COMPUTER SCIENCE PROJECT

OMNIFLOW

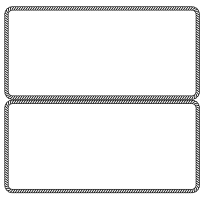
UNIVERSAL ORDER PROCESSING & BILLING SYSTEM

WITH PYTHON, CUSTOMTKINTER AND MYSQL

ADIEESH V K
XII C
12302

BONAFIDE CERTIFICATE

Register No:



Certified to the Bonafide Project Work in COMPUTER SCIENCE Done by _____ of Class _XII_, Section _C_ of DAV Boys Senior Secondary School, Chennai – 86.
During the year 20__ to 20__.

_____ Signature of Principal
School Seal

_____ Signature of Subject Teacher

Submitted for the Practical Examination held on _____, at DAV Boys Senior Secondary School.

Internal Examiner:	External Examiner:

Chief Superintendent: _____
Date:

ACKNOWLEDGEMENT

I would like to express my heartfelt gratitude to all the teachers and mentors who have shaped us into what we are today.

I would also like to extend my immense gratitude to Mrs. Hemalatha Loganathan, my Computer Science teachers and, Mrs. Swarna Karpagavalli, the Principal of DAV Boys Sr. Sec. School.

I thank them for providing me this unique opportunity to express my thoughts in a creative format.

Further, I am immensely grateful to my parents who have served as stable pillars to the cause of Education and supported me in every way possible.

Finally, I thank my friends without whom the activity would not have been what it is.

CONTENTS

Introduction

OmniFlow is a universal order, billing, and customer management system designed to streamline operations for small and medium-sized businesses across various industries. While the demonstration example showcases a tailoring workflow, the software itself is fully customizable and adaptable to any business that requires order tracking, customer database management, and automated invoice/job card generation.

The project focuses on reducing manual workload by digitizing key business processes such as order entry, customer handling, delivery date tracking, and record-keeping. With an intuitive modern UI, real-time analytics dashboard, and database-backed data management, OmniFlow enhances efficiency, accuracy, and productivity.

Built using Python, CustomTkinter, MySQL, and ReportLab, the system provides a practical demonstration of how software can automate and optimize everyday business operations.

Project Overview

OmniFlow is a versatile business workflow automation application designed to streamline daily operations for small and medium-sized businesses across various industries. The system aims to replace traditional manual record-keeping with a unified, digital platform capable of managing customer data, processing orders, generating invoices, tracking delivery timelines, and offering real-time business insights.

OmniFlow features an intuitive modern UI created with CustomTkinter, enabling smooth navigation between core modules such as Dashboard, Order Management, Customer Records, and Settings. The application uses MySQL as its backend, providing reliable, structured, and scalable data storage. Automated PDF generation enables professional invoices and job cards to be created within seconds, further enhancing operational efficiency.

Additionally, the Dashboard provides visual analytics such as order distribution, revenue trends, and garment/category-wise breakdowns using Matplotlib charts. These insights empower businesses to make informed decisions and analyze patterns over time. With its modular architecture, OmniFlow can be easily adapted to different business domains like printing shops, repair centers, boutiques, customized gifts, and service-based industries.

Technologies Used

- **Python 3.13** – Core programming language used to build the application logic.
- **CustomTkinter** – Provides a modern and visually appealing GUI with dark mode support.
- **MySQL Database** – Stores orders, customers, users, and settings in a structured and scalable format.
- **ReportLab** – Generates professional PDF invoices and job cards programmatically.
- **Matplotlib** – Creates interactive charts and analytics for the dashboard.
- **JSON Module** – Stores flexible measurement data in an easily serializable format.
- **Datetime Module** – Handles date formatting, delivery dates, and timestamps.
- **tkcalendar** – Provides a user-friendly date selector for delivery dates.
- **mysql.connector** – Allows Python to communicate with the MySQL database.

Features of the Application

6.1 Login System :-

- Secure login for users.
- Username and password validation.

6.2 Dashboard :-

- Shows total orders.
- Shows pending orders.
- Displays monthly revenue.
- Pie chart of garment types.
- Bar chart of status distribution.

6.3 New Order Creation :-

- Customer autocomplete suggestions.
- Dynamic measurements section.
- DateEntry for selecting delivery date.
- Ability to save order, generate job card, or create invoice.

6.4 Manage Orders :-

- View all orders.
- Filter by status.
- Update order status.
- View full order details.
- Delete order.

6.5 Customer Management :-

- View all customers.
- See order history and total spending.

6.6 PDF Generation :-

- Job Cards with measurements.
- Invoices with tax calculation and totals.

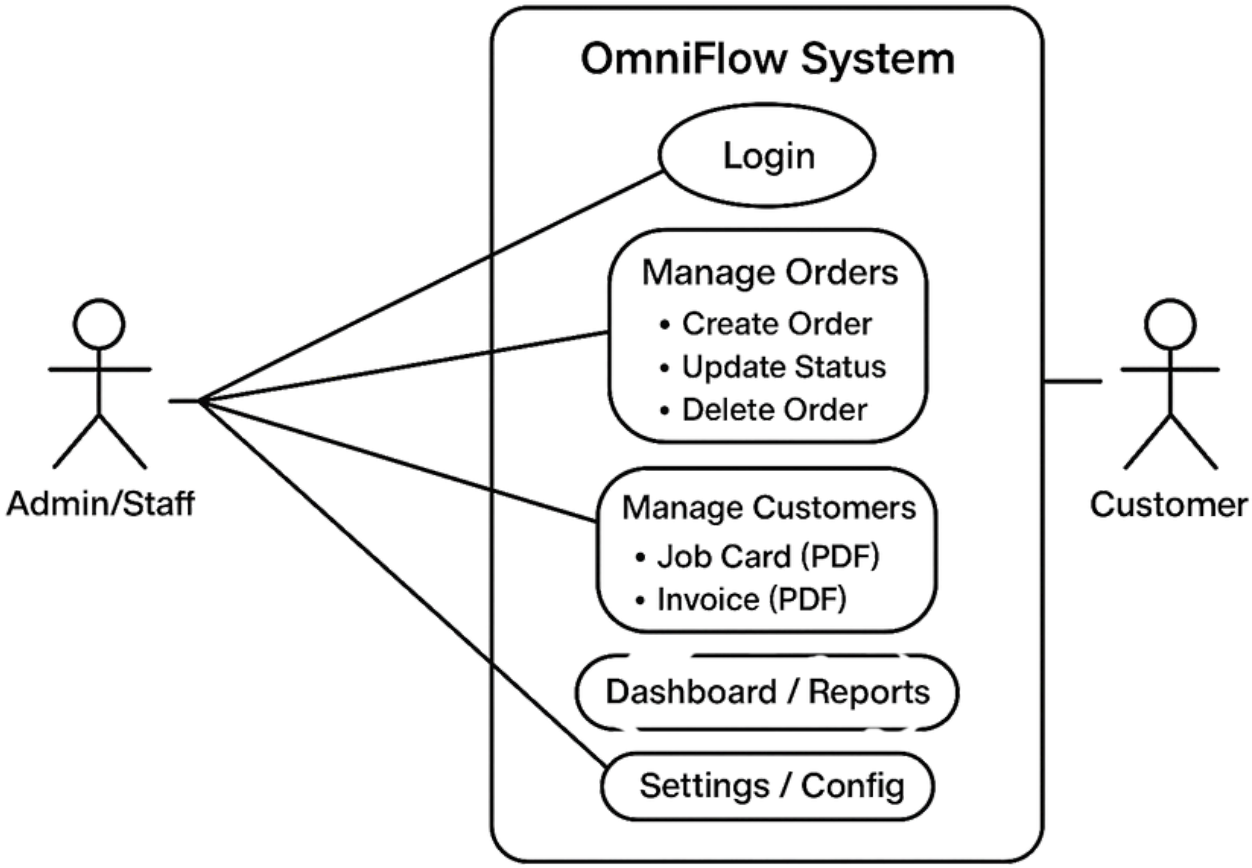
6.7 Settings Page :-

- Update shop name, address, phone, tax rate.

System Design

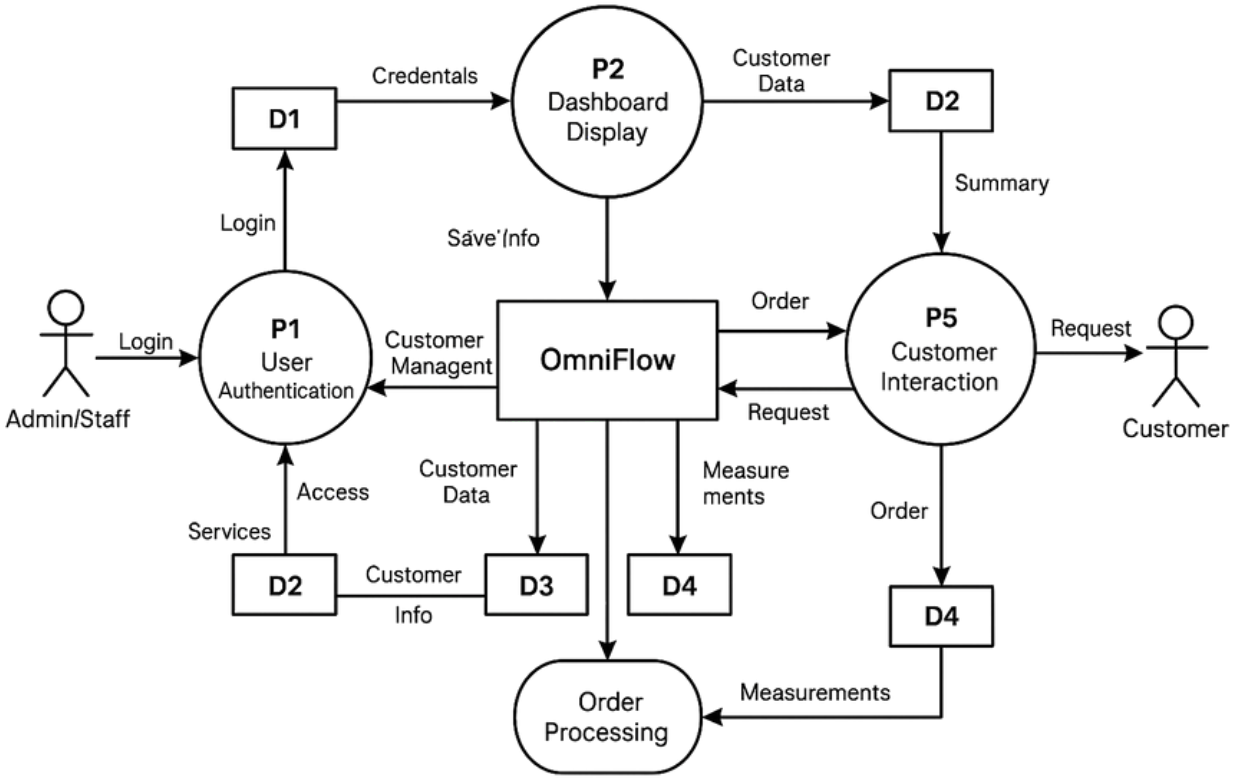
System Design defines how OmniFlow is structured internally. It includes the system's architecture, diagrams, data flow, and interaction between users and the system. These design models help understand the system before implementation.

