li>
</ul>
<hr />
<div class="d-flex align-items-center mt-3">
<img
src="https://t3.ftcdn.net/jpg/00/65/75/68/360_F_65756860_GUZwzOKNMUU3HldFoIA44qss7ZIrCG8I.jpg"
width="40"
height="40"
class="rounded-circle me-2"
```

```
/>
<strong>Admin</strong>
</div>
<a
href="/"
class="btn btn-danger mt-2 w-100"
style="background-color: #e94a2d"
><i class="fas fa-power-off"></i> Sign Out</a

<!-- Button to collapse the sidebar -->
<button class="collapse-btn d-md-none" id="collapseSidebarBtn">
☰ Collapse Menu
</button>
</div>
<!-- Main Content -->
<div class="main-content">
<div id="home" class="content-section active-section">
<center><h2 style="color: #e94a2d">Home</h2></center>
<br />
<div
class="d-flex justify-content-center align-items-center form-container"
>
<div class="p-4 rounded shadow" style="max-width: 500px; width: 100%">
<div class="mb-3">
<p><strong>Date:</strong> <span id="today-date"></span></p>
<p id="total-sales">
Today's Sales:
<img
src="/static/images/loading-buffering.gif"
alt=""
style="height: 25px; width: auto"
```

```
/>
</p>
</div>
<div class="mb3">
<p id="total-attendance">
Total Attendance:
<img
src="/static/images/loading-buffering.gif"
alt=""
style="height: 25px; width: auto"
/>
</p>
</div>
</div>
</div>
</div>
<script src="script.js"></script>
<div id="dashboard" class="content-section">
<center><h2 style="color: #e94a2d;">Inventory Management</h2></center>
<br>
<button class="btn mt-3" data-bs-toggle="modal" data-bs-target="#addProductModal" style="background-color: #e94a2d; color: #fff;">Add More Products</button>

<!-- Tabs for categories -->
<ul class="nav nav-tabs" id="categoryTabs">
<li class="nav-item"><a class="nav-link active" href="#" data-category="All"> All </a></li>
<li class="nav-item"><a class="nav-link" href="#" data-category=" Groceries "> Groceries </a></li>
<li class="nav-item"><a class="nav-link" href="#" data-category=" Household Essentials "> Household Essentials </a></li>
<li class="nav-item"><a class="nav-link" href="#" data-category=" Personal Care & Hygiene "> Personal Care & Hygiene </a></li>
<li class="nav-item"><a class="nav-link" href="#" data-category=" Baby & Kids "> Baby & Kids Products </a></li>
```

```
<li class="nav-item"><a class="nav-link" href="#" data-category=" Health & Wellness "> Health & Wellness </a></li>

<li class="nav-item"><a class="nav-link" href="#" data-category=" Stationery & Books "> Stationery & Books </a></li>

<li class="nav-item"><a class="nav-link" href="#" data-category=" Electronics & Accessories "> Electronics & Accessories </a></li>

<li class="nav-item"><a class="nav-link" href="#" data-category=" Clothing & Footwear "> Clothing & Footwear </a></li>

<li class="nav-item"><a class="nav-link" href="#" data-category=" Beauty & Cosmetics "> Beauty & Cosmetics </a></li>

<li class="nav-item"><a class="nav-link" href="#" data-category=" Home & Kitchen Essentials "> Home & Kitchen Essentials </a></li>

<li class="nav-item"><a class="nav-link" href="#" data-category=" Automotive Accessories "> Automotive Accessories </a></li>

<li class="nav-item"><a class="nav-link" href="#" data-category=" Pet Supplies "> Pet Supplies </a></li>

<li class="nav-item"><a class="nav-link" href="#" data-category=" Sports & Fitness "> Sports & Fitness </a></li>

<li class="nav-item"><a class="nav-link" href="#" data-category=" Home Improvement & Hardware "> Home Improvement & Hardware </a></li>

<li class="nav-item"><a class="nav-link" href="#" data-category=" Travel & Luggage "> Travel & Luggage </a></li>

<li class="nav-item"><a class="nav-link" href="#" data-category=" Luxury & Fashion Accessories "> Luxury & Fashion Accessories </a></li>

<li class="nav-item"><a class="nav-link" href="#" data-category=" Toys & Games "> Toys & Games </a></li>

<li class="nav-item"><a class="nav-link" href="#" data-category=" Party & Festive Supplies "> Party & Festive Supplies </a></li>

<li class="nav-item"><a class="nav-link" href="#" data-category=" Gardening & Outdoor Supplies "> Gardening & Outdoor Supplies </a></li>

<li class="nav-item"><a class="nav-link" href="#" data-category=" Industrial & Office Supplies "> Industrial & Office Supplies </a></li>

<li class="nav-item"><a class="nav-link" href="#" data-category=" Special Sections (Seasonal & Occasional) "> Special Sections (Seasonal & Occasional) </a></li>

<li class="nav-item"><a class="nav-link" href="#" data-category=" Books & Media "> Books & Media </a></li>

</ul>


<!-- Product Cards -->

<div id="product-container" class="row mt-3"></div>
```

```
<!-- Add Product Button -->

</div>

<script>

document.getElementById("categoryDropdown").addEventListener("change", function() {

let selectedCategory = this.value;

filterProducts(selectedCategory);

});

function filterProducts(category) {

let products = document.querySelectorAll("#product-container .product");

products.forEach(product => {

if (category === "All" || product.dataset.category === category) {

product.style.display = "block";

} else {

product.style.display = "none";

}

});

}

</script>

<!-- Bootstrap Modal for Adding Products -->

<div class="modal fade" id="addProductModal" tabindex="-1" aria-labelledby="addProductModalLabel" aria-
hidden="true">

<div class="modal-dialog">

<div class="modal-content">

<div class="modal-header">

<h5 class="modal-title" id="addProductModalLabel">Add New Product</h5>

<button type="button" class="btn-close" data-bs-dismiss="modal" aria-label="Close"></button>

</div>

<div class="modal-body">

<form id="addProductForm">

<div class="mb-3">
```

```
<label class="form-label">Product Name</label>
<input type="text" class="form-control" id="productName" required>
</div>
<div class="mb-3">
<label class="form-label">Image URL</label>
<input type="url" class="form-control" id="productImage" required>
</div>
<div class="mb-3">
<label class="form-label">Category</label>
<select class="form-control" id="productCategory" required>
<option value=" Groceries "> Groceries </option>
<option value=" Household Essentials "> Household Essentials </option>
<option value=" Personal Care & Hygiene "> Personal Care & Hygiene </option>
<option value=" Baby & Kids "> Baby & Kids Products </option>
<option value=" Health & Wellness "> Health & Wellness </option>
<option value=" Stationery & Books "> Stationery & Books </option>
<option value=" Electronics & Accessories "> Electronics & Accessories </option>
<option value=" Clothing & Footwear "> Clothing & Footwear </option>
<option value=" Beauty & Cosmetics "> Beauty & Cosmetics </option>
<option value=" Home & Kitchen Essentials "> Home & Kitchen Essentials </option>
<option value=" Automotive Accessories "> Automotive Accessories </option>
<option value=" Pet Supplies "> Pet Supplies </option>
<option value=" Sports & Fitness "> Sports & Fitness </option>
<option value=" Home Improvement & Hardware "> Home Improvement & Hardware </option>
<option value=" Travel & Luggage "> Travel & Luggage </option>
<option value=" Luxury & Fashion Accessories "> Luxury & Fashion Accessories </option>
<option value=" Toys & Games "> Toys & Games </option>
<option value=" Party & Festive Supplies "> Party & Festive Supplies </option>
<option value=" Gardening & Outdoor Supplies "> Gardening & Outdoor Supplies </option>
<option value=" Industrial & Office Supplies "> Industrial & Office Supplies </option>
```

```
<option value=" Special Sections (Seasonal & Occasional) "> Special Sections (Seasonal & Occasional)
</option>

<option value=" Books & Media "> Books & Media </option>

</select>

</div>

<div class="mb-3">

<label class="form-label">Price (₹)</label>

<input type="number" class="form-control" id="productPrice" required>

</div>

<div class="mb-3">

<label class="form-label">Stock Quantity</label>

<input type="number" class="form-control" id="productStock" required>

</div>

<button type="submit" class="btn btn-primary">Add Product</button>

</form>

</div>

</div>

</div>

</div>


<div id="orders" class="content-section">

<center><h2 style="color: #e94a2d">Sales Records</h2></center>

<!-- Date Filter -->

<div class="filter-section">

<label for="filterDate">Filter by Date:</label>

<input type="date" id="filterDate" class="form-control" />

</div>

<!-- Sales Records Table (employee list style) -->

<table class="table table-bordered table-hover mt-3">

<thead class="table-dark">
```

```
<tr>
<th style="background-color: #008ab3;">Date</th>
<th style="background-color: #008ab3;">Employee Code</th>
<th style="background-color: #008ab3;">Bill Number</th>
<th style="background-color: #008ab3;">Total Cost</th>
</tr>
</thead>
<tbody id="salesTableBody">
<!-- Sales records will be dynamically inserted here -->
</tbody>
</table>

<!-- Download Report Buttons -->
<div class="download-buttons">
<button class="btn" onclick="downloadPDF()" style="background-color: #e94a2d; color: #fff;">Download Report</button>
</div>
</div>

<div id="products" class="content-section">
<center><h2 style="color: #e94a2d">Employee Attendance Records</h2></center>
<div class="filter-section">
<label for="filterEMPCode">Filter by Employee Code</label>
<input type="text" id="filterEMPCode" class="form-control" />
<label for="filterEMPDate">Filter by Date:</label>
<input type="date" id="filterEMPDate" class="form-control" />
</div>
<table class="table table-bordered table-hover mt-3">
<thead class="table-dark">
<tr>
<th style="background-color: #008ab3;">Date</th>
<th style="background-color: #008ab3;">Employee Code</th>
```

```
</tr>
</thead>
<tbody id="AttendanceTableBody">
</tbody>
</table>
<div class="download-buttons">
<button class="btn" onclick="downloadAPDF()" style="background-color: #e94a2d; color: #fff;">Download Report</button>
</div>
</div>

