**CENTRAL MICHIGAN UNIVERSITY**

**BIS 698 Information System Project**

**Vehicle Service & Inventory Management – Final project documentation**


**GROUP 6**

Richitha Ananthoju

Tharun Dasari

Sindhu Kothuri

Sai Vineeth Neeli

Rohan Chandran Ravi


Submission Date: 4/29/2025

# Table of Contents

## Installation requirements

pip install MySQL-connector-python

pip install tkcalendar

## To start:

Run the **/dist/Login.exe** to start the project.

## Or

Run the **Login.py** to start the project.

## Admin Credentials:

Username: admin

Password: admin123

## Customer Credentials (Like any):

Username: Richi@gmail.com

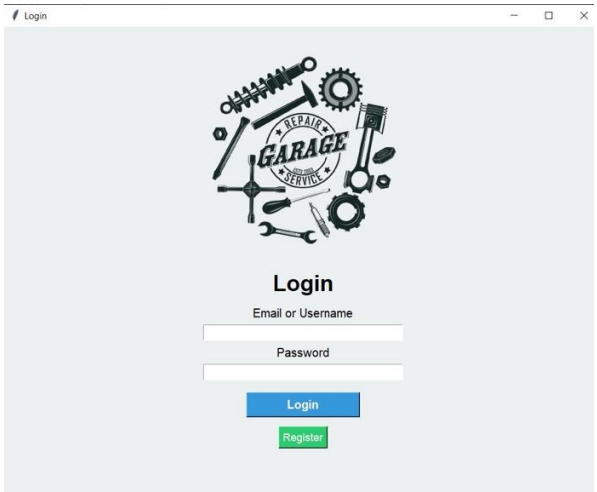
Password: 12345678

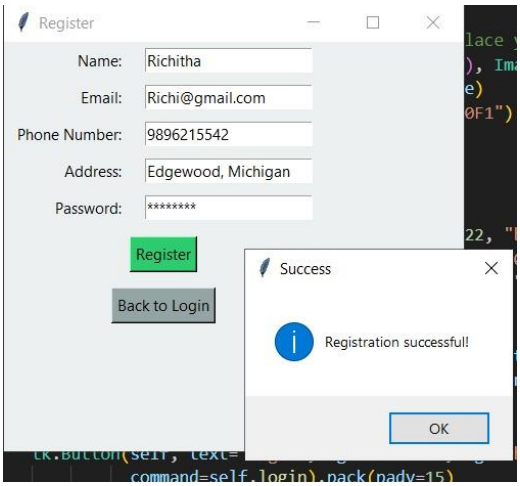
## Mechanic Credentials (Like any):

Username: MechUser4

Password: MechPass4

# Login Page for Admin and Customer

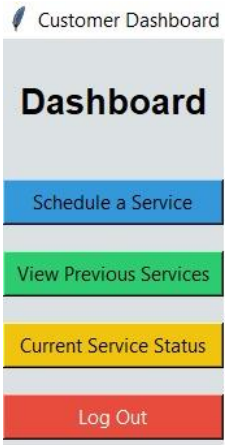



This page presents a user interface designed for both administrators and customers to securely log in to a garage services platform. The login section prompts users to enter their email or username and password, with a visible "Login" button and a "Register" link for new users.

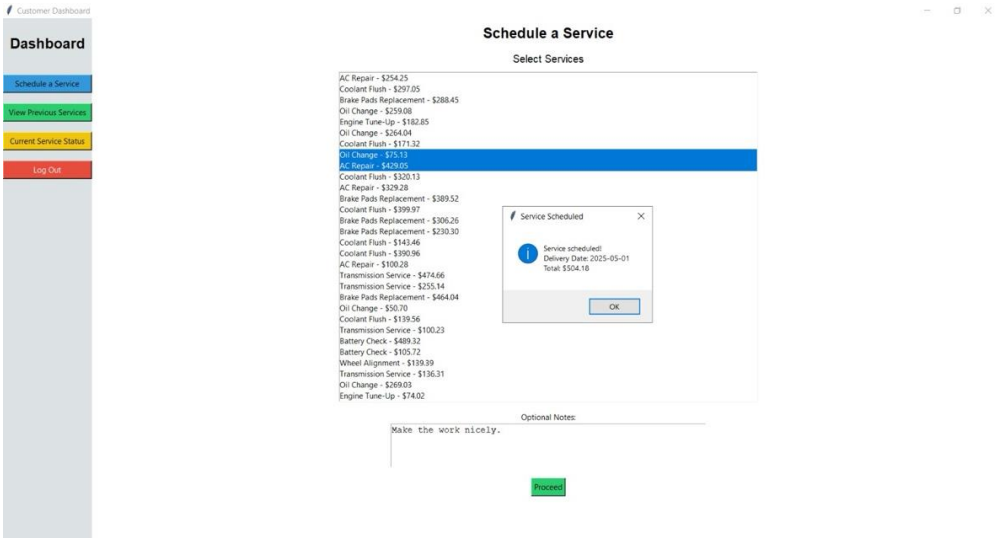
The registration section allows new customers to create an account by providing their name, email, phone number, address, and a chosen password, featuring a "Register" button and a "Back to Login" option. Upon successful registration, a confirmation dialog box appears, indicating "Registration successful!" with an "OK" button. The overall design is clean and straightforward, aiming for ease of use for both administrative and customer access.