Use Case Diagram :-



The Use Case Diagram of OmniFlow represents the interaction between the user and the various features of the system. The admin can log in, manage customers etc. Each use case illustrates a specific function the user can perform, showing how OmniFlow supports smooth business operations by organizing essential tasks like data entry, order tracking, document creation, and settings management.

Data Flow Diagram (DFD) :-



The Data Flow Diagram (DFD) for OmniFlow shows how data flows between users, system processes, and the database. At the context level the system interacts with Admin/Staff (who create orders, manage customers and generate documents). The Level-1 DFD breaks the system into key processes — Authentication, Order Management, Customer Management, Document Generation and Dashboard/Reporting which read from and write to data stores (Users, Orders, Customers, Settings). Data flows (e.g., order details, customer lookup, PDF output, and aggregated metrics) move between these processes and stores, ensuring a clear, auditable path from user actions to persistent storage and generated outputs.

Program code

```
import customtkinter as ctk
from tkinter import messagebox
from tkcalendar import DateEntry
import mysql.connector
import json
from datetime import datetime
from reportlab.lib.pagesizes import letter
from reportlab.pdfgen import canvas
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from matplotlib.figure import Figure
connection = None
logged_in_user = None
window = None
main_area = None
form_fields = {}
#<------------------------------------------------------------>
def connect_database():
 global connection
 try:
  connection = mysql.connector.connect(
   host='localhost',
   user='root',
   password='sasikala',
   database='stitchsync'
  )
  print("Database connected!")
  return True
 except:
  try:
   temp_conn = mysql.connector.connect(
    host='localhost',
    user='root',
    password='sasikala'
   )
   cursor = temp_conn.cursor()
   cursor.execute("CREATE DATABASE stitchsync")
   cursor.close()
   temp_conn.close()
   print("Database created!")
   return connect_database()
  except Exception as e:
   messagebox.showerror("Error", f"Cannot connect: {e}")
   return False
#<------------------------------------------------------------>
def create_tables():
 global connection
 try:
  cursor = connection.cursor()
  cursor.execute("""
```

```
    CREATE TABLE IF NOT EXISTS users (
      id INT AUTO_INCREMENT PRIMARY KEY,
      username VARCHAR(50) UNIQUE NOT NULL,
      password VARCHAR(255) NOT NULL
    )
    """)
    cursor.execute("SELECT COUNT(*) FROM users WHERE username = 'admin'")
    if cursor.fetchone()[0] == 0:
      cursor.execute("INSERT INTO users (username, password) VALUES ('admin',
'admin123')")
    cursor.execute("""
      CREATE TABLE IF NOT EXISTS customers (
        id INT AUTO_INCREMENT PRIMARY KEY,
        name VARCHAR(100) NOT NULL,
        contact VARCHAR(20) UNIQUE NOT NULL,
        created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
      )
    """)
    cursor.execute("""
      CREATE TABLE IF NOT EXISTS orders (
        id INT AUTO_INCREMENT PRIMARY KEY,
        order_id VARCHAR(20) UNIQUE NOT NULL,
        customer_name VARCHAR(100) NOT NULL,
        contact VARCHAR(20) NOT NULL,
        garment_type VARCHAR(50) NOT NULL,
        fabric VARCHAR(50) NOT NULL,
        measurements TEXT NOT NULL,
        collar_type VARCHAR(50),
        sleeve_type VARCHAR(50),
        fit_type VARCHAR(50),
        delivery_date DATE NOT NULL,
        notes TEXT,
        price DECIMAL(10, 2) NOT NULL,
        status VARCHAR(20) DEFAULT 'Pending',
        created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
      )
    """)
    cursor.execute("""
      CREATE TABLE IF NOT EXISTS settings (
        id INT PRIMARY KEY,
        shop_name VARCHAR(100) NOT NULL,
        address TEXT NOT NULL,
        phone VARCHAR(20) NOT NULL,
        tax_rate DECIMAL(5, 2) NOT NULL
      )
    """)
    cursor.execute("SELECT COUNT(*) FROM settings WHERE id = 1")
    if cursor.fetchone()[0] == 0:
      cursor.execute("""
        INSERT INTO settings (id, shop_name, address, phone, tax_rate)
        VALUES (1, 'StitchSync Tailors', '123 Fashion Street', '+91 9876543210', 5.00)
      """)
    connection.commit()
    cursor.close()
    print("Tables created!")
```

```
    return True
  except Exception as e:
    print(f"Error: {e}")
    return False
#<------------------------------------------------------------>
def run_query(query, values=None):
  global connection
  try:
    cursor = connection.cursor()
    if values:
      cursor.execute(query, values)
    else:
      cursor.execute(query)
    connection.commit()
    cursor.close()
    return True
  except Exception as e:
    print(f"Query error: {e}")
    return False
#<------------------------------------------------------------>
def fetch_data(query, values=None):
  global connection
  try:
    cursor = connection.cursor(dictionary=True)
    if values:
      cursor.execute(query, values)
    else:
      cursor.execute(query)
    result = cursor.fetchall()
    cursor.close()
    return result
  except Exception as e:
    print(f"Fetch error: {e}")
    return []
#<------------------------------------------------------------>
def fetch_one(query, values=None):
  result = fetch_data(query, values)
  if result:
    return result[0]
  return None
#<------------------------------------------------------------>
def clear_screen():
  global window
  for widget in window.winfo_children():
    widget.destroy()
#<------------------------------------------------------------>
def show_login_page():
  global window, logged_in_user
  clear_screen()
  login_frame = ctk.CTkFrame(window, width=400, height=400, corner_radius=20)
  login_frame.place(relx=0.5, rely=0.5, anchor="center")
  login_frame.pack_propagate(False)
  title = ctk.CTkLabel(login_frame, text="StitchSync",
        font=("Helvetica", 38, "bold"))
```

```
 title.pack(pady=(25,5))
 subtitle = ctk.CTkLabel(login_frame, text="Tailor Management System",
        font=("Helvetica", 14))
 subtitle.pack(pady=(5,30))
 username_entry = ctk.CTkEntry(login_frame, width=320, height=45,
          placeholder_text="Username",
          font=("Helvetica", 13))
 username_entry.pack(pady=10)
 password_entry = ctk.CTkEntry(login_frame, width=320, height=45,
          placeholder_text="Password",
          show="●",
          font=("Helvetica", 13))
 password_entry.pack(pady=10)
#<------------------------------------------------------------>
 def check_login():
  global logged_in_user
  username = username_entry.get()
  password = password_entry.get()
  if not username or not password:
   messagebox.showwarning("Error", "Enter username and password")
   return
  user = fetch_one("SELECT * FROM users WHERE username = %s AND password = %s",
       (username, password))
  if user:
   logged_in_user = username
   show_main_page()
  else:
   messagebox.showerror("Error", "Wrong username or password!")
 login_btn = ctk.CTkButton(login_frame, text="Login",
          width=320, height=50,
          font=("Helvetica", 16, "bold"),
          command=check_login)
 login_btn.pack(pady=30)
 password_entry.bind('<Return>', lambda e: check_login())
#<------------------------------------------------------------>
def show_main_page():
 global window, main_area
 clear_screen()
 sidebar = ctk.CTkFrame(window, width=280, corner_radius=0, fg_color="#242424")
 sidebar.pack(side="left", fill="y")
 sidebar.pack_propagate(False)
 logo = ctk.CTkLabel(sidebar, text="StitchSync",
       font=("Helvetica", 26, "bold"))
 logo.pack(pady=15)
 user_label = ctk.CTkLabel(sidebar, text=f"Logged in as: {logged_in_user}",
          font=("Helvetica", 13, "bold"))
 user_label.pack(pady=10)
 dashboard_btn = ctk.CTkButton(sidebar, text="📊 Dashboard",
          width=240, height=50,
          command=show_dashboard_page)
 dashboard_btn.pack(pady=8, padx=20)
 new_order_btn = ctk.CTkButton(sidebar, text="➕ New Order",
          width=240, height=50,
          command=show_new_order_page)
```

```
 all_orders_btn = ctk.CTkButton(sidebar, text="📋 All Orders",
          width=240, height=50,
          command=show_all_orders_page)
 all_orders_btn.pack(pady=8, padx=20)
 customers_btn = ctk.CTkButton(sidebar, text="👥 Customers",
          width=240, height=50,
          command=show_customers_page)
 customers_btn.pack(pady=8, padx=20)
 settings_btn = ctk.CTkButton(sidebar, text="⚙️ Settings",
          width=240, height=50,
          command=show_settings_page)
 settings_btn.pack(pady=8, padx=20)
 logout_btn = ctk.CTkButton(sidebar, text="'🚪 Logout",
         width=240, height=50,
         fg_color="#c42b1c",
         command=show_login_page)
 logout_btn.pack(side="bottom", pady=20, padx=20)
 main_area = ctk.CTkFrame(window)
 main_area.pack(side="right", fill="both", expand=True)
 show_dashboard_page()
#<------------------------------------------------------------>
def show_dashboard_page():
 global main_area
 for widget in main_area.winfo_children():
  widget.destroy()
 title = ctk.CTkLabel(main_area, text="📊 Dashboard",
       font=("Helvetica", 32, "bold"))
 title.pack(pady=20, padx=30, anchor="w")
 stats_frame = ctk.CTkFrame(main_area)
 stats_frame.pack(fill="x", padx=30, pady=20)
 total_orders = fetch_one("SELECT COUNT(*) as count FROM orders")['count']
 pending_orders = fetch_one("SELECT COUNT(*) as count FROM orders WHERE status =
'Pending'")['count']
 current_month = datetime.now().strftime("%Y-%m")