<div id="customers" class="content-section container mt-5">
<h2 class="text-center mb-4" style="color: #e94a2d">Manage Employees</h2>
<!-- Add / > Employee Form -->
<div class="card shadow-lg">
<div class="card-header text-white" style="background-color: #008ab3">
<h4 class="mb-0" id="formTitle">Add Employee</h4>
</div>
<div class="card-body">
<form id="employeeForm" class="row g-3">
<div class="col-md-4">
<label for="emp_code" class="form-label">Employee Code</label>
<input
type="text"
id="emp_code"
class="form-control"
required
/>
</div>
<div class="col-md-4">
<label for="name" class="form-label">Name</label>
```

```
<input type="text" id="name" class="form-control" required />
</div>
<div class="col-md-4">
<label for="doj" class="form-label">Date of Joining</label>
<input type="date" id="doj" class="form-control" required />
</div>
<div class="col-12 text-center mt-3">
<button
type="submit"
class="btn btn-success w-25"
id="submitButton"
style="background-color: #e94a2d"
>
Add Employee
</button>
<button
type="button"
class="btn btn-secondary w-25 d-none"
id="cancelEdit"
style="background-color: #e94a2d"
>
Cancel
</button>
</div>
</form>
</div>
</div>
<hr class="my-5" />
<!-- Employee Table -->
<h3 class="text-center">Employee List</h3>
```

```
<div class="table-responsive">
<table class="table table-bordered table-hover mt-3">
<thead class="table-dark">
<tr>
<th style="background-color: #008ab3;">Employee Code</th>
<th style="background-color: #008ab3;">Name</th>
<th style="background-color: #008ab3;">Date of Joining</th>
<th style="background-color: #008ab3;">Actions</th>
</tr>
</thead>
<tbody id="employeeTableBody">
<!-- Employee records will be dynamically inserted here -->
</tbody>
</table>
</div>
</div>
<!-- JavaScript -->
<script src="/static/js/admin.js"></script>
<!-- Bootstrap JS (for interactivity) -->
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"></script>
</div>
<button class="btn btn-primary d-md-none m-3" id="toggleBtn">☰ Menu</button>
<script>
function showContent(event, sectionId) {
event.preventDefault();
document
.querySelectorAll(".content-section")
.forEach((section) => section.classList.remove("active-section"));
document.getElementById(sectionId).classList.add("active-section");
document
```

```
.querySelectorAll(".nav-link")
.forEach((link) => link.classList.remove("active"));
event.target.classList.add("active");
}
document
.getElementById("toggleBtn")
.addEventListener("click", function () {
document.getElementById("sidebar").classList.toggle("show");
// Hide the collapse button when expanded
});
// Collapse button functionality
document
.getElementById("collapseSidebarBtn")
.addEventListener("click", function () {
document.getElementById("sidebar").classList.remove("show");
document.getElementById("toggleBtn").classList.remove("d-none"); // Show the main toggle button again
// Hide the collapse button
});
// 📌 Fetch data from FastAPI and update the dashboard after a delay
document.addEventListener("DOMContentLoaded", function () {
setTimeout(() => {
fetch("/api/today") // Change URL if hosted on a server
.then((response) => response.json())
.then((data) => {
document.getElementById("total-sales").innerText =
"Today's Sales: ₹" + data.total_sales;
document.getElementById("total-attendance").innerText =
"Total Attendance: " + data.total_attendance;
})
.catch((error) => console.error("Error fetching data:", error));
```

```
}, 1500); // 2000 milliseconds = 2 seconds

// 📌 Update the date dynamically in the HTML after a delay
setTimeout(() => {
let todayDate = new Date().toISOString().split("T")[0]; // Format: YYYY-MM-DD
document.getElementById("today-date").innerText = todayDate;
}, 1000); // 2000 milliseconds = 2 seconds
});

// Fetch and display the products in the inventory management section
document.addEventListener("DOMContentLoaded", function () {
loadProducts("All"); // Load all products initially
setupCategoryFilters();
setupSalesRecords();
setupAttendanceRecords();
});
// ☑ Load Products from API and Display
function loadProducts(category = "All") {
let container = document.getElementById("product-container");
container.innerHTML = ""; // Clear previous content
fetch("/api/products")
.then(response => {
if (!response.ok) throw new Error("Network response was not ok");
return response.json();
})
.then(data => {
console.log("Fetched Products:", data);
for (let i = 0; i < data.product.length; i++) {
if (category === "All" || data.category[i] === category) {
let card = document.createElement("div");
card.classList.add("col-lg-3", "col-md-4", "col-sm-6", "mb-3");
```

```
card.innerHTML = `

<div class="card shadow-sm" style="width: 100%; max-width: 220px; margin: auto;" loading="lazy">

<img src="${data.img_add[i]}" class="card-img-top" alt="${data.product[i]}" style="height: 150px; object-fit: contain;">

<div class="card-body p-2">

<h6 class="card-title mb-1">${data.product[i]}</h6>

<p class="card-text text-muted small">Category: <strong>${data.category[i]}</strong></p>

<p class="card-text text-muted small">Price: <strong>₹${data.price[i]}</strong></p>

<!--<p class="card-text text-muted small">Stock: <strong>${data.quantity[i]}</strong> left</p>-->

<div class="d-flex justify-content-between">

<!--<button class="btn btn-sm btn-primary">Edit Quantity</button>-->

<button class="btn btn-sm btn-danger" onclick="deleteProduct('${data.product[i]}')">Delete</button>

</div>

</div>

</div>

`;

container.appendChild(card);

}

}

})

.catch(error => console.error("Error fetching products:", error));

}

// Delete Employee Function

window.deleteProduct = async function (productName) {

if (!productName) {

alert("Invalid product name!");

return;

}

if (confirm(`Are you sure you want to delete ${productName}?`)) {

try {

let response = await fetch(`/api/products/${encodeURIComponent(productName)}`, {
```

```
method: "DELETE",
});
if (!response.ok) {
throw new Error("Failed to delete product");
}
let result = await response.json();
alert(result.message);
loadProducts(); // Reload products after deletion
} catch (error) {
console.error("Error deleting product:", error);
alert("An error occurred while deleting the product.");
}
}
};
// ☑ Set up Category Filtering
function setupCategoryFilters() {
let tabs = document.querySelectorAll("#categoryTabs .nav-link");
tabs.forEach(tab => {
tab.addEventListener("click", function (e) {
e.preventDefault();
tabs.forEach(t => t.classList.remove("active"));
this.classList.add("active");
loadProducts(this.getAttribute("data-category"));
});
});
}
// ☑ Handle Add Product Form Submission
document.getElementById("addProductForm").addEventListener("submit", function (event) {
event.preventDefault();
```

```
let formData = {
product_name: document.getElementById("productName").value,
image_address: document.getElementById("productImage").value,
product_category: document.getElementById("productCategory").value,
product_price: parseFloat(document.getElementById("productPrice").value),
product_quantity: parseInt(document.getElementById("productStock").value)
};
fetch("/api/add_product", {
method: "POST",
headers: { "Content-Type": "application/json" },
body: JSON.stringify(formData)
})
.then(response => response.json())
.then(data => {
alert("Product added successfully!");
loadProducts("All"); // Reload all products
document.getElementById("addProductForm").reset();
document.querySelector("#addProductModal .btn-close").click();
})
.catch(error => console.error("Error adding product:", error));
});
// Fetch and display the sales records in the table
document.addEventListener("DOMContentLoaded", function () {
fetch("/api/sales")
.then((response) => response.json())
.then((data) => {
const tableBody = document.getElementById("salesTableBody");
// Convert raw data into an array of objects for easy filtering
const salesRecords = data.date.map((date, index) => ({
date: date,
```

```
emp_code: data.emp_code[index],
bill_no: data.bill_no[index],
total_cost: data.total_cost[index],
}));
// Function to populate the table with records
function populateTable(records) {
tableBody.innerHTML = ""; // Clear previous data
records.forEach((record) => {
const row = document.createElement("tr");
row.innerHTML = `
<td>${record.date}</td>
<td>${record.emp_code}</td>
<td>${record.bill_no}</td>
<td>${record.total_cost}</td>
`;
tableBody.appendChild(row);
});
}
// Initially populate the table with all records
populateTable(salesRecords);
// Add event listener for filtering
document.getElementById("filterDate").addEventListener("input", function () {
const selectedDate = this.value;
if (selectedDate) {
// Filter records by selected date
const filteredRecords = salesRecords.filter(record => record.date === selectedDate);
populateTable(filteredRecords);
} else {
// If no date is selected, show all records
```

```
populateTable(salesRecords);
}
});
})
.catch((error) => console.error("Error fetching sales data:", error));
});

// Function to display sales records in the table
// Download PDF Report using jsPDF
function downloadPDF() {
const { jsPDF } = window.jspdf;
const doc = new jsPDF();
// Get the filtered date value
const filterDate = document.getElementById('filterDate').value;
fetch('/api/sales')
.then(response => response.json())
.then(data => {
let filteredData = {
date: [],
emp_code: [],
bill_no: [],
total_cost: []
};
// If a date is selected, filter records, otherwise use all data
data.date.forEach((date, index) => {
if (!filterDate || date === filterDate) {
filteredData.date.push(date);
filteredData.emp_code.push(data.emp_code[index]);
filteredData.bill_no.push(data.bill_no[index]);
filteredData.total_cost.push(data.total_cost[index]);
```

```
}
});
let yOffset = 20;
doc.setFont("times", "normal"); // Set font to Times New Roman
// Add logo
doc.addImage('/static/images/logo png square.png', 'PNG', 80, yOffset, 40, 40); // Center logo
yOffset += 50;
// Title
doc.setFontSize(16);
doc.text('Sales Records', 105, yOffset, { align: 'center' });
yOffset += 10;
// Table Headers (Centered)
doc.setFontSize(12);
doc.setFont("times", "bold");
doc.text('Date', 50, yOffset, { align: 'center' });
doc.text('Employee Code', 85, yOffset, { align: 'center' });
doc.text('Bill Number', 120, yOffset, { align: 'center' });
doc.text('Total Cost', 155, yOffset, { align: 'center' });
yOffset += 10;
// Reset font for content
doc.setFont("times", "normal");
// Add table data
filteredData.date.forEach((date, index) => {
doc.text(date, 50, yOffset, { align: 'center' });
doc.text(filteredData.emp_code[index], 85, yOffset, { align: 'center' });
doc.text(String(filteredData.bill_no[index]), 120, yOffset, { align: 'center' });
doc.text(String(filteredData.total_cost[index]), 155, yOffset, { align: 'center' });
yOffset += 10;
// Check if yOffset exceeds page limit and add a new page if needed
```