# Customer Login Dashboard



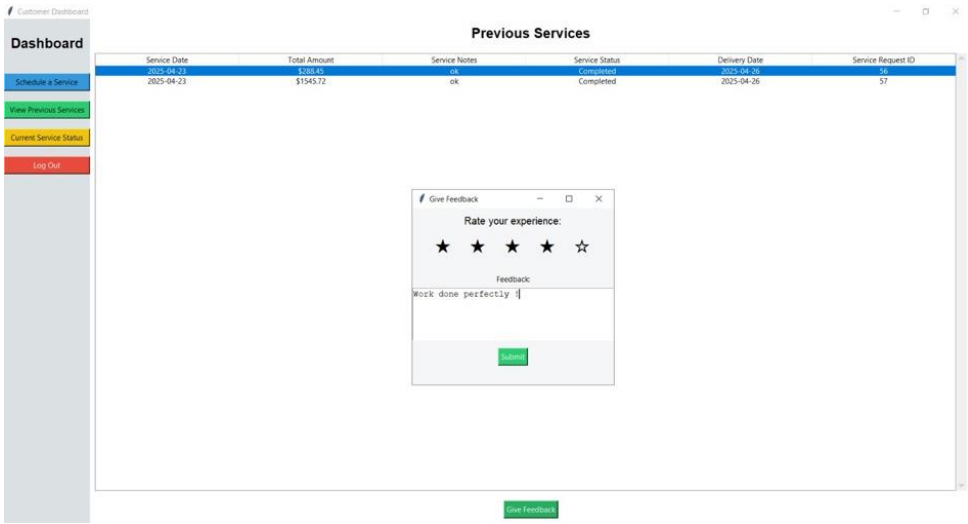

This screen displays a simplified customer login dashboard for a garage service platform. Upon logging in, customers are presented with a clear and concise menu on the left-hand side, titled "Dashboard." This dashboard provides quick access to essential functionalities, enabling customers to "Schedule a Service," "View Previous Services," check their "Current Service Status," and securely "Log Out" of their account. The use of distinct colors for each button enhances visual clarity and ease of navigation, allowing customers to efficiently manage their garage service needs.

# Schedule Service Screen



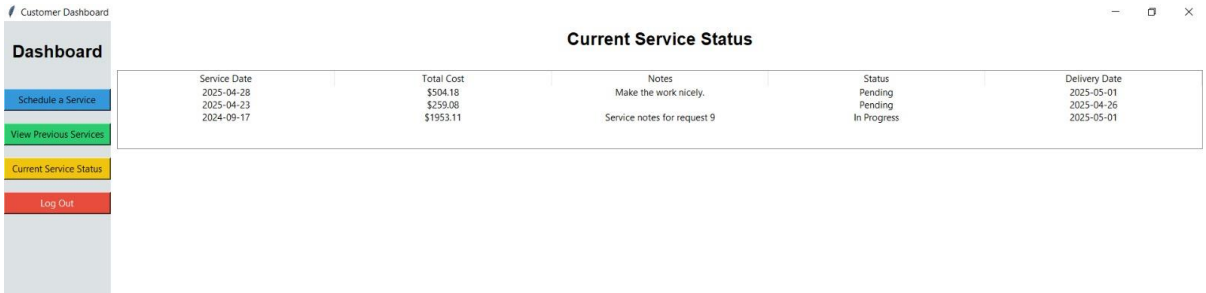

The "Schedule Service" screen presents customers with a list of garage services and their prices. Users can select a service, like the highlighted "Oil Change - $75.13," and add optional notes. A confirmation pop-up appears, showing the selected service and scheduled date (May 1, 2025), allowing the customer to finalize their booking.