 revenue_data = fetch_one(
  "SELECT SUM(price) as total FROM orders WHERE DATE_FORMAT(created_at, '%%Y-
%%m') = %s",
  (current_month,)
 )
 revenue = revenue_data['total'] if revenue_data['total'] else 0
 stats = [
  ("📦 Total Orders", total_orders, "#3b8ed0"),
  ("⏳ Pending", pending_orders, "#ca5010"),
  ("💰 Revenue", f"₹{revenue:.2f}", "#107c10")
 ]
 for i, (label, value, color) in enumerate(stats):
  card = ctk.CTkFrame(stats_frame, fg_color=color, corner_radius=15)
  card.grid(row=0, column=i, padx=10, sticky="ew")
  stats_frame.grid_columnconfigure(i, weight=1)
  value_label = ctk.CTkLabel(card, text=str(value),
          font=("Helvetica", 36, "bold"))
  value_label.pack(pady=(25, 5))
  text_label = ctk.CTkLabel(card, text=label,
          font=("Helvetica", 15))
```

```
    text_label.pack(pady=(0, 25))
    chart_frame = ctk.CTkFrame(main_area)
    chart_frame.pack(fill="both", expand=True, padx=30, pady=20)
    fig = Figure(figsize=(12, 5), facecolor='#2b2b2b')
    garment_data = fetch_data(
      "SELECT garment_type, COUNT(*) as count FROM orders GROUP BY garment_type"
    )
    if garment_data:
      ax1 = fig.add_subplot(121)
      labels = [row['garment_type'] for row in garment_data]
      sizes = [row['count'] for row in garment_data]
      colors = ['#3b8ed0', '#ca5010', '#107c10', '#8764b8', '#00bcf2']
      ax1.pie(sizes, labels=labels, autopct='%1.1f%%', colors=colors, textprops={'color':
'white'})
      ax1.set_title('Orders by Garment', color='white', fontsize=15)
      status_data = fetch_data(
        "SELECT status, COUNT(*) as count FROM orders GROUP BY status"
      )
      ax2 = fig.add_subplot(122)
      status_labels = [row['status'] for row in status_data]
      status_counts = [row['count'] for row in status_data]
      ax2.bar(status_labels, status_counts, color=colors)
      ax2.set_title('Orders by Status', color='white', fontsize=15)
      ax2.set_facecolor('#2b2b2b')
      ax2.tick_params(colors='white')
    canvas = FigureCanvasTkAgg(fig, chart_frame)
    canvas.draw()
    canvas.get_tk_widget().pack(fill="both", expand=True, padx=20, pady=20)
#<------------------------------------------------------------>
def show_new_order_page():
    global main_area, form_fields
    form_fields = {}
    for widget in main_area.winfo_children():
      widget.destroy()
    title = ctk.CTkLabel(main_area, text="+ New Order",
          font=("Helvetica", 32, "bold"))
    title.pack(pady=20, padx=30, anchor="w")
    form_frame = ctk.CTkScrollableFrame(main_area)
    form_frame.pack(fill="both", expand=True, padx=30, pady=20)
    rows_frame = ctk.CTkFrame(form_frame)
    rows_frame.pack(fill="both", expand=True, padx=10, pady=10)
    rows_frame.grid_columnconfigure(0, weight=0, minsize=220)
    rows_frame.grid_columnconfigure(1, weight=1)
    row_idx = 0
#<------------------------------------------------------------>
    def add_row(label_text, widget, row_padding_y=(5,5)):
      nonlocal row_idx
      lbl = ctk.CTkLabel(rows_frame, text=label_text, font=("Helvetica", 13))
      lbl.grid(row=row_idx, column=0, sticky="w", padx=(5,15), pady=row_padding_y)
      widget.grid(row=row_idx, column=1, sticky="ew", padx=(5,10), pady=row_padding_y)
      row_idx += 1
      return lbl
```

```
#<------------------------------------------------------------>
 def search_customers(event):
  typed = form_fields['name'].get()
  for widget in suggestions_frame.winfo_children():
   widget.destroy()
  suggestions_frame.place_forget()
  if len(typed) >= 2:
   customers = fetch_data(
    "SELECT * FROM customers WHERE LOWER(name) LIKE %s LIMIT 5",
    (f"%{typed.lower()}%",)
   )
   if customers:
    form_fields['name'].update_idletasks()
    entry_width = form_fields['name'].winfo_width() - 240
    suggestions_frame.configure(corner_radius=8, width=entry_width)
    suggestions_frame.place(in_=form_fields['name'], relx=0, rely=1, x=0, y=4,
anchor='nw')
    suggestions_frame.lift()
    for customer in customers:
#<------------------------------------------------------------>
      def fill_customer(c=customer):
       form_fields['name'].delete(0, 'end')
       form_fields['name'].insert(0, c['name'])
       form_fields['contact'].delete(0, 'end')
       form_fields['contact'].insert(0, c['contact'])
       suggestions_frame.place_forget()
      btn = ctk.CTkButton(suggestions_frame,
            text=f"{customer['name']} - {customer['contact']}",
            width=entry_width, height=35,
            anchor="w",
            command=fill_customer)
      btn.pack(fill="x", pady=2, padx=2)
 form_fields['name'].bind('<KeyRelease>', search_customers)
 form_fields['contact'] = ctk.CTkEntry(rows_frame, height=40)
 add_row("Contact Number *", form_fields['contact'])
 form_fields['garment_type'] = ctk.CTkComboBox(
  rows_frame, height=40, values=["Shirt", "Kurta", "Blazer", "Trouser", "Saree Blouse"]
 )
 add_row("Garment Type *", form_fields['garment_type'])
 form_fields['fabric'] = ctk.CTkComboBox(
  rows_frame, height=40, values=["Cotton", "Silk", "Linen", "Synthetic"]
 )
 add_row("Fabric Type *", form_fields['fabric'])
 hdr = ctk.CTkLabel(rows_frame, text="📏 Measurements (inches)", font=("Helvetica", 18,
"bold"))
 hdr.grid(row=row_idx, column=0, columnspan=2, sticky="w", pady=(15,8), padx=5)
 row_idx += 1
 measurements_list = ["Chest", "Waist", "Hip", "Length", "Sleeve Length"]
 form_fields['measurements'] = {}
 for measure in measurements_list:
  ent = ctk.CTkEntry(rows_frame, height=40)
  key = measure.lower().replace(" ", "_")
  form_fields['measurements'][key] = ent
  add_row(measure, ent)
```

```
 form_fields['collar'] = ctk.CTkComboBox(
  rows_frame, height=40, values=["Regular", "Mandarin", "Spread", "Button-Down",
"None"]
 )
 add_row("Collar Type", form_fields['collar'])
 form_fields['sleeve'] = ctk.CTkComboBox(
  rows_frame, height=40, values=["Full", "Half", "Three-Quarter", "Sleeveless"]
 )
 add_row("Sleeve Type", form_fields['sleeve'])
 form_fields['fit'] = ctk.CTkComboBox(
  rows_frame, height=40, values=["Slim Fit", "Regular Fit", "Loose Fit", "Custom"]
 )
 add_row("Fit Type", form_fields['fit'])
 form_fields['delivery_date'] = ctk.CTkEntry(rows_frame, height=40,
placeholder_text="YYYY-MM-DD")
 add_row("Delivery Date *", form_fields['delivery_date'])
 notes_label = ctk.CTkLabel(rows_frame, text="Additional Notes", font=("Helvetica", 13))
 notes_label.grid(row=row_idx, column=0, sticky="nw", padx=(5,15), pady=5)
 form_fields['notes'] = ctk.CTkTextbox(rows_frame, height=100)
 form_fields['notes'].grid(row=row_idx, column=1, sticky="ew", padx=(5,10), pady=5)
 row_idx += 1
 form_fields['price'] = ctk.CTkEntry(rows_frame, height=40)
 add_row("Base Price (₹) *", form_fields['price'])
 button_frame = ctk.CTkFrame(rows_frame,fg_color='#2B2B2B')
 button_frame.grid(row=row_idx, column=0, columnspan=2, pady=20,)
 row_idx += 1
 save_btn = ctk.CTkButton(button_frame, text="💾 Save Order", width=200, height=50,
command=save_order)
 save_btn.pack(side="left", padx=10)
 jobcard_btn = ctk.CTkButton(button_frame, text="📄 Job Card", width=200, height=50,
command=create_jobcard)
 jobcard_btn.pack(side="left", padx=10)
 invoice_btn = ctk.CTkButton(button_frame, text="📄 Invoice", width=200, height=50,
command=create_invoice)
 invoice_btn.pack(side="left", padx=10)
#<------------------------------------------------------------>
def save_order():
 global form_fields
 try:
  if not form_fields['name'].get():
   messagebox.showerror("Error", "Enter customer name")
   return
  if not form_fields['contact'].get():
   messagebox.showerror("Error", "Enter contact number")
   return
  if not form_fields['price'].get():
   messagebox.showerror("Error", "Enter price")
   return
  order_count = fetch_one("SELECT COUNT(*) as count FROM orders")['count']
  order_id = f"ORD{order_count + 1:05d}"
  measurements = {}
  for key, entry in form_fields['measurements'].items():
```

```
    existing = fetch_one("SELECT * FROM customers WHERE contact = %s",
          (form_fields['contact'].get(),))
    if not existing:
     run_query("INSERT INTO customers (name, contact) VALUES (%s, %s)",
        (form_fields['name'].get(), form_fields['contact'].get()))
    query = """INSERT INTO orders (order_id, customer_name, contact, garment_type,
       fabric, measurements, collar_type, sleeve_type, fit_type,
       delivery_date, notes, price, status)
       VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, 'Pending')"""
    values = (
     order_id,
     form_fields['name'].get(),
     form_fields['contact'].get(),
     form_fields['garment_type'].get(),
     form_fields['fabric'].get(),
     measurements_json,
     form_fields['collar'].get(),
     form_fields['sleeve'].get(),
     form_fields['fit'].get(),
     form_fields['delivery_date'].get(),
     form_fields['notes'].get("1.0", "end-1c"),
     float(form_fields['price'].get())
    )
    run_query(query, values)
    messagebox.showinfo("Success", f"Order {order_id} saved!")
    show_new_order_page()
  except Exception as e:
    messagebox.showerror("Error", f"Failed: {str(e)}")
#<------------------------------------------------------------>
def create_jobcard():
 global form_fields
 try:
   if not form_fields['name'].get():
     messagebox.showerror("Error", "Fill customer details")
     return
   filename =
f"JobCard_{form_fields['name'].get()}_{datetime.now().strftime('%Y%m%d_%H%M%S')}.p
df"
   pdf = canvas.Canvas(filename, pagesize=letter)
   width, height = letter
   settings = fetch_one("SELECT * FROM settings WHERE id = 1")