```
if (yOffset > 280) {
doc.addPage();
yOffset = 20;
}
});
// Save PDF
doc.save('sales_report.pdf');
})
.catch(error => console.error('Error downloading PDF:', error));
}
document.addEventListener("DOMContentLoaded", function () {
fetchAttendanceData();
});
let attendanceData = [];
function fetchAttendanceData() {
fetch("/api/attendance") // Your API endpoint
.then(response => response.json())
.then(data => {
if (Array.isArray(data)) {
attendanceData = data.map(item => ({
date: item.date,
emp_code: item.emp_code
}));
} else if (data.date && data.emp_code) {
attendanceData = data.date.map((date, index) => ({
date: date,
emp_code: data.emp_code[index]
}));
} else {
console.error("Invalid data format received from API:", data);
```

```
return;
}
populateTable(attendanceData);
})
.catch(error => console.error("Error fetching attendance data:", error));
}
function populateTable(records) {
const tableBody = document.getElementById("AttendanceTableBody");
tableBody.innerHTML = "";
records.forEach(record => {
const row = document.createElement("tr");
row.innerHTML = `
<td>${record.date}</td>
<td>${record.emp_code}</td>
`;
tableBody.appendChild(row);
});
}
function filterAttendance() {
const selectedDate = document.getElementById("filterEMPDate").value;
const selectedEmpCode = document.getElementById("filterEMPCode").value.toUpperCase();
const filteredRecords = attendanceData.filter(record => {
const dateMatch = !selectedDate || record.date === selectedDate;
const empCodeMatch = !selectedEmpCode || record.emp_code.toUpperCase() === selectedEmpCode;
return dateMatch && empCodeMatch;
});
populateTable(filteredRecords);
}
document.getElementById("filterEMPDate").addEventListener("input", filterAttendance);
document.getElementById("filterEMPCode").addEventListener("input", filterAttendance);
```

```
// Download PDF Report using jsPDF
function downloadAPDF() {
const { jsPDF } = window.jspdf;
const doc = new jsPDF();
// Get the filtered date and EMPCode values
const filterDate = document.getElementById('filterEMPDate').value;
const filterEMPCode = document.getElementById('filterEMPCode').value;
fetch('/api/attendance')
.then(response => response.json())
.then(data => {
let filteredData = {
date: [],
emp_code: [],
};
// Filter records based on both date and EMPCode
data.date.forEach((date, index) => {
// If filterDate or filterEMPCode is provided, only add matching records
if (
(!filterDate || date === filterDate) &&
(!filterEMPCode || data.emp_code[index] === filterEMPCode)
) {
filteredData.date.push(date);
filteredData.emp_code.push(data.emp_code[index]);
}
});
if (filteredData.date.length === 0) {
alert('No records found for the selected filters.');
return;
}
let yOffset = 20;
```

```
doc.setFont("times", "normal"); // Set font to Times New Roman

// Add logo

doc.addImage('/static/images/logo png square.png', 'PNG', 80, yOffset, 40, 40); // Center logo

yOffset += 50;

// Title

doc.setFontSize(16);

doc.text('Employee Attendance Records', 105, yOffset, { align: 'center' });

yOffset += 10;

// Table Headers (Centered)

doc.setFontSize(12);

doc.setFont("times", "bold");

doc.text('Date', 50, yOffset, { align: 'center' });

doc.text('Employee Code', 85, yOffset, { align: 'center' });

yOffset += 10;

// Reset font for content

doc.setFont("times", "normal");

// Add table data

filteredData.date.forEach((date, index) => {

doc.text(date, 50, yOffset, { align: 'center' });

doc.text(filteredData.emp_code[index], 85, yOffset, { align: 'center' });

yOffset += 10;

// Check if yOffset exceeds page limit and add a new page if needed

if (yOffset > 280) {

doc.addPage();

yOffset = 20;

}

});

// Save PDF

doc.save('attendance_report.pdf');
```

```
})
.catch(error => console.error('Error downloading PDF:', error));
}

// Fetch records when the page loads
window.onload = fetchSalesRecords;

</script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/jspdf/2.5.1/jspdf.umd.min.js"></script>
</body>
</html>
```

Source Code (emp.html)

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<title>DealDepot - Employee Panel</title>
<link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css"
rel="stylesheet"
/>
<link
rel="shortcut icon"
href="/static/images/logo%20favicon.png"
type="image/x-icon"
/>
<style>
body {
background-color: #f8f9fa;
```

```
font-family: Arial, sans-serif;
}
header img {
display: block;
margin: 20px auto;
max-width: 400px;
}
.welcome {
text-align: center;
font-size: 1.5rem;
font-weight: 500;
margin-bottom: 30px;
}
.dialog-box {
max-width: 500px;
margin: 0 auto;
background-color: white;
padding: 25px;
border-radius: 10px;
box-shadow: 0 0 15px rgba(0, 0, 0, 0.1);
text-align: center;
}
.btn-mark {
background-color: #e94a2d;
color: white;
}
</style>
</head>
<body>
```

```
<header>
<img src="/static/images/logo png.png" alt="Logo" />
</header>
<div class="welcome">Welcome, <span id="empcode"></span>!</div>
<div class="dialog-box">
<p>
Do you want to mark attendance for <strong id="today-date"></strong>?
</p>
<button class="btn btn-mark" onclick="markAttendance()">
Mark Attendance
</button>
<button class="btn btn-mark" onclick="window.location.href='/pos'">
Proceed to POS
</button>
</div>
<script>
// Get emp_code from localStorage
const empCode = localStorage.getItem("emp_code");
document.getElementById("empcode").textContent = empCode ?? "Unknown";
// Display today's date
const today = new Date().toISOString().split("T")[0];
document.getElementById("today-date").textContent = today;
function markAttendance() {
const empCode = localStorage.getItem("emp_code");
const formData = new FormData();
formData.append("emp_code", empCode);
fetch("/mark_attendance", {
method: "POST",
```

```
body: formData,
})
.then((res) => res.json())
.then((data) => {
alert(data.message);
})
.catch((err) => {
alert("Something went wrong!");
console.error(err);
});
}
</script>
</body>
</html>
```

Chapter 5: Tech Stack

The development of the *DealDepot: Inventory Management System* involved the use of a modern and efficient technology stack suited for rapid web development, real-time interaction, and smooth UI/UX experience. This chapter outlines and explains all the technologies, tools, frameworks, and libraries used in both frontend and backend development of the project.

5.1 Programming Language

- Python:
  The entire backend of the project is built using Python. It is chosen for its simplicity, vast ecosystem, and excellent support for asynchronous programming, which is beneficial for handling real-time requests in the inventory and billing operations.

5.2 Backend Framework

- FastAPI:
  FastAPI is a high-performance, modern web framework for building APIs with Python. It is used as the core framework for routing, server-side logic, and API documentation (using built-in /docs and /redoc endpoints). It offers asynchronous support with async/await, automatic validation with Pydantic, and automatic generation of OpenAPI schemas.

5.3 Frontend Technologies

- HTML5: Used for structuring all the pages such as login, admin dashboard, employee panel, POS screen, and reports.
- CSS3 & Bootstrap 4: Styling and responsive design are achieved using CSS and Bootstrap. Bootstrap offers pre-defined components and a grid system, which helped build a consistent and mobile-friendly UI.
- JavaScript:
  JavaScript is used for DOM manipulation, event handling, and dynamic updates on the POS screen. It also controls functions like quantity adjustments and real-time cart updates.
- QRCode.js:
  Used to generate QR codes during the billing process for fast payment or receipt encoding.
- jsPDF:
  A JavaScript library used to generate downloadable PDF bills/invoices at checkout directly from the browser.

5.4 Data Storage

- CSV Files (Comma Separated Values):
  For simplicity and ease of implementation, CSV files are used to store and manage:
    - attendance.csv: Logs employee attendance records.
    - sales.csv: Stores all billing and sales transactions.
    - users.csv: Maintains login credentials and user types.
    - products.csv: Lists available products with details.
    - employeedetails.csv: Stores employee information.

This method allows fast file read/write access without the overhead of managing a full database system in small to medium-scale environments.

5.5 Folder & File Structure

- Main File: main.py – The entry point and controller of the FastAPI application.
- /data/: Contains all the .csv files for attendance, sales, users, and products.
- /templates/: Holds HTML files for frontend rendering (index.html, admin.html, emp.html, pos.html, pg.html, 404.html, 500.html).
- /static/:
    - /css/: Custom stylesheets.
    - /bootstrap/: Bootstrap files for layout and components.
    - /javascript/: All JavaScript files including POS logic, jsPDF, and QRCode.

    - /images/: Icons, logos, and UI images.
- /documents/: Contains documentation files like ReadMe.md, Synopsis.pdf, and the final report.