# Previous Screen



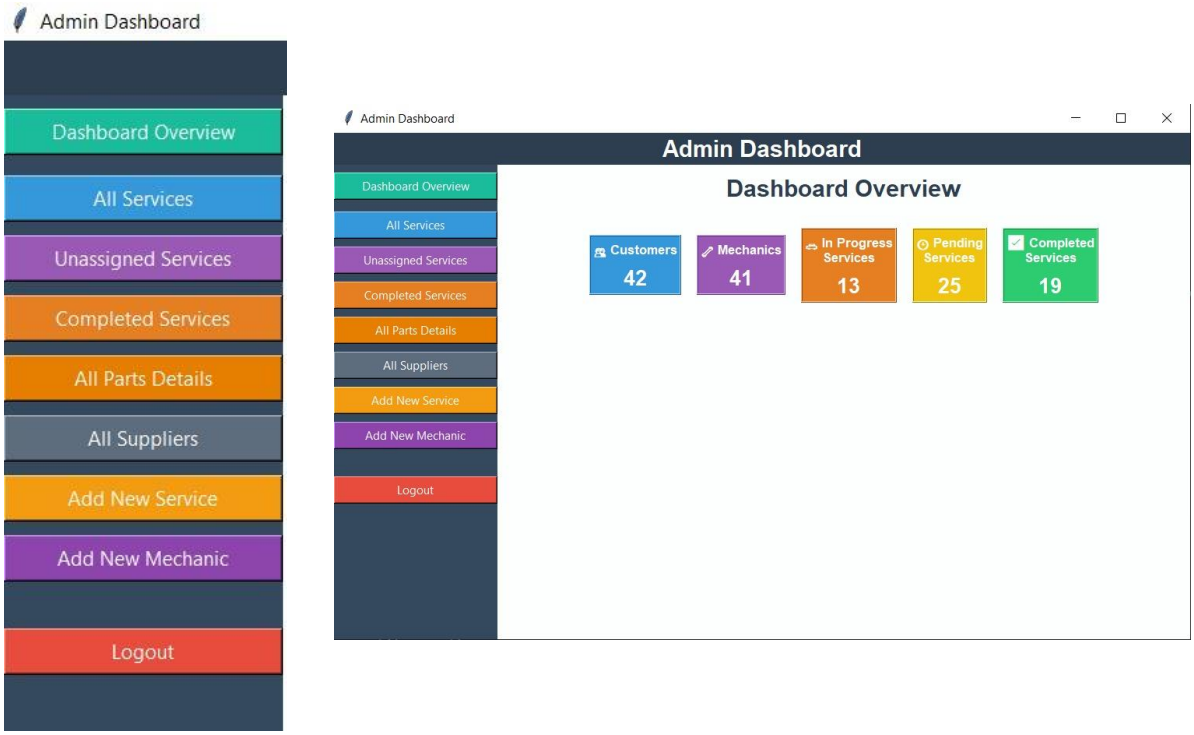

This "Previous Services" screen within the customer dashboard displays a record of completed services, including the service date, total amount, any service notes, the service status ("Completed" in both listed instances), the delivery date, and a unique service request ID. Below the service history, a "Give Feedback" button is visible, and a pop-up window titled "Give Feedback" is currently open. This window allows customers to rate their experience using a five-star system and provide written feedback in a text area, as shown with the comment "Work done perfectly”.

# Current Service Status




The "Current Service Status" screen provides customers with a real-time overview of their ongoing service requests. The table displays key information for each service, including the service date, total cost, any specific notes provided, the status (such as "Pending" or "In Progress"), and the estimated delivery date. This allows customers to easily track the progress of their scheduled garage services and stay informed about when to expect their vehicle to be ready.

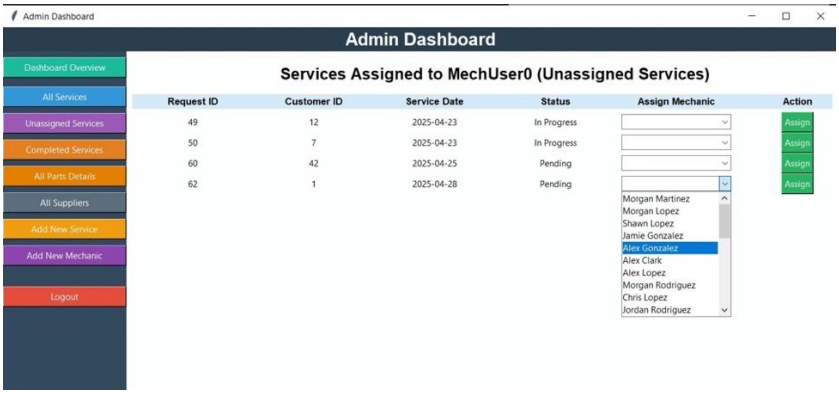
# Admin Dashboard Page