   pdf.setFont("Helvetica-Bold", 20)
   pdf.drawString(50, height - 50, "JOB CARD")
   pdf.setFont("Helvetica", 10)
   pdf.drawString(50, height - 70, settings['shop_name'])
   pdf.drawString(50, height - 85, settings['address'])
   pdf.drawString(50, height - 100, settings['phone'])
   pdf.drawString(50, height - 115, 'Job card generated by')
   pdf.drawString(150, height - 115, logged_in_user)
   y = height - 150
   pdf.setFont("Helvetica-Bold", 12)
   pdf.drawString(50, y, "Customer Details:")
   y -= 25
   pdf.setFont("Helvetica", 10)
```

```
    pdf.drawString(50, y, f"Name: {form_fields['name'].get()}")
    y -= 20
    pdf.drawString(50, y, f"Contact: {form_fields['contact'].get()}")
    y -= 40
    pdf.setFont("Helvetica-Bold", 12)
    pdf.drawString(50, y, "Garment Details:")
    y -= 25
    pdf.setFont("Helvetica", 10)
    pdf.drawString(50, y, f"Type: {form_fields['garment_type'].get()}")
    y -= 20
    pdf.drawString(50, y, f"Fabric: {form_fields['fabric'].get()}")
    y -= 40
    pdf.setFont("Helvetica-Bold", 12)
    pdf.drawString(50, y, "Measurements:")
    y -= 25
    pdf.setFont("Helvetica", 10)
    for key, entry in form_fields['measurements'].items():
      value = entry.get() if entry.get() else "0.0"
      pdf.drawString(50, y, f"{key.replace('_', ' ').title()}: {value} inches")
      y -= 20
    pdf.save()
    messagebox.showinfo("Success", f"Job Card saved: {filename}")
  except Exception as e:
    messagebox.showerror("Error", f"Failed: {str(e)}")
#<------------------------------------------------------------>
def create_invoice():
  global form_fields
  try:
    if not form_fields['name'].get() or not form_fields['price'].get():
      messagebox.showerror("Error", "Fill all required fields")
      return
    filename =
f"Invoice_{form_fields['name'].get()}_{datetime.now().strftime('%Y%m%d_%H%M%S')}.p
df"
    pdf = canvas.Canvas(filename, pagesize=letter)
    width, height = letter
    settings = fetch_one("SELECT * FROM settings WHERE id = 1")
    pdf.setFont("Helvetica-Bold", 24)
    pdf.drawString(50, height - 50, "INVOICE")
    pdf.setFont("Helvetica", 10)
    pdf.drawString(50, height - 75, settings['shop_name'])
    pdf.drawString(50, height - 90, settings['address'])
    order_count = fetch_one("SELECT COUNT(*) as count FROM orders")['count']
    pdf.drawString(400, height - 75, f"Date: {datetime.now().strftime('%Y-%m-%d')}")
    pdf.drawString(400, height - 90, f"Invoice #: INV{order_count + 1:04d}")
    y = height - 150
    pdf.setFont("Helvetica-Bold", 12)
    pdf.drawString(50, y, "Bill To:")
    y -= 20
    pdf.setFont("Helvetica", 10)
    pdf.drawString(50, y, form_fields['name'].get())
    y -= 15
    pdf.drawString(50, y, form_fields['contact'].get())
    y -= 50
    pdf.setFont("Helvetica-Bold", 10)
```

```
    pdf.drawString(50, y, "Description")
    pdf.drawString(300, y, "Qty")
    pdf.drawString(400, y, "Price")
    pdf.drawString(500, y, "Total")
    y -= 5
    pdf.line(50, y, 550, y)
    y -= 25
    pdf.setFont("Helvetica", 10)
    base_price = float(form_fields['price'].get())
    pdf.drawString(50, y, f"{form_fields['garment_type'].get()} -
{form_fields['fabric'].get()}")
    pdf.drawString(320, y, "1")
    pdf.drawString(400, y, f"Rs.{base_price:.2f}")
    pdf.drawString(500, y, f"Rs.{base_price:.2f}")
    y -= 100
    pdf.line(400, y, 550, y)
    y -= 20
    pdf.drawString(400, y, "Subtotal:")
    pdf.drawString(500, y, f"Rs.{base_price:.2f}")
    y -= 20
    tax_amount = base_price * (float(settings['tax_rate']) / 100)
    pdf.drawString(400, y, f"Tax ({settings['tax_rate']}%):")
    pdf.drawString(500, y, f"Rs.{tax_amount:.2f}")
    y -= 20
    pdf.line(400, y, 550, y)
    y -= 20
    pdf.setFont("Helvetica-Bold", 12)
    total = base_price + tax_amount
    pdf.drawString(400, y, "Total:")
    pdf.drawString(500, y, f"Rs.{total:.2f}")
    pdf.setFont("Helvetica-Italic", 9)
    pdf.drawString(50, 50, "Thank you for your business!")
    pdf.save()
    messagebox.showinfo("Success", f"Invoice saved: {filename}")
  except Exception as e:
    messagebox.showerror("Error", f"Failed: {str(e)}")
#<------------------------------------------------------------>
def show_all_orders_page():
  global main_area
  for widget in main_area.winfo_children():
    widget.destroy()
  title = ctk.CTkLabel(main_area, text="📋 All Orders",
          font=("Helvetica", 32, "bold"))
  title.pack(pady=20, padx=30, anchor="w")
  filter_frame = ctk.CTkFrame(main_area)
  filter_frame.pack(fill="x", padx=30, pady=10)
  ctk.CTkLabel(filter_frame, text="Filter by Status:",
       font=("Helvetica", 14)).pack(side="left", padx=10)
#<------------------------------------------------------------>
  def load_orders(status):
    for widget in orders_frame.winfo_children():
      widget.destroy()
    if status == "All":
```

```
      "SELECT * FROM orders WHERE status = %s ORDER BY created_at DESC",
      (status,)
     )
    if not orders:
     no_orders = ctk.CTkLabel(orders_frame, text="No orders found",
           font=("Helvetica", 16))
     no_orders.pack(pady=50)
     return
    for order in orders:
     order_card = ctk.CTkFrame(orders_frame, corner_radius=10)
     order_card.pack(fill="x", pady=8)
     info_frame = ctk.CTkFrame(order_card,fg_color='#2B2B2B')
     info_frame.pack(fill="x", padx=20, pady=15)
     left_frame = ctk.CTkFrame(info_frame)
     left_frame.pack(side="left", fill="x", expand=True)
     ctk.CTkLabel(left_frame, text=f"📦 {order['order_id']}",
        font=("Helvetica", 16, "bold")).pack(anchor="w")
     ctk.CTkLabel(left_frame, text=f"Customer: {order['customer_name']}",
        font=("Helvetica", 12)).pack(anchor="w", pady=3)
     ctk.CTkLabel(left_frame, text=f"Item: {order['garment_type']}",
        font=("Helvetica", 11)).pack(anchor="w")
     middle_frame = ctk.CTkFrame(info_frame)
     middle_frame.pack(side="left", padx=30)
     ctk.CTkLabel(middle_frame, text=f"Delivery: {order['delivery_date'],}",
        font=("Helvetica", 11)).pack(anchor="w")
     ctk.CTkLabel(middle_frame, text=f"Price: ₹{order['price']:.2f}",
        font=("Helvetica", 13, "bold")).pack(anchor="w", pady=5)
     right_frame = ctk.CTkFrame(info_frame)
     right_frame.pack(side="right")
     status_colors = {
      "Pending": "#ca5010",
      "In Progress": "#0078d4",
      "Ready": "#107c10",
      "Delivered": "#8764b8"
     }
#<------------------------------------------------------------>
     def change_status(o=order):
      dialog = ctk.CTkToplevel(window)
      dialog.title("Change Status")
      dialog.geometry("350x300")
      dialog.transient(window)
      dialog.grab_set()
      ctk.CTkLabel(dialog, text=f"Order: {o['order_id']}",
         font=("Helvetica", 18, "bold")).pack(pady=20)
      ctk.CTkLabel(dialog, text="Select new status:",
         font=("Helvetica", 14)).pack(pady=10)
      status_var = ctk.StringVar(value=o['status'])
      for s in ["Pending", "In Progress", "Ready", "Delivered"]:
       ctk.CTkRadioButton(dialog, text=s,
            variable=status_var,
            value=s).pack(pady=5)
#<------------------------------------------------------------>
      def save():
```

```
            (status_var.get(), o['id']))
        dialog.destroy()
        show_all_orders_page()
      ctk.CTkButton(dialog, text="Save", width=200, height=45,
          command=save).pack(pady=20)
    status_btn = ctk.CTkButton(right_frame, text=order['status'],
            width=120, height=35,
            fg_color=status_colors.get(order['status'], "#0078d4"),
            command=change_status)
    status_btn.pack(pady=(0, 10))
    actions_frame = ctk.CTkFrame(right_frame)
    actions_frame.pack()
#<------------------------------------------------------------>
    def view_order(o=order):
      dialog = ctk.CTkToplevel(window)
      dialog.title(f"Order Details - {o['order_id']}")
      dialog.geometry("700x750")
      dialog.transient(window)
      header = ctk.CTkFrame(dialog, height=80)
      header.pack(fill="x")
      header.pack_propagate(False)
      ctk.CTkLabel(header, text=f"📦 {o['order_id']}",
          font=("Helvetica", 24, "bold")).pack(pady=25)
      scroll = ctk.CTkScrollableFrame(dialog)
      scroll.pack(fill="both", expand=True, padx=20, pady=20)
      details = [
        ("Customer", o['customer_name']),
        ("Contact", o['contact']),
        ("Garment", o['garment_type']),
        ("Fabric", o['fabric']),
        ("Status", o['status']),
        ("Delivery", str(o['delivery_date'])),
        ("Price", f"₹{o['price']:.2f}"),
      ]
      for label, value in details:
        frame = ctk.CTkFrame(scroll)
        frame.pack(fill="x", pady=5)
        ctk.CTkLabel(frame, text=f"{label}:", width=150,
            anchor="w", font=("Helvetica", 12, "bold")).pack(side="left", padx=10)
        ctk.CTkLabel(frame, text=str(value), anchor="w").pack(side="left")
      ctk.CTkLabel(scroll, text="Measurements:",
          font=("Helvetica", 16, "bold")).pack(anchor="w", pady=10)
      measurements = json.loads(o['measurements'])
      for key, value in measurements.items():
        frame = ctk.CTkFrame(scroll)
        frame.pack(fill="x", pady=3)
        ctk.CTkLabel(frame, text=f"{key.replace('_', ' ').title()}:",
            width=150, anchor="w").pack(side="left", padx=10)
        ctk.CTkLabel(frame, text=f"{value} inches").pack(side="left")
#<------------------------------------------------------------>
    def delete_order(o=order):
      result = messagebox.askyesno("Delete", f"Delete order {o['order_id']}?")
```