5.6 Hosting & Networking

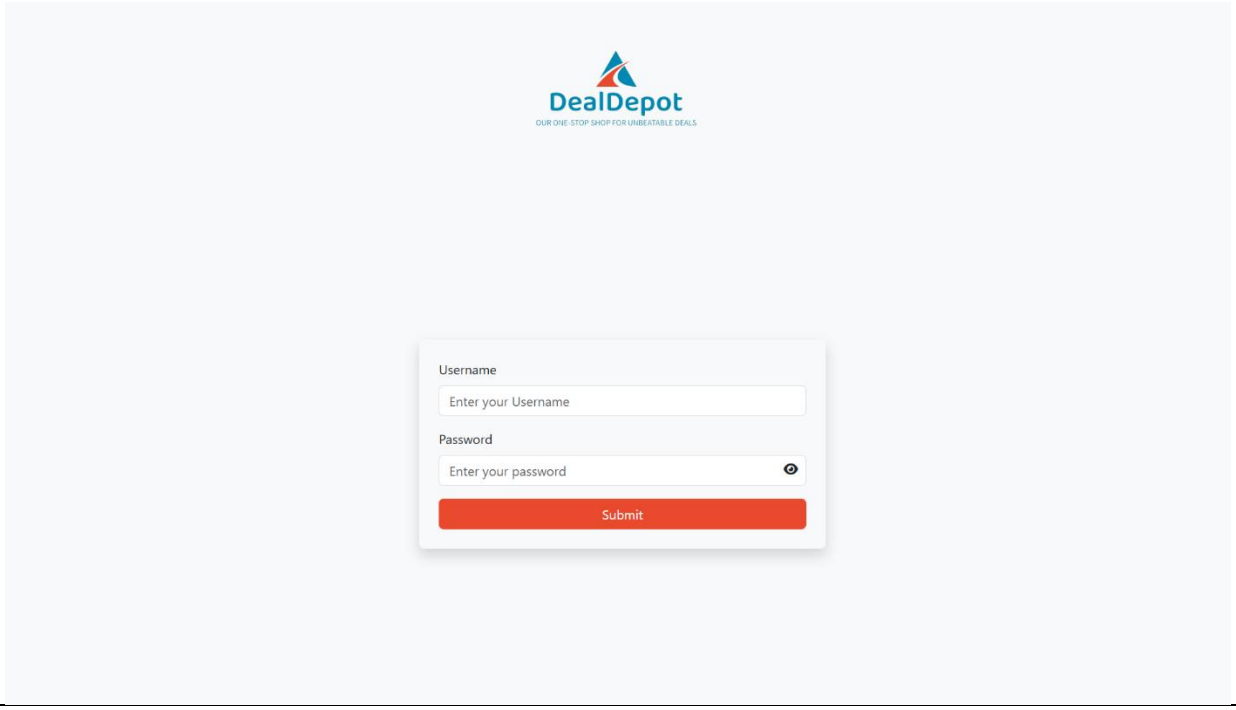
- Python Socket Module:
  The system uses the socket module to enable local area networking. The host (usually the admin system) acts as a server, and multiple clients (employee billing counters) can connect via LAN/Wi-Fi for shared access.
- Local Hosting:
  The server runs locally using uvicorn (an ASGI server compatible with FastAPI) and is accessible on other devices connected to the same network using the system's IP address.

5.7 API Documentation & Testing

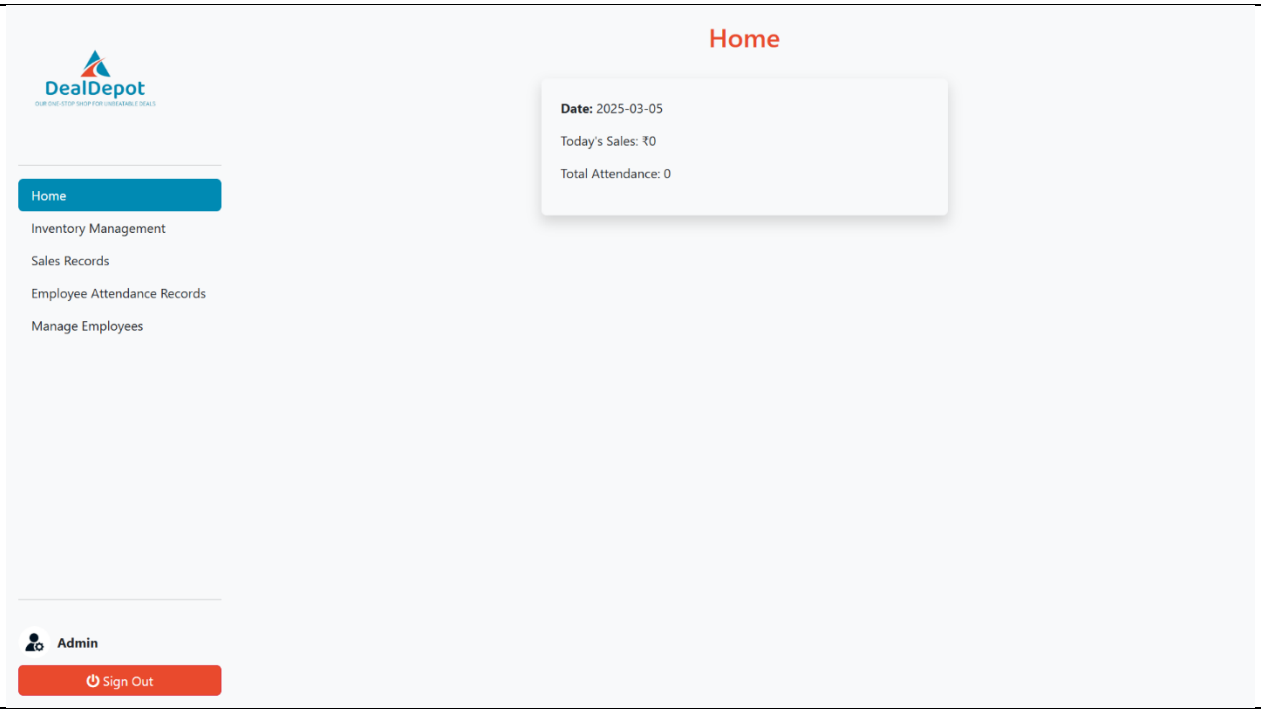
- Swagger UI (/docs) and ReDoc (/redoc):
  FastAPI automatically generates interactive API documentation, which helps during development and debugging of endpoints for login, product fetch, attendance punch, and bill generation.

This technology stack was chosen to provide a balance of performance, ease of development, scalability, and a modern user experience, while keeping the setup accessible for small to medium-sized businesses like local malls or retail stores.

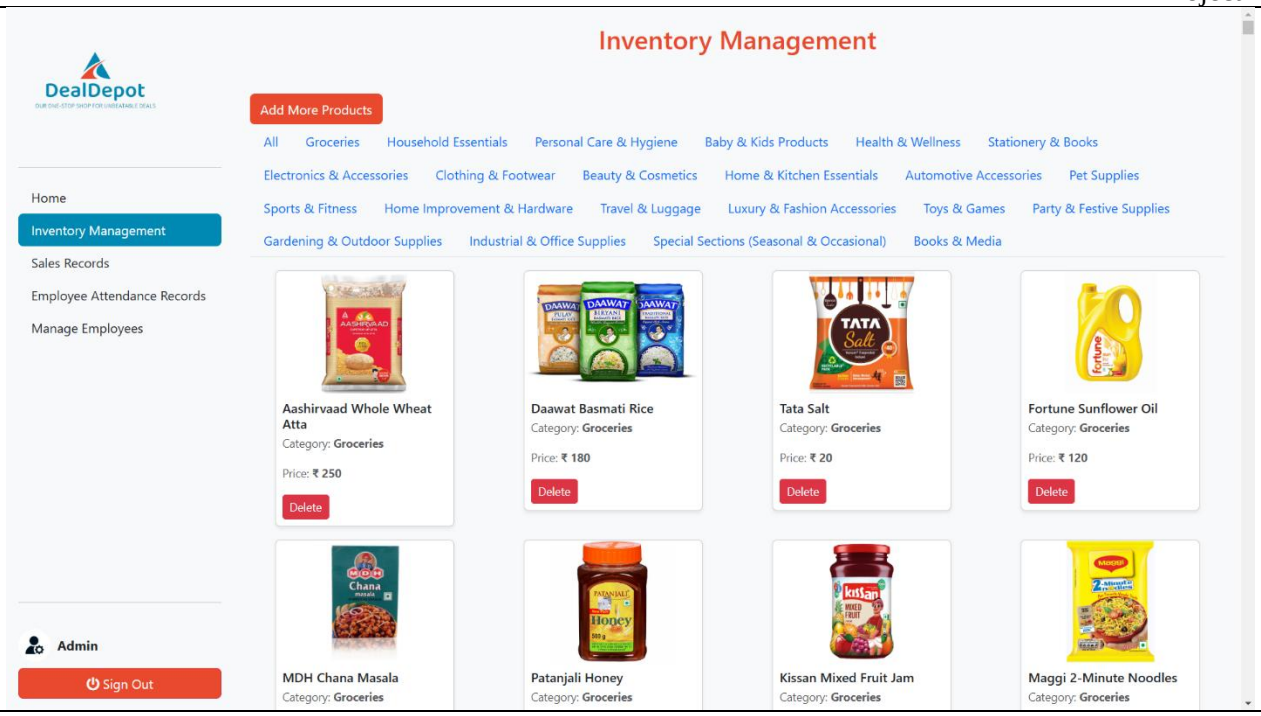
Chapter 6: UI Screens



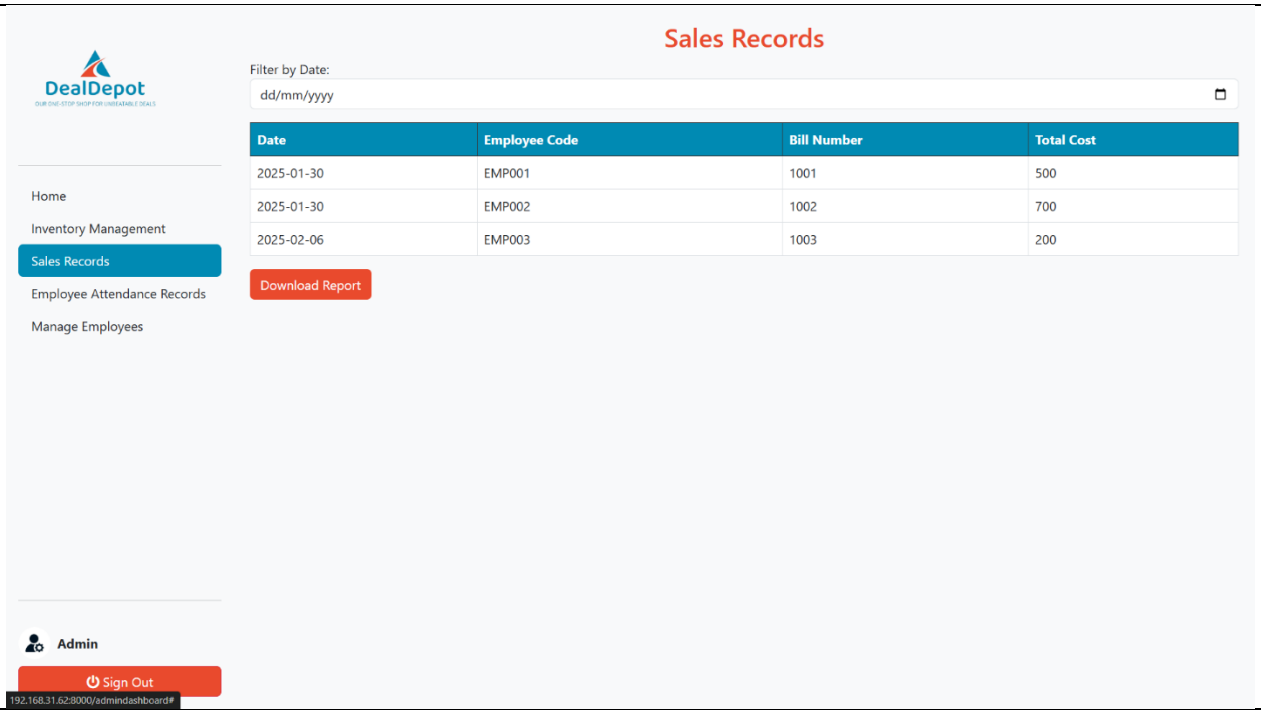
Login Page



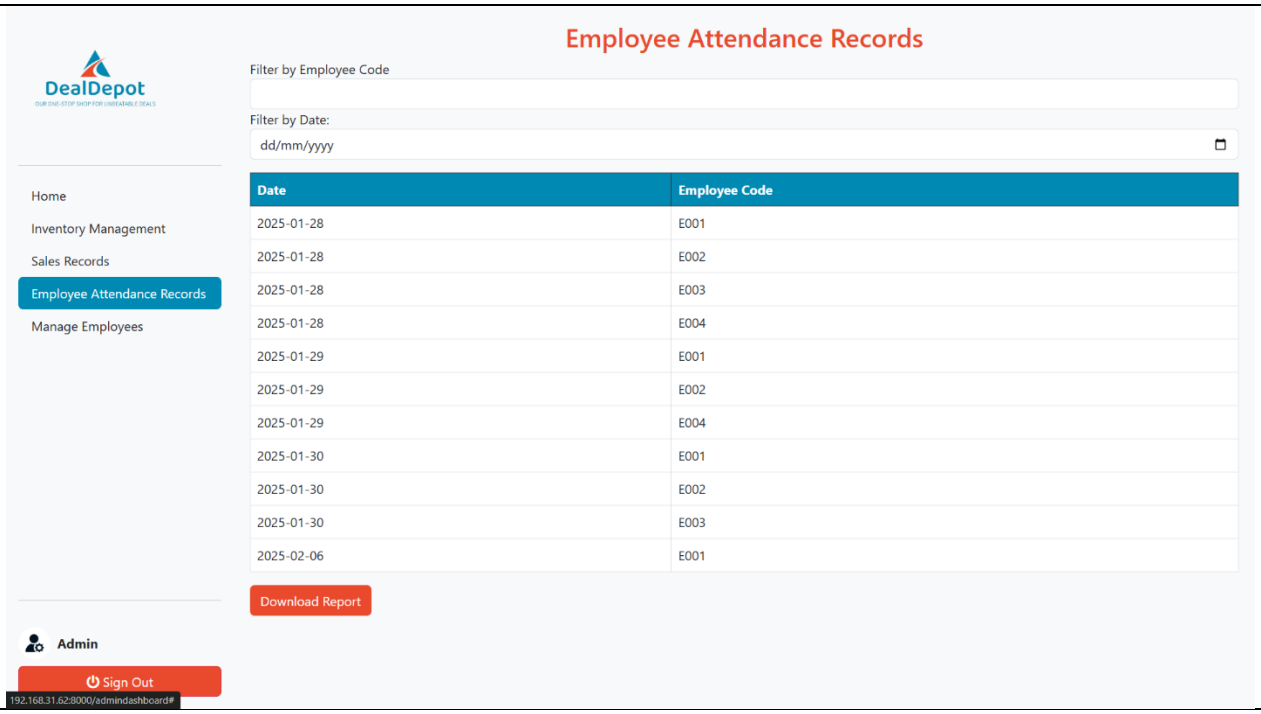
Admin Panel



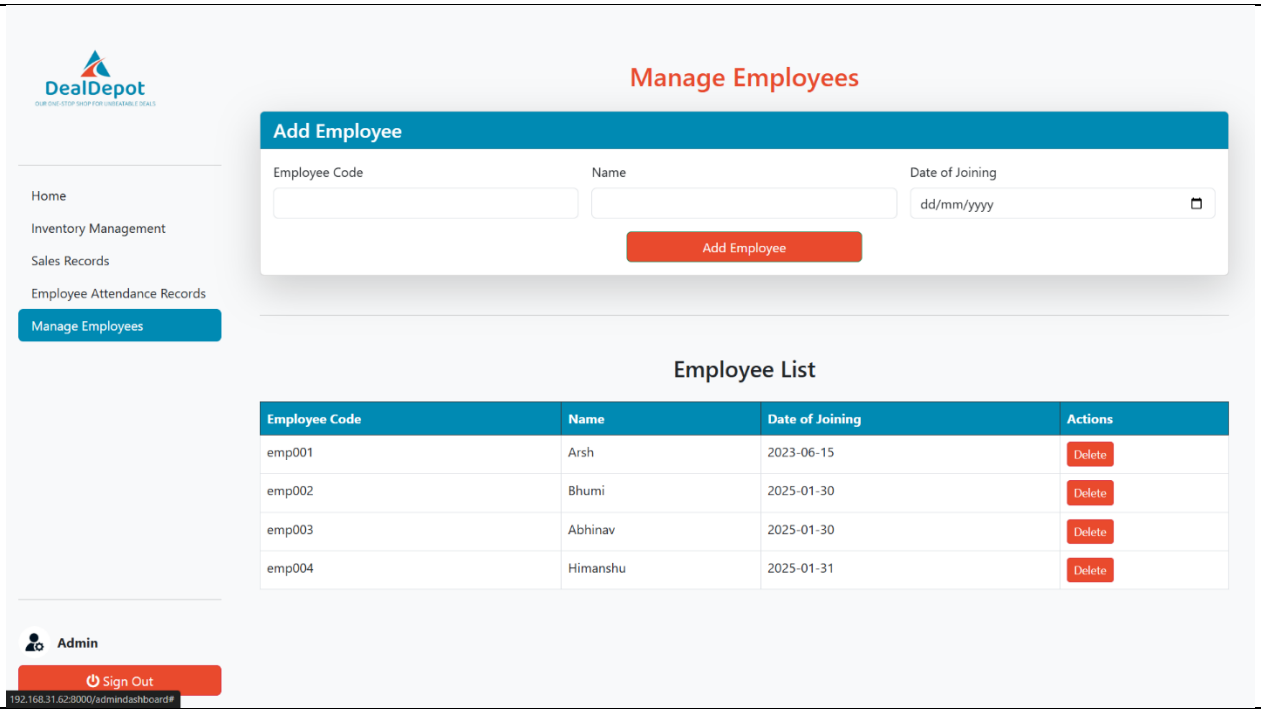
Admin Panel



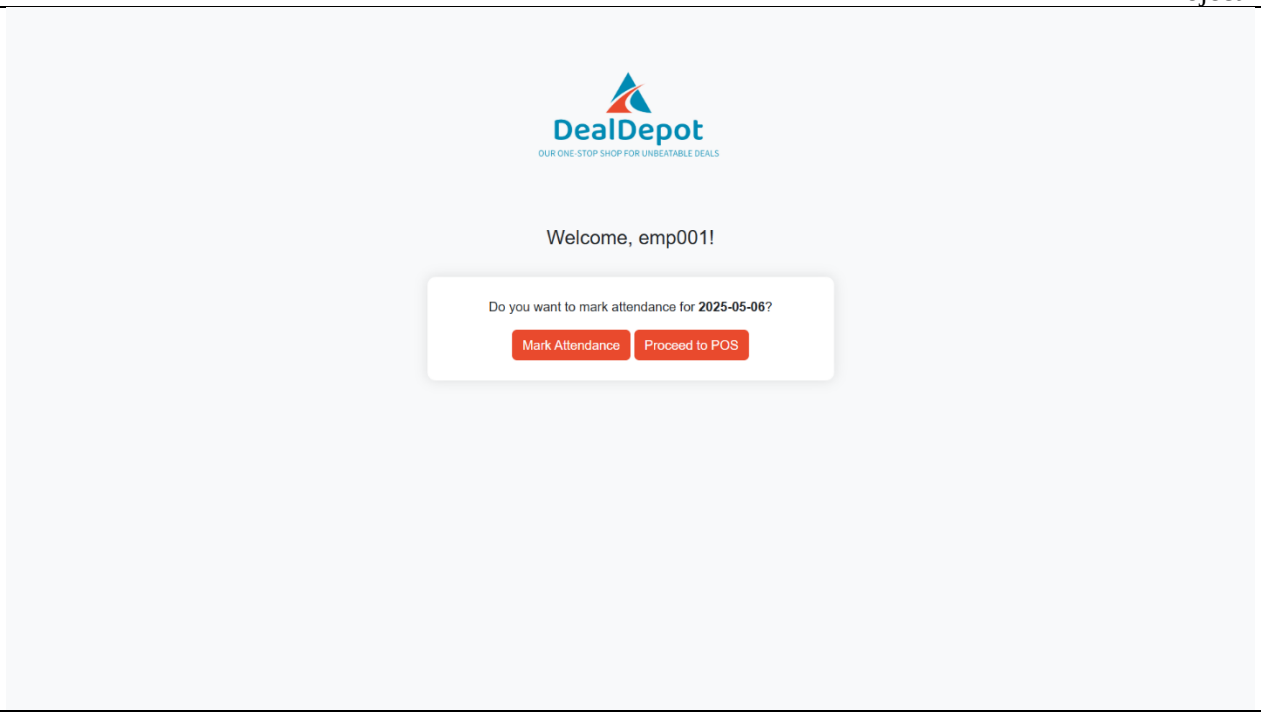
Admin Panel



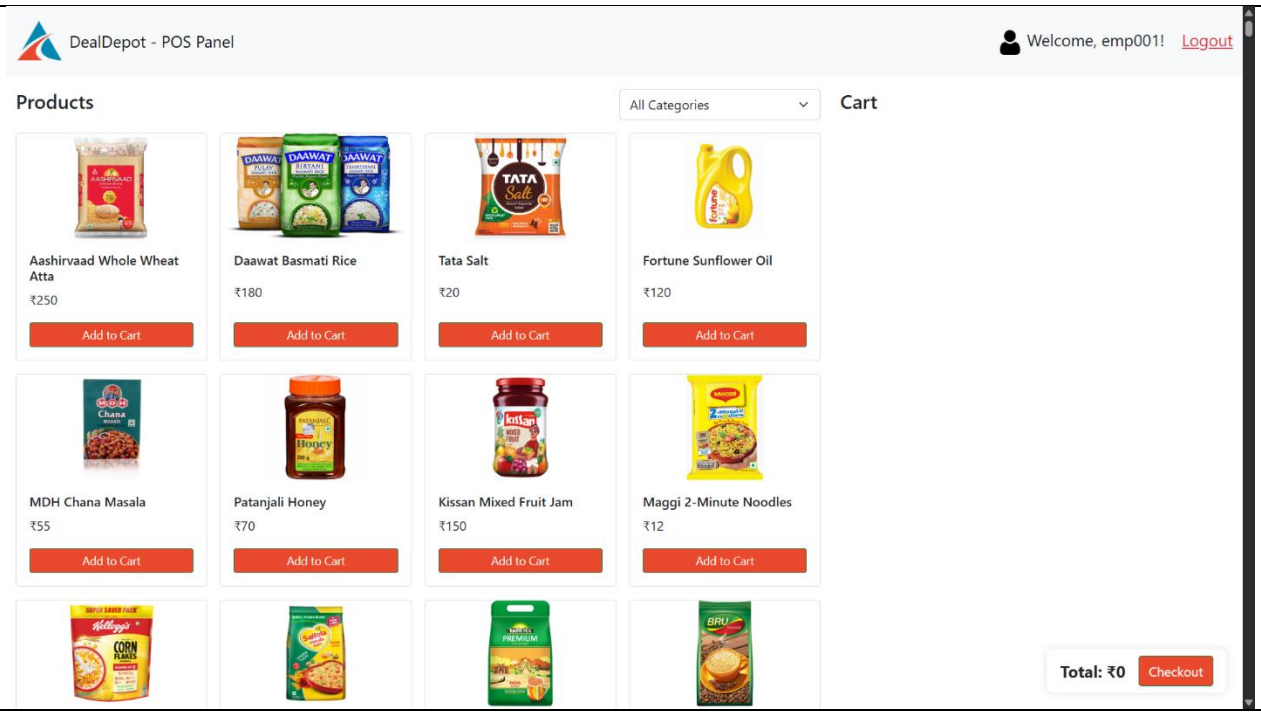
Admin Panel



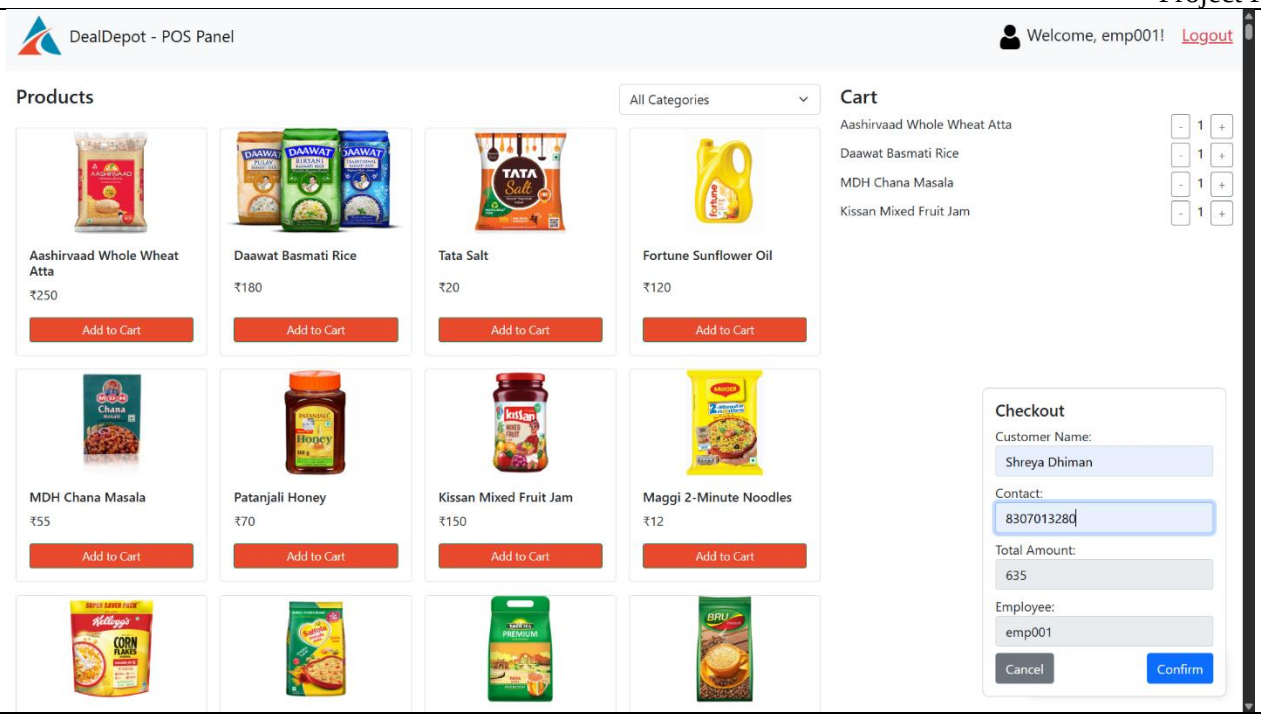
Admin Panel



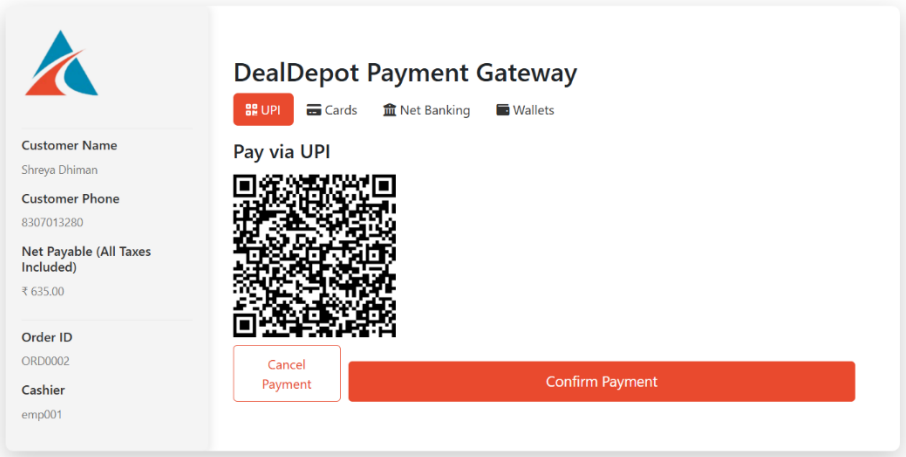
Employee Panel



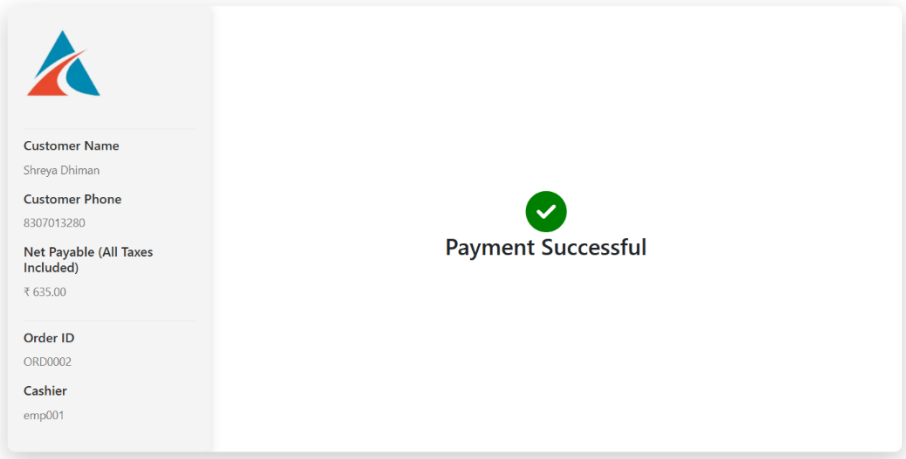
POS Panel



POS Panel



Payment Gateway



Payment Gateway

Chapter 7: Reports

Here are Sample Reports Auto-Generated within the project

DealDepot
OUR ONE-STOP SHOP FOR UNBEATABLE DEALS

Sales Records

Date	Employee Code	Bill Number	Total Cost
2025-04-22	emp002	1745295366386	125
2025-04-22	emp001	1745295399678	695
2025-04-22	emp002	1745295478205	200
2025-04-22	emp002	1745295563226	624
2025-04-22	emp002	1745295737771	6696
2025-04-22	emp002	1745298336554	17673
2025-04-22	emp003	1745299617404	250
2025-04-22	emp003	1745299641745	0
2025-04-22	emp002	1745300608786	375
2025-04-22	emp001	1745300667904	80
2025-04-22	emp002	1745303447172	320
2025-04-22	emp002	1745304377585	81298
2025-04-22	emp001	1745305723818	180

Sales Records – Admin Panel

DealDepot
OUR ONE-STOP SHOP FOR UNBEATABLE DEALS

Employee Attendance Records

Date	Employee Code
2025-04-22	emp002
2025-04-22	emp001
2025-04-22	emp003

Attendance Records – Admin Panel

Purchase Invoice

Customer: Shreya Dhiman

Phone: 8307013280

Cashier: emp001

Amount Paid: Rs. 12/-

Date: 5/6/2025

Bill Generated – Employee Panel

Chapter 8: Testing

8.1 Introduction to Testing

Testing is a crucial phase in the software development life cycle that ensures the system behaves as intended, meets specified requirements, and functions reliably in various scenarios. The primary aim of testing is to identify bugs or inconsistencies early and validate the correctness, completeness, and quality of the developed software. In the context of this project — *DealDepot: Inventory Management System* — testing helped us evaluate individual modules such as employee login, attendance punching, product management, billing, and reporting.