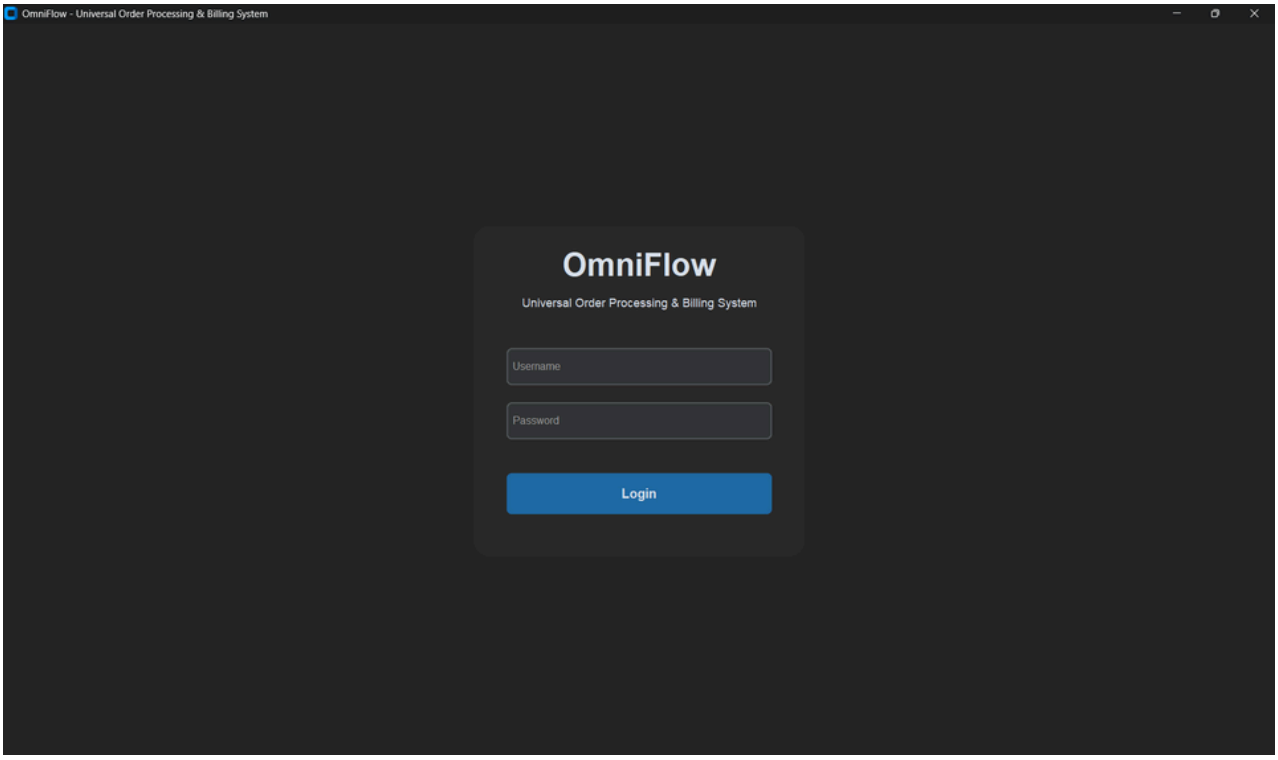
```
            messagebox.showinfo("Success", "Order deleted")
            show_all_orders_page()
        view_btn = ctk.CTkButton(actions_frame, text="👁️",
                 width=40, height=35,
                 command=view_order)
        view_btn.pack(side="left", padx=2)
        delete_btn = ctk.CTkButton(actions_frame, text="🗑️",
                 width=40, height=35,
                 fg_color="#c42b1c",
                 command=delete_order)
        delete_btn.pack(side="left", padx=2)
    status_filter = ctk.CTkComboBox(
     filter_frame, width=200,
     values=["All", "Pending", "In Progress", "Ready", "Delivered"],
     command=load_orders
    )
    status_filter.set("All")
    status_filter.pack(side="left", padx=10)
    orders_frame = ctk.CTkScrollableFrame(main_area)
    orders_frame.pack(fill="both", expand=True, padx=30, pady=20)
    load_orders("All")
#<------------------------------------------------------------>
def show_customers_page():
    global main_area
    for widget in main_area.winfo_children():
        widget.destroy()
    title = ctk.CTkLabel(main_area, text="👥 Customers",
            font=("Helvetica", 32, "bold"))
    title.pack(pady=20, padx=30, anchor="w")
    customer_count = fetch_one("SELECT COUNT(*) as count FROM customers")['count']
    stats_card = ctk.CTkFrame(main_area, corner_radius=15, fg_color='#2B2B2B')
    stats_card.pack(fill="x", padx=30, pady=10)
    ctk.CTkLabel(stats_card, text=str(customer_count),
        font=("Helvetica", 28, "bold")).pack(pady=(20, 5))
    ctk.CTkLabel(stats_card, text="Total Customers",
        font=("Helvetica", 16)).pack(pady=(0, 20))
    customers_frame = ctk.CTkScrollableFrame(main_area)
    customers_frame.pack(fill="both", expand=True, padx=30, pady=20)
    customers = fetch_data("SELECT * FROM customers ORDER BY created_at DESC")
    if not customers:
        ctk.CTkLabel(customers_frame, text="No customers found",
            font=("Helvetica", 16)).pack(pady=50)
        return
    for customer in customers:
        order_count = fetch_one(
            "SELECT COUNT(*) as count FROM orders WHERE contact = %s",
            (customer['contact'],)
        )['count']
        total_spent = fetch_one(
            "SELECT SUM(price) as total FROM orders WHERE contact = %s",
            (customer['contact'],)
        )
        spent = total_spent['total'] if total_spent['total'] else 0
```

```
    customer_card.pack(fill="x", pady=8)
    info_frame = ctk.CTkFrame(customer_card, fg_color='#2B2B2B')
    info_frame.pack(fill="x", padx=20, pady=15)
    left_frame = ctk.CTkFrame(info_frame)
    left_frame.pack(side="left", fill="x", expand=True)
    ctk.CTkLabel(left_frame, text=f"👤 {customer['name']}",
        font=("Helvetica", 16, "bold")).pack(anchor="w")
    ctk.CTkLabel(left_frame, text=f"📞 {customer['contact']}",
        font=("Helvetica", 12)).pack(anchor="w", pady=3)
    right_frame = ctk.CTkFrame(info_frame)
    right_frame.pack(side="right", padx=20)
    ctk.CTkLabel(right_frame, text=f"Orders: {order_count}",
        font=("Helvetica", 13)).pack(anchor="e")
    ctk.CTkLabel(right_frame, text=f"Total: ₹{spent:.2f}",
        font=("Helvetica", 14, "bold")).pack(anchor="e", pady=5)
#<------------------------------------------------------------>
def show_settings_page():
  global main_area
  for widget in main_area.winfo_children():
    widget.destroy()
  title = ctk.CTkLabel(main_area, text="⚙ Settings",
         font=("Helvetica", 32, "bold"))
  title.pack(pady=20, padx=30, anchor="w")
  settings_frame = ctk.CTkScrollableFrame(main_area)
  settings_frame.pack(fill="both", expand=True, padx=30, pady=20)
  settings = fetch_one("SELECT * FROM settings WHERE id = 1")
  ctk.CTkLabel(settings_frame, text="Shop Name:",
      font=("Helvetica", 13, "bold")).pack(pady=5, anchor="w")
  shop_name_entry = ctk.CTkEntry(settings_frame, width=400, height=40)
  shop_name_entry.insert(0, settings['shop_name'])
  shop_name_entry.pack(pady=5)
  ctk.CTkLabel(settings_frame, text="Address:",
      font=("Helvetica", 13, "bold")).pack(pady=5, anchor="w")
  address_entry = ctk.CTkEntry(settings_frame, width=400, height=40)
  address_entry.insert(0, settings['address'])
  address_entry.pack(pady=5)
  ctk.CTkLabel(settings_frame, text="Phone:",
      font=("Helvetica", 13, "bold")).pack(pady=5, anchor="w")
  phone_entry = ctk.CTkEntry(settings_frame, width=400, height=40)
  phone_entry.insert(0, settings['phone'])
  phone_entry.pack(pady=5)
  ctk.CTkLabel(settings_frame, text="Tax Rate (%):",
      font=("Helvetica", 13, "bold")).pack(pady=5, anchor="w")
  tax_entry = ctk.CTkEntry(settings_frame, width=400, height=40)
  tax_entry.insert(0, str(settings['tax_rate']))
  tax_entry.pack(pady=5)
#<------------------------------------------------------------>
  def save_settings():
    try:
      query = """UPDATE settings
          SET shop_name = %s, address = %s, phone = %s, tax_rate = %s
          WHERE id = 1"""
      values = (
        shop_name_entry.get(),
```

```
      address_entry.get(),
      phone_entry.get(),
      float(tax_entry.get())
     )
     run_query(query, values)
     messagebox.showinfo("Success", "Settings saved!")
    except Exception as e:
     messagebox.showerror("Error", f"Failed: {str(e)}")
  save_btn = ctk.CTkButton(settings_frame, text="💾 Save Settings",
          width=250, height=50,
          command=save_settings)
  save_btn.pack(pady=30)
#<------------------------------------------------------------>
def main():
  global window, connection, logged_in_user
  window = ctk.CTk()
  window.title("StitchSync - Tailor Management System")
  window.geometry("1400x850")
  if connect_database():
    create_tables()
    show_login_page()
  else:
    messagebox.showerror("Error", "Cannot start application")
    return
#<------------------------------------------------------------>
  def on_close():
    if connection:
      connection.close()
    window.destroy()
  window.protocol("WM_DELETE_WINDOW", on_close)
  window.mainloop()
if __name__ == "__main__":
  main()
```