The project adopted a mix of testing approaches, including unit testing, integration testing, and manual testing through the API documentation generated via FastAPI's built-in tools like /docs and /redoc. These auto-generated documentation endpoints also served as a reference for endpoint testing.

8.2 Types of Testing Applied

1. Unit Testing
Each individual function or route was tested in isolation. Examples include:

- Login validation (ensuring users are authenticated with the right credentials)
- Attendance module (preventing multiple punch-ins for the same day)
- Product filtering by category
- Bill generation logic in POS screen

2. Integration Testing
Modules that rely on each other — such as employee login feeding into attendance, or POS interacting with product data and sales — were tested together to ensure they function seamlessly.

3. System Testing
End-to-end testing of the complete system was done in a real-world LAN-based deployment using Python’s socket module. Testers accessed the system from multiple devices to simulate employee counters.

4. User Acceptance Testing (UAT)
The system was used by team members and volunteer testers in a simulated retail setup. Their feedback helped validate the usability of the interface and reliability of core features.

8.3 Sample Test Cases

Test Case ID	Feature	Input	Expected Output	Actual Output	Status
TC001	User Login	Valid Employee Code	Redirect to Employee Dashboard	Redirects successfully	Pass
TC002	Attendance Punch-in	Already marked attendance	Show alert "Attendance already marked today"	Alert shown as expected	Pass
TC003	POS Cart	Select item, add quantity	Item added to cart with correct amount	Works as expected	Pass
TC004	Checkout & Billing	Customer form filled, payment success	Bill generated and sales saved	PDF bill generated, sale saved to sales.csv	Pass
TC005	Product Filter	Select category "Groceries"	Only grocery products visible	Category filter working	Pass
TC006	Admin Add Product	Submit new product via modal form	Product added and shown in inventory	Product visible in UI and products.csv	Pass
TC007	Invalid Login	Wrong Employee Code	Show alert "Invalid credentials"	Alert appears	Pass

8.4 Bug Reports & Fixes

- Issue: Attendance button allowed double punching if page wasn't refreshed.
  Fix: Backend check added to verify if entry for the day already exists.
- Issue: POS cart would not retain items on slow networks.
  Fix: Cart stored in sessionStorage and updated via JavaScript with better timeout handling.
- Issue: Products not filtered properly by category.
  Fix: Removed whitespace in category names and matched clean strings in JavaScript filtering logic.
- Issue: PDF not generated on some browsers.
  Fix: jsPDF updated to latest version and tested compatibility across Chrome, Firefox.

8.5 Test Summary

The DealDepot system underwent rigorous testing to ensure stability, reliability, and usability. All core modules — such as authentication, inventory handling, POS, and reporting — passed their functional tests. The use of FastAPI's /docs and /redoc interfaces was particularly helpful in testing and validating all available API endpoints systematically. Minimal bugs were encountered during testing, and all were resolved during development. The system is now ready for deployment in a small to mid-sized retail environment such as a supermarket or mall, with confidence in its functionality and performance.

Chapter 9: Future Enhancements

As with any software system, *DealDepot: Inventory Management System* has room for further development and scaling. The current system is fully functional for small to medium-sized retail environments, but to adapt to larger business needs and modern retail trends, several enhancements can be considered in future iterations.

9.1 Multi-Branch and Cloud Support

At present, the system operates in a single location environment using LAN hosting. In future versions, support for multi-branch management can be introduced. By integrating cloud storage and deploying the backend on platforms like AWS or Azure, administrators will be able to monitor and control inventory, employees, and sales across multiple branches from a central dashboard.

9.2 Mobile Application Integration

While the current system is browser-based, future updates can include dedicated Android/iOS applications for employees and administrators. Employees could take attendance or process bills using a mobile device, and administrators could view real-time sales summaries and reports on the go.

9.3 Role-Based Access Control (RBAC)

At present, the system separates users as either employees or admins. However, more granular role-based access control can be implemented. For example:

- Managers can approve returns
- Cashiers can only create bills
- Inventory staff can manage stock but not view financial data

This will improve both security and accountability.

9.4 Real-Time Data Sync & WebSocket Integration

Integrating real-time data synchronization using WebSockets or technologies like FastAPI WebSocket endpoints can allow instant updates of stock availability, sales figures, or attendance changes across all connected users.

9.5 Barcode/QR Code Scanner Integration

Currently, QR codes are generated using qrcode.js for billing. Future enhancements can support barcode or QR code scanning using hardware devices or mobile cameras to speed up product selection during billing and reduce manual entry errors.

9.6 Advanced Reporting & Analytics

Basic reports are currently stored in CSV format. An upgraded system could include:

- Graphical dashboards using libraries like Chart.js or D3.js
- Predictive analytics using sales trends
- Downloadable Excel/PDF reports with filters and summaries

9.7 Customer Loyalty and Billing System

Adding modules to manage customer profiles, purchase history, and loyalty points would significantly increase the value of the system. This can help in personalized marketing and increasing customer retention.

9.8 Database Migration

Currently, CSV files are used for simplicity and ease of access. Migrating to a full-fledged relational database like PostgreSQL or MySQL would allow better data integrity, faster queries, and scalability for large datasets.

Chapter 10: Conclusions

The development of the *DealDepot: Inventory Management System* marks a significant step toward the digital transformation of small to medium-scale retail businesses, particularly malls and multi-counter retail outlets. The project has been successfully conceptualized, designed, developed, and implemented to cater to real-time inventory tracking, sales processing, attendance logging, and centralized management from an admin dashboard.

Throughout the development lifecycle, the project has addressed the key shortcomings of conventional inventory systems by offering a centralized, easily accessible, and modular solution that allows seamless interaction between different stakeholders – primarily administrators (store owners) and employees (billing staff). Using a modern tech stack based on FastAPI, CSV-based file storage, and HTML/CSS/JavaScript on the frontend, DealDepot ensures a lightweight yet powerful system that runs efficiently over a local network.

The implementation of real-time features like POS cart handling, QR code billing, and PDF generation using jsPDF enhances the user experience and minimizes manual errors during transactions. Furthermore, the use of FastAPI's auto-documented endpoints and asynchronous capabilities has allowed us to create a well-structured and scalable backend.

By using local hosting over LAN and incorporating common frontend technologies, the system remains cost-effective and easy to deploy without reliance on third-party cloud infrastructure. The documentation, testing, and modular folder organization ensure that future enhancements or migrations (such as shifting to SQL databases or cloud APIs) can be achieved without rewriting the core logic.

In conclusion, *DealDepot* stands as a practical solution for inventory and billing management in small to medium retail environments. It not only meets the current operational requirements but also sets the foundation for future upgrades like online access, analytics, and customer management. This project demonstrates the ability to integrate various technologies into a cohesive, user-friendly product that addresses real-world business needs.

Part C:

ENDNOTE

Bibliography

- **W3Schools** – Official tutorials and references for HTML, CSS, JavaScript, and Bootstrap.
  https://www.w3schools.com/

- **Geeks for Geeks** – Articles and code examples related to Python, web development, and data structures.
  https://www.geeksforgeeks.org/

- **FastAPI Documentation** – Official documentation for FastAPI, covering asynchronous APIs, routing, and OpenAPI generation.
  https://fastapi.tiangolo.com/

- **Python Official Documentation** – For language syntax, built-in libraries, and socket programming concepts.
  https://docs.python.org/3/

- **Bootstrap Documentation** – For frontend design components and responsive layouts.
  https://getbootstrap.com/

- **jsPDF Documentation** – For generating PDF invoices directly from JavaScript.
  https://github.com/parallax/jsPDF

- **QRCode.js Documentation** – For client-side generation of QR codes used in payment and billing.
  https://davidshimjs.github.io/qrcodejs/

- **MDN Web Docs** – For detailed reference on web APIs, JavaScript, and HTML/CSS standards.
  https://developer.mozilla.org/

- **Jinja2 Documentation** – For HTML templating used in FastAPI rendering.
  https://jinja.palletsprojects.com/

- **Socket Programming Resources** – Various online resources and tutorials to implement basic LAN hosting using Python sockets.
  (Used collectively from educational blogs and forums)

- **YouTube Tutorials** – For practical understanding of FastAPI, Bootstrap, and POS system integration.
  (Credited to CodeWithHarry in appendix or footnotes of implementation sections)

- **The Python Cookbook** by Team CodeWithTechries – Practical code examples and patterns used for backend logic and API development.

- **Beyond the Browser** by Team CodeWithTechries – Reference for integrating web apps with native hardware/network functionalities.

Appendices

Appendix A: API Documentation (ReDoc)

This appendix provides a visual snapshot of the ReDoc-based API documentation generated by FastAPI. It gives a comprehensive overview of the backend endpoints used in the DealDepot – Inventory Management System.

The ReDoc documentation includes:

- A categorized list of all available API endpoints.
- HTTP methods associated with each endpoint such as GET, POST, etc.
- Required and optional parameters for each endpoint.
- Schema of request and response bodies.
- Descriptive tags for logically grouping endpoints (e.g., Employee Operations, POS System, Attendance, Admin Controls).
- Error response formats for easier debugging and consistency.

This documentation helps developers and testers understand how to interact with the backend effectively. Screenshots of the ReDoc UI are attached below for reference. The live documentation is accessible during runtime at the route /redoc.