Output

Login :-

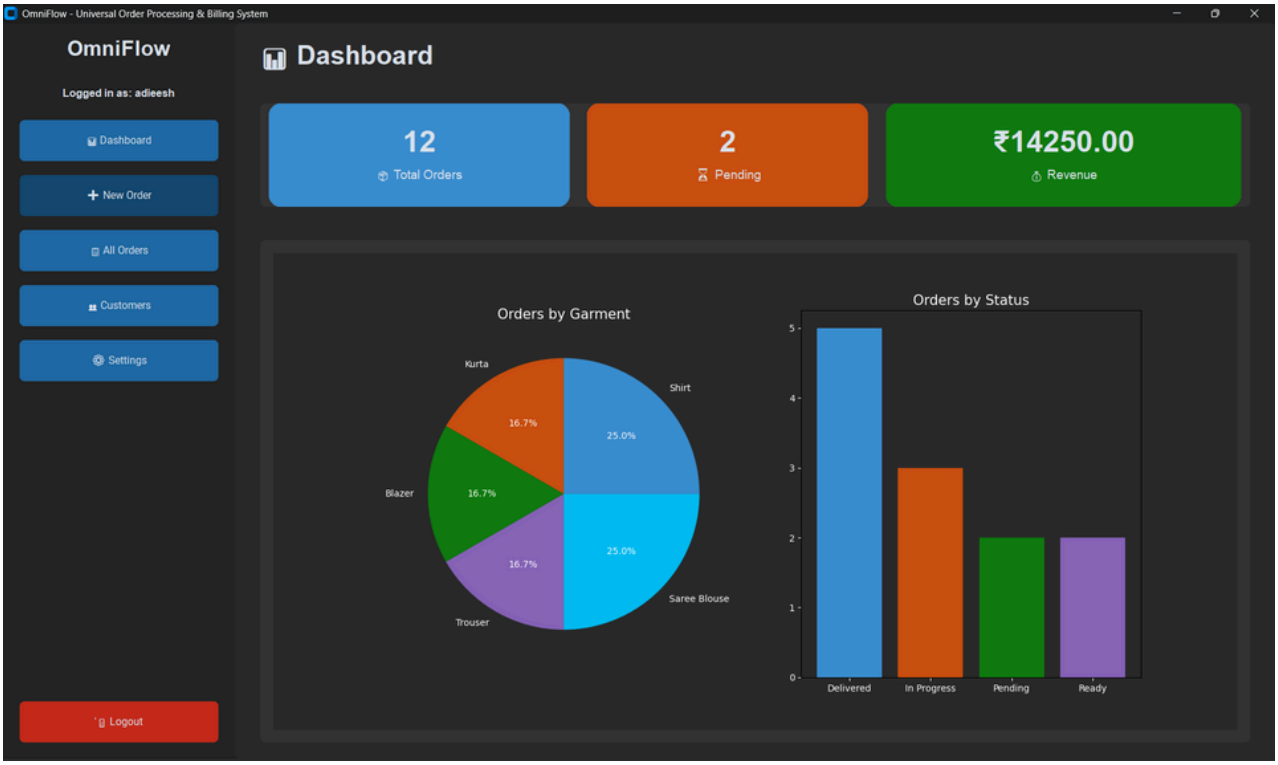


The system provides a secure login panel where each employee is assigned a unique username and password.

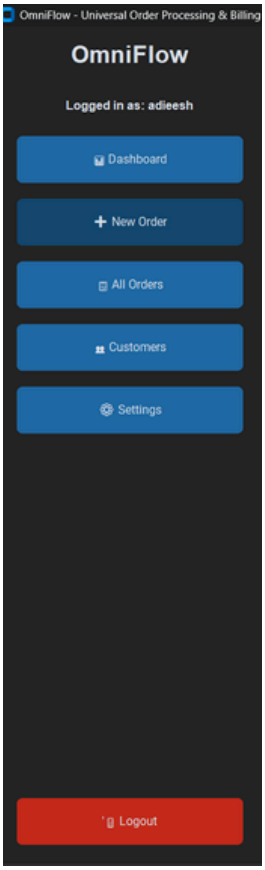
This ensures:

- Personalised access to the application
- Data privacy for customer and order records
- Improved security through controlled user authentication
- Prevention of unauthorized access

Dashboard :-



Navigation side bar :-



The left sidebar provides easy access to all major modules of the application.

Each button is clearly labeled for quick navigation:

- User Display: Shows the name of the currently logged-in user.
- Dashboard: Returns to the main dashboard to view overall statistics.
- New Order: Opens the form to create a new tailoring order.
- All Orders: Allows the user to view, search, and edit existing orders.
- Customers: Displays all saved customer details with options to manage them.
- Settings: Provides controls to edit the shop name, tax percentage, and other configuration options.
- Logout: Logs out the current user and returns to the login screen to switch accounts.

This sidebar ensures smooth navigation and makes the application user-friendly for tailors and employees.

It displays key statistics such as:

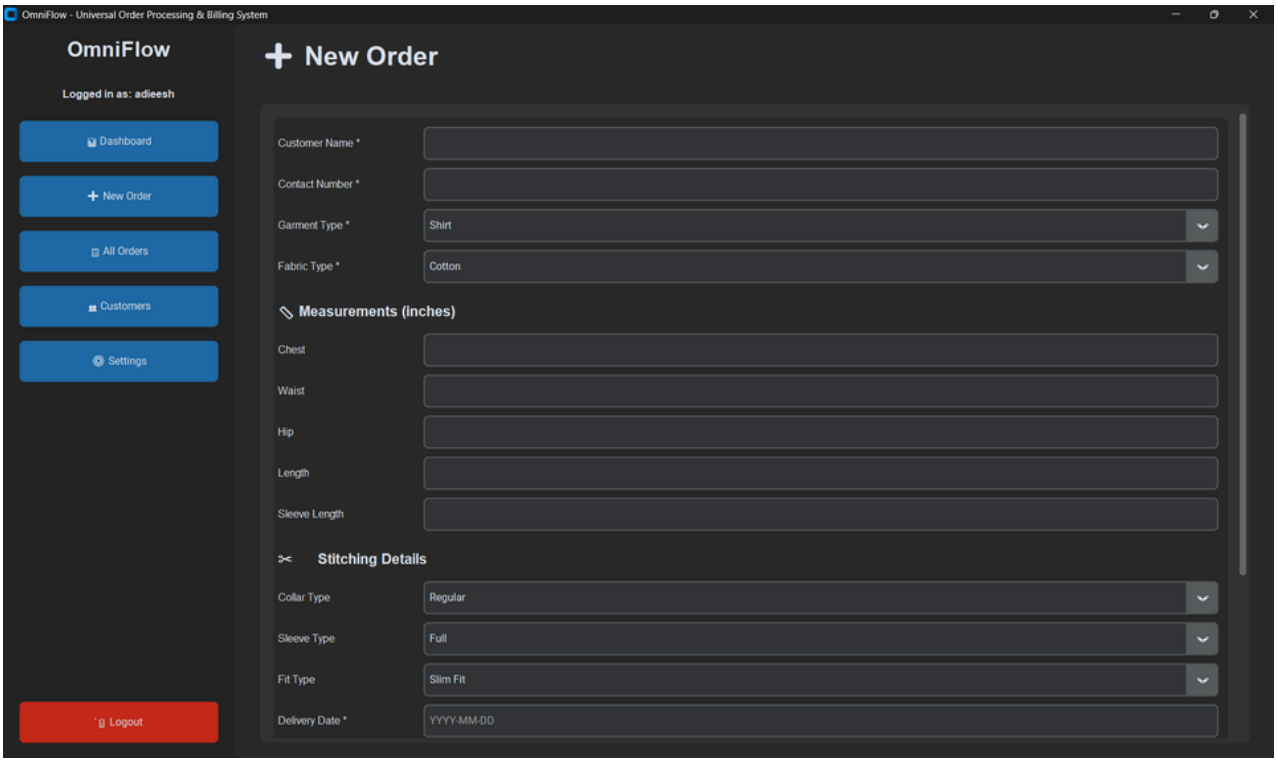
- Total number of orders
- Pending orders
- Total revenue generated

It also includes graphical representations of data:

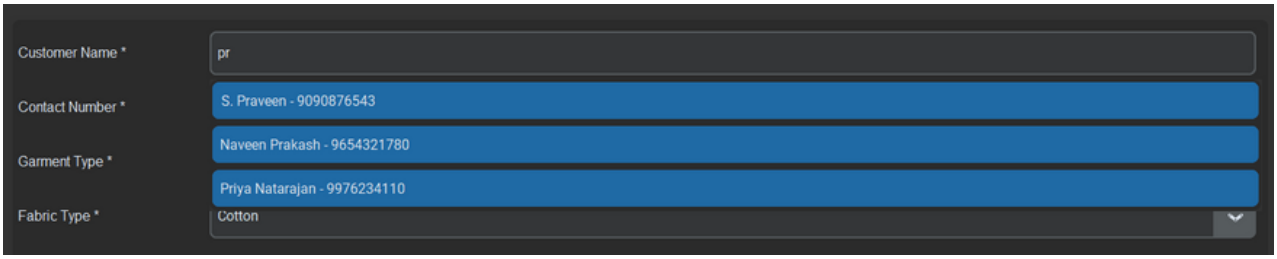
- A Pie Chart showing the distribution of orders by garment type
- A Bar Chart showing the number of orders by their status (Delivered, In Progress, Pending, Ready)

All charts are generated using Matplotlib, and the values are fetched dynamically from the database.
This allows the dashboard to update automatically whenever a new order is added or modified.

New Order :-



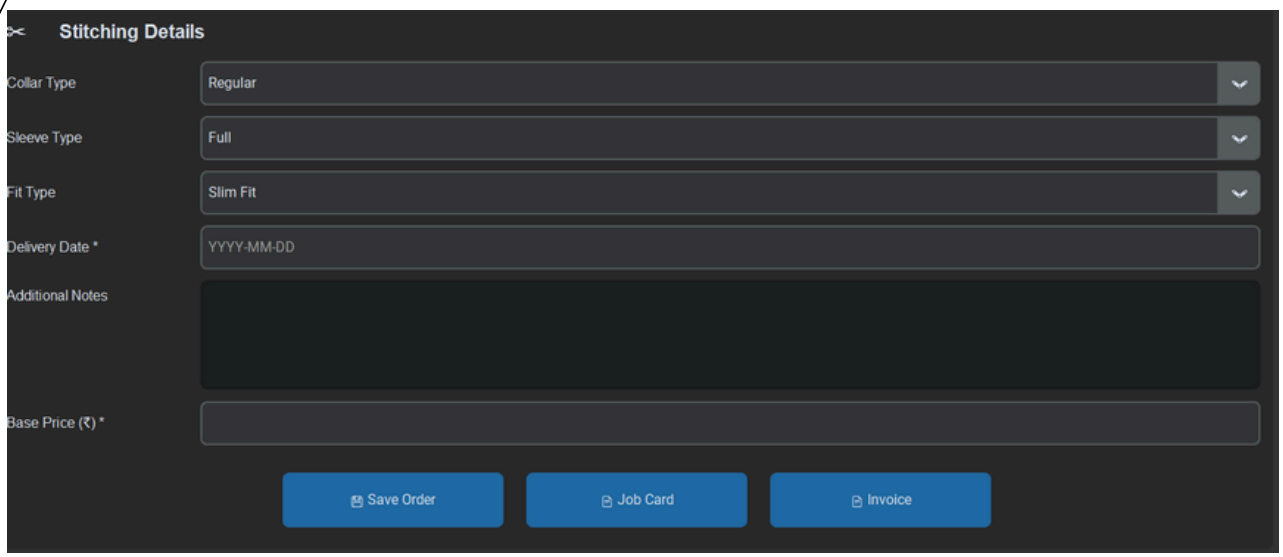
Autofill feature from customers database :-



When the user begins typing a customer name or phone number:

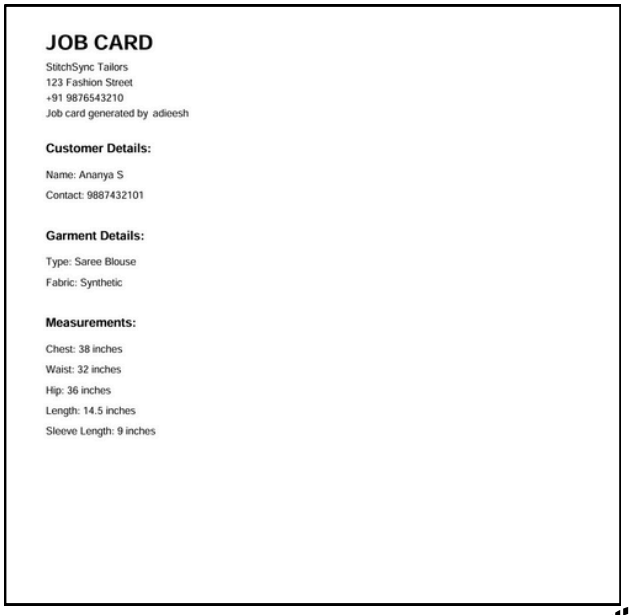
- The system automatically searches the customers database
- Matching entries are displayed in a dropdown list
- On selecting a customer, all related fields (name, contact, previous garment type, measurements, etc.) are automatically filled

The autofill system smartly retrieves past records and reduces repetitive data entry, making the ordering process significantly faster and smoother.



This option saves the entire order into the database

The Job Card feature creates a neatly formatted document
The job card is used internally by tailors during the stitching process. It helps reduce confusion, ensures accuracy, and provides tailors with a quick reference while working.



JOB CARD

StitchSync Tailors
123 Fashion Street
+91 9876543210
Job card generated by adieesh

Customer Details:

Name: Ananya S
Contact: 9887432101

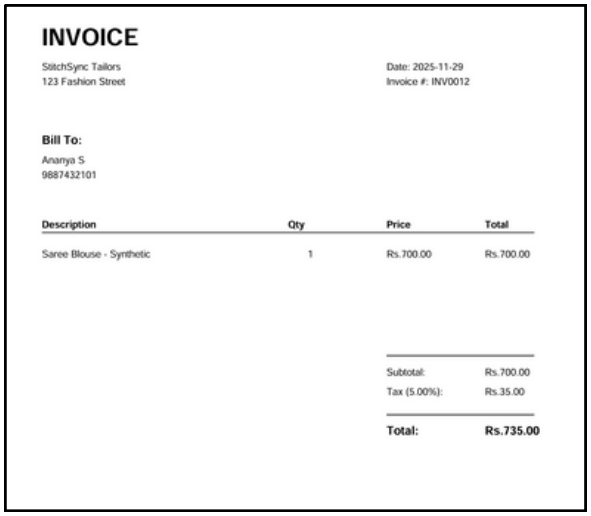
Garment Details:

Type: Saree Blouse
Fabric: Synthetic

Measurements:

Chest: 38 inches
Waist: 32 inches
Hip: 36 inches
Length: 14.5 inches
Sleeve Length: 9 inches

The Invoice button generates a professional invoice for the customer.
This invoice can be saved, printed, or shared, making the billing process fast and reliable.



INVOICE

StitchSync Tailors
123 Fashion Street

Date: 2025-11-29
Invoice #: INV0012

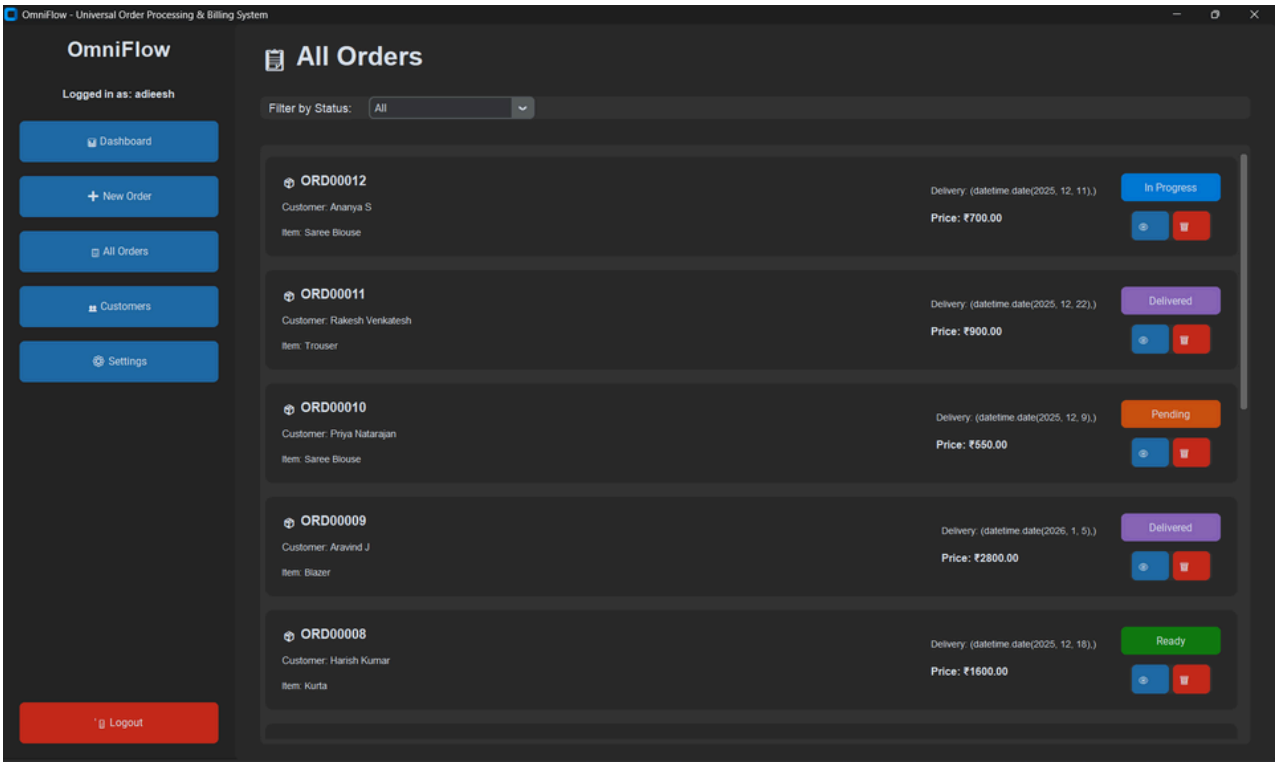
Bill To:

Ananya S
9887432101

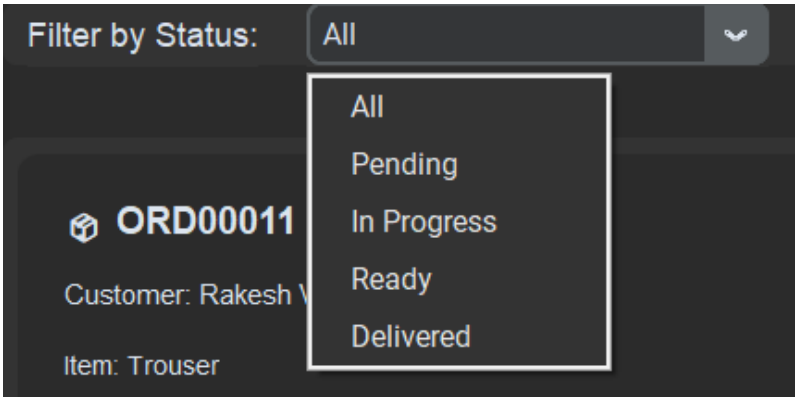
Description	Qty	Price	Total
Saree Blouse - Synthetic	1	Rs.700.00	Rs.700.00

Subtotal: Rs.700.00
Tax (5.00%): Rs.35.00
Total: Rs.735.00

All Orders :-

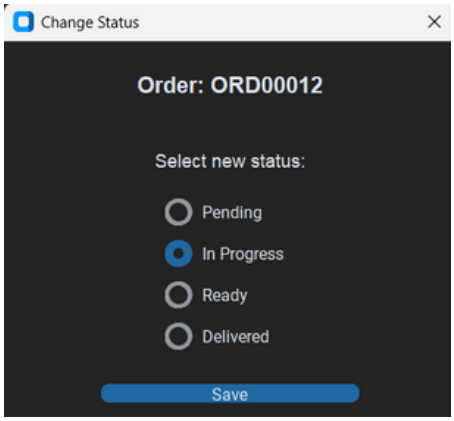


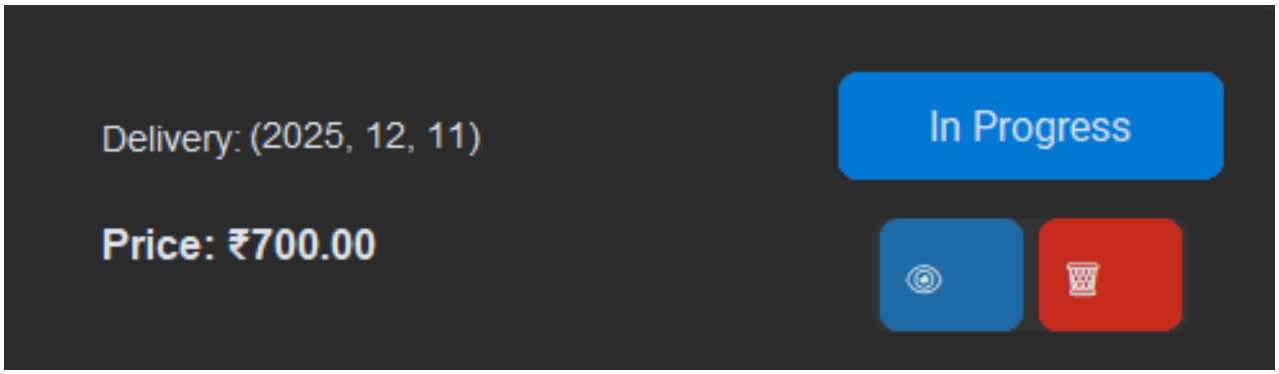
The All Orders section displays a complete list of every order stored in the system.
This page is designed to give the tailor or administrator a quick overview of all ongoing and completed work.