Appendix B: OpenAPI Specification (openapi.json)

The OpenAPI Specification file (openapi.json) represents the complete programmatic blueprint of the backend API. It was auto-generated by FastAPI and defines all endpoints, models, schemas, and validation rules.

The file contains:

- General metadata about the API (title, version, description).
- Server information including base URLs.
- Detailed path definitions for each route (e.g., /login, /attendance, /sales).
- JSON Schema-based definitions for each request and response.
- Authentication/security schemes (if applicable).

Due to its size, only selected portions of the JSON are presented here, highlighting key parts of the backend structure. The complete file is available in the documents directory of the project repository under the name openapi.json.

```
{
"openapi": "3.1.0",
"info": {
"title": "FastAPI",
"version": "0.1.0"
},
"paths": {
"/": {
"get": {
"summary": "Show Login Page",
"operationId": "show_login_page__get",
"responses": {
"200": {
"description": "Successful Response",
"content": {
"text/html": {
"schema": {
"type": "string"
}
}
}
}
}
}
},
"/login": {
"post": {
"summary": "Login User",
"operationId": "login_user_login_post",
"requestBody": {
"content": {
"application/x-www-form-urlencoded": {
"schema": {
"$ref": "#/components/schemas/Body_login_user_login_post"
}
}
},
"required": true
},
"responses": {
"200": {
"description": "Successful Response",
"content": {
"application/json": {
"schema": {}
}
}
},
"422": {
"description": "Validation Error",
"content": {
"application/json": {
```

```
"schema": {
"$ref": "#/components/schemas/HTTPValidationError"
}
}
}
}
}
}
},
"/admindashboard": {
"get": {
"summary": "Show Admin Dashboard",
"operationId": "show_admin_dashboard_admindashboard_get",
"responses": {
"200": {
"description": "Successful Response",
"content": {
"text/html": {
"schema": {
"type": "string"
}
}
}
}
}
}
},
"/empdashboard": {
"get": {
"summary": "Show Employee Dashboard",
"operationId": "show_employee_dashboard_empdashboard_get",
"responses": {
"200": {
"description": "Successful Response",
"content": {
"text/html": {
"schema": {
"type": "string"
}
}
}
}
}
}
},
"/pos": {
"get": {
"summary": "Show Employee Dashboard",
"operationId": "show_employee_dashboard_pos_get",
"responses": {
"200": {
"description": "Successful Response",
```

```
"content": {
"text/html": {
"schema": {
"type": "string"
}
}
}
}
}
}
},
"/pg": {
"get": {
"summary": "Show Employee Dashboard",
"operationId": "show_employee_dashboard_pg_get",
"responses": {
"200": {
"description": "Successful Response",
"content": {
"text/html": {
"schema": {
"type": "string"
}
}
}
}
}
}
},
"/api/employees": {
"get": {
"summary": "Get Employees",
"operationId": "get_employees_api_employees_get",
"responses": {
"200": {
"description": "Successful Response",
"content": {
"application/json": {
"schema": {}
}
}
}
}
},
"post": {
"summary": "Add Employee",
"operationId": "add_employee_api_employees_post",
"requestBody": {
"content": {
"application/x-www-form-urlencoded": {
"schema": {
"$ref": "#/components/schemas/Body_add_employee_api_employees_post"
```

```
}
}
},
"required": true
},
"responses": {
"200": {
"description": "Successful Response",
"content": {
"application/json": {
"schema": {}
}
}
},
"422": {
"description": "Validation Error",
"content": {
"application/json": {
"schema": {
"$ref": "#/components/schemas/HTTPValidationError"
}
}
}
}
}
}
},
"/api/employees/{emp_code}": {
"delete": {
"summary": "Delete Employee",
"operationId": "delete_employee_api_employees__emp_code__delete",
"parameters": [
{
"name": "emp_code",
"in": "path",
"required": true,
"schema": {
"type": "string",
"title": "Emp Code"
}
}
],
"responses": {
"200": {
"description": "Successful Response",
"content": {
"application/json": {
"schema": {}
}
}
},
"422": {
```

```
"description": "Validation Error",
"content": {
"application/json": {
"schema": {
"$ref": "#/components/schemas/HTTPValidationError"
}
}
}
}
}
}
},
"/api/today": {
"get": {
"summary": "Fetch Today Data",
"operationId": "fetch_today_data_api_today_get",
"responses": {
"200": {
"description": "Successful Response",
"content": {
"application/json": {
"schema": {}
}
}
}
}
}
},
"/api/products": {
"get": {
"summary": "Get Products",
"operationId": "get_products_api_products_get",
"responses": {
"200": {
"description": "Successful Response",
"content": {
"application/json": {
"schema": {}
}
}
}
}
}
},
"/api/add_product": {
"post": {
"summary": "Add Product",
"operationId": "add_product_api_add_product_post",
"requestBody": {
"content": {
"application/json": {
"schema": {
```

```
"$ref": "#/components/schemas/Product"
}
}
},
"required": true
},
"responses": {
"200": {
"description": "Successful Response",
"content": {
"application/json": {
"schema": {}
}
}
},
"422": {
"description": "Validation Error",
"content": {
"application/json": {
"schema": {
"$ref": "#/components/schemas/HTTPValidationError"
}
}
}
}
}
}
},
"/api/products/{product_name}": {
"delete": {
"summary": "Delete Product",
"operationId": "delete_product_api_products__product_name__delete",
"parameters": [
{
"name": "product_name",
"in": "path",
"required": true,
"schema": {
"type": "string",
"title": "Product Name"
}
}
],
"responses": {
"200": {
"description": "Successful Response",
"content": {
"application/json": {
"schema": {}
}
}
},
```

```
"422": {
"description": "Validation Error",
"content": {
"application/json": {
"schema": {
"$ref": "#/components/schemas/HTTPValidationError"
}
}
}
}
}
}
},
"/api/sales": {
"get": {
"summary": "Get Sales",
"operationId": "get_sales_api_sales_get",
"responses": {
"200": {
"description": "Successful Response",
"content": {
"application/json": {
"schema": {}
}
}
}
}
}
},
"/api/save_sales": {
"post": {
"summary": "Save Sales",
"operationId": "save_sales_api_save_sales_post",
"requestBody": {
"content": {
"application/json": {
"schema": {
"$ref": "#/components/schemas/Sale"
}
}
},
"required": true
},
"responses": {
"200": {
"description": "Successful Response",
"content": {
"application/json": {
"schema": {}
}
}
},
```

```
"422": {
"description": "Validation Error",
"content": {
"application/json": {
"schema": {
"$ref": "#/components/schemas/HTTPValidationError"
}
}
}
}
}
}
},
"/api/attendance": {
"get": {
"summary": "Get Attendance",
"operationId": "get_attendance_api_attendance_get",
"responses": {
"200": {
"description": "Successful Response",
"content": {
"application/json": {
"schema": {}
}
}
}
}
}
},
"/mark_attendance": {
"post": {
"summary": "Mark Attendance",
"operationId": "mark_attendance_mark_attendance_post",
"requestBody": {
"content": {
"application/x-www-form-urlencoded": {
"schema": {
"$ref": "#/components/schemas/Body_mark_attendance_mark_attendance_post"
}
}
},
"required": true
},
"responses": {
"200": {
"description": "Successful Response",
"content": {
"application/json": {
"schema": {}
}
}
},
```

```
"422": {
"description": "Validation Error",
"content": {
"application/json": {
"schema": {
"$ref": "#/components/schemas/HTTPValidationError"
}
}
}
}
}
}
}
},
"components": {
"schemas": {
"Body_add_employee_api_employees_post": {
"properties": {
"emp_code": {
"type": "string",
"title": "Emp Code"
},
"name": {
"type": "string",
"title": "Name"
},
"doj": {
"type": "string",
"title": "Doj"
}
},
"type": "object",
"required": [
"emp_code",
"name",
"doj"
],
"title": "Body_add_employee_api_employees_post"
},
"Body_login_user_login_post": {
"properties": {
"username": {
"type": "string",
"title": "Username"
},
"password": {
"type": "string",
"title": "Password"
}
},
"type": "object",
"required": [
```

"username",
"password"
],
"title": "Body_login_user_login_post"
},
"Body_mark_attendance_mark_attendance_post": {
"properties": {
"emp_code": {
"type": "string",
"title": "Emp Code"
}
},
"type": "object",
"required": [
"emp_code"
],
"title": "Body_mark_attendance_mark_attendance_post"
},
"HTTPValidationError": {
"properties": {
"detail": {
"items": {
"$ref": "#/components/schemas/ValidationError"
},
"type": "array",
"title": "Detail"
}
},
"type": "object",
"title": "HTTPValidationError"
},
"Product": {
"properties": {
"product_name": {
"type": "string",
"title": "Product Name"
},
"image_address": {
"type": "string",
"title": "Image Address"
},
"product_category": {
"type": "string",
"title": "Product Category"
},
"product_price": {
"type": "number",
"title": "Product Price"
},
"product_quantity": {
"type": "integer",
"title": "Product Quantity"

```
}
},
"type": "object",
"required": [
"product_name",
"image_address",
"product_category",
"product_price",
"product_quantity"
],
"title": "Product"
},
"Sale": {
"properties": {
"name": {
"type": "string",
"title": "Name"
},
"amount": {
"type": "number",
"title": "Amount"
},
"emp_code": {
"type": "string",
"title": "Emp Code"
},
"date": {
"type": "string",
"title": "Date"
},
"bill_no": {
"type": "integer",
"title": "Bill No"
}
},
"type": "object",
"required": [
"name",
"amount",
"emp_code",
"date",
"bill_no"
],
"title": "Sale"
},
"ValidationError": {
"properties": {
"loc": {
"items": {
"anyOf": [
{
"type": "string"
```

```
},
{
"type": "integer"
}
]
},
"type": "array",
"title": "Location"
},
"msg": {
"type": "string",
"title": "Message"
},
"type": {
"type": "string",
"title": "Error Type"
}
},
"type": "object",
"required": [
"loc",
"msg",
"type"
],
"title": "ValidationError"
}
}
}
}
```

List of Figures