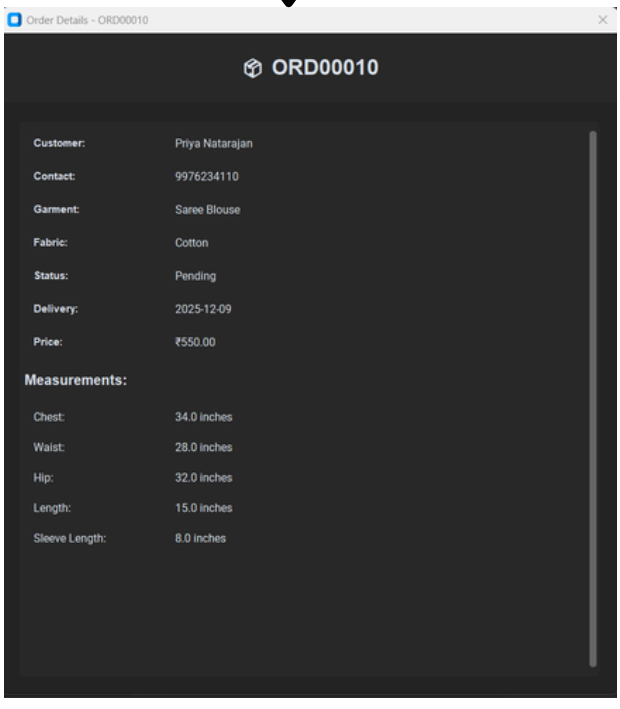


A dropdown filter is provided at the top of the page, allowing the user to view orders based on their current stage in the workflow.
This feature helps the tailoring shop manage workload efficiently by focusing on orders that require immediate attention.

When the user clicks on the status button, a Status Update Window appears.
This window allows the user to easily change the status of the order by selecting one of the available options and clicking Save.
The status update directly reflects on the dashboard charts, helping monitor overall workflow.

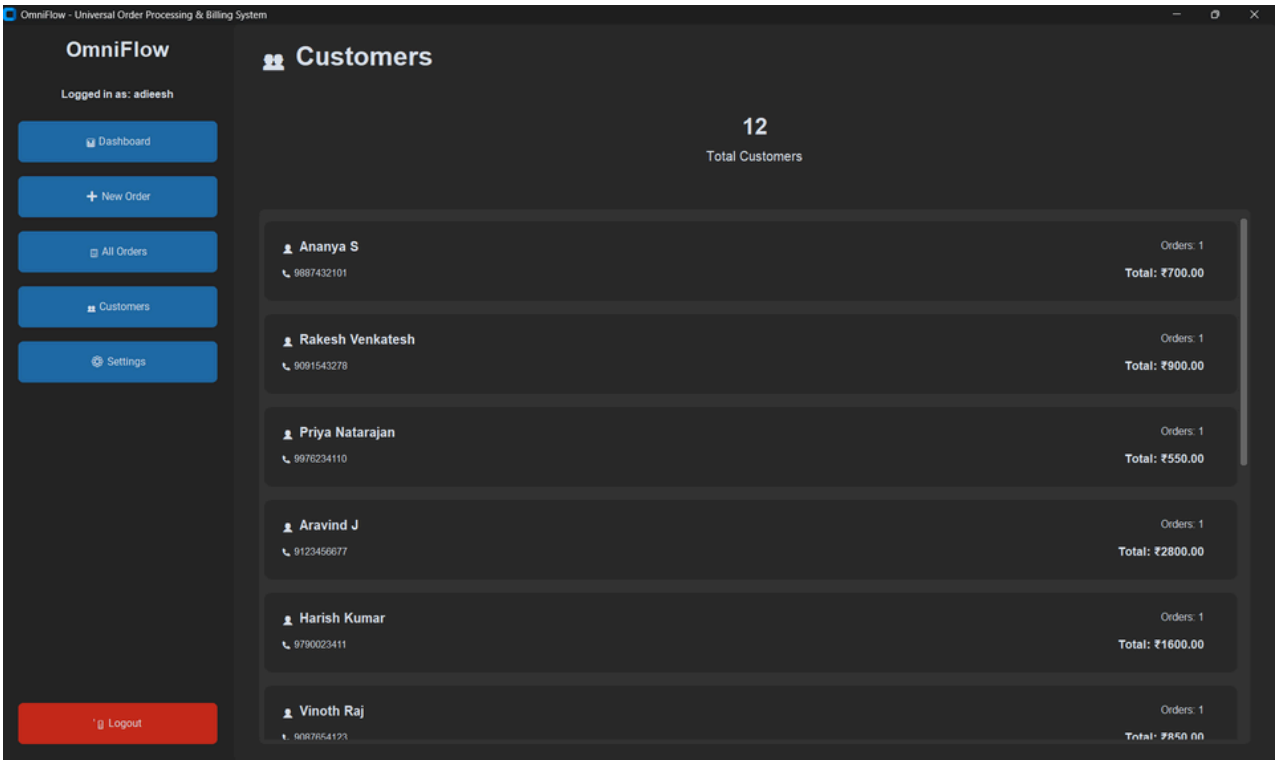






Alongside the status update button, each order card has an eye icon that allows the user to view the full details of that particular order.
This detailed view helps the tailor verify all measurements and instructions before beginning stitching.
It also ensures accuracy, reduces errors, and allows easy reference whenever needed.

Customers :-



The Customers section provides a complete list of all customers stored in the system's database.

This module allows the tailoring shop to easily manage and access customer records, especially useful for repeat customers.

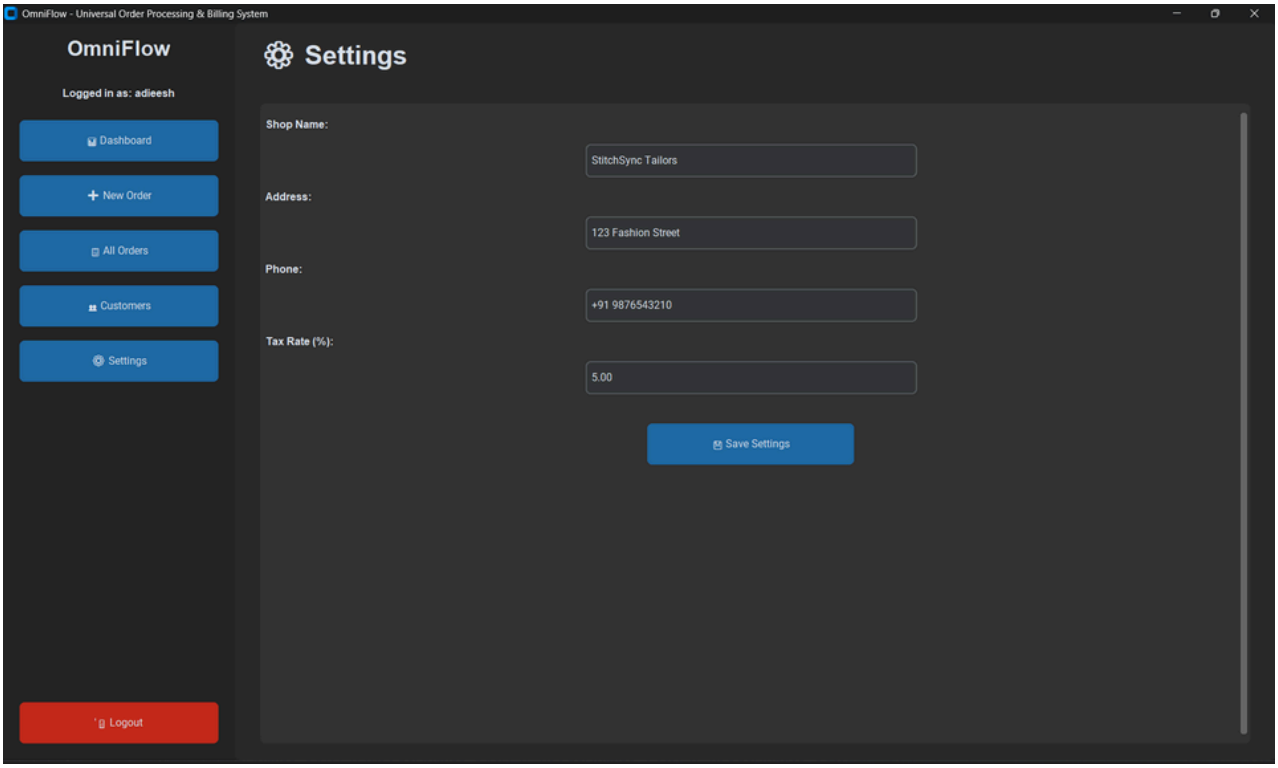
Each customer entry displays:

- Customer Name
- Phone Number
- Number of Orders Placed
- Total Amount Spent

At the top of the page, the system also shows the Total Number of Customers, giving a clear overview of the customer base maintained by the shop.

This module plays an essential role in ensuring that customer data is well-organized, accessible, and efficiently maintained.

Settings :-



The Settings module allows the shop owner or authorized staff to customize and manage the basic configuration of the tailoring system.

This ensures flexibility and personalization based on the shop's requirements.

After updating any information, the user can click Save Settings, which stores the new values in the database and updates them throughout the application.

Conclusion

The development of OmniFlow – Universal Order Processing & Billing System successfully demonstrates how software can simplify and modernize business operations. Throughout the project, various modules such as order creation, customer management, invoice generation, and dashboard analytics were implemented to streamline tasks traditionally performed manually.

By integrating Python, CustomTkinter, MySQL, Matplotlib, and ReportLab, the system offers a seamless experience for both administrators and staff members. Features like autofill customer data, real-time charts, PDF job cards, invoices, and status-based filtering significantly improve efficiency, reduce errors, and enhance overall productivity.

This project has also provided valuable learning in GUI development, database handling, system design, and real-world problem-solving. OmniFlow can be adapted for tailoring shops, repair centers, boutiques, and other order-based businesses, proving its versatility and scalability.

Overall, the project meets its goals of delivering a user-friendly, reliable, and practical business automation system.

Bibilography

- realpython.com
  - “How to Build Desktop Apps with Python”

- Programming with Mosh
  - MySQL for Beginners

- GeeksforGeeks.com
  - “An Introduction to SQL Databases”
  - Python–MySQL Connectivity Guide

- W3Schools.com
  - Matplotlib Data Visualization Basics

- TutorialsPoint.com
  - “Generating PDFs in Python Using ReportLab”

- CustomTkinter Official Documentation
  - https://customtkinter.tomschimansky.com

- Local Tailoring Shop Interaction
  - Observed the typical workflow of order taking, measurement handling, and delivery process to design realistic system features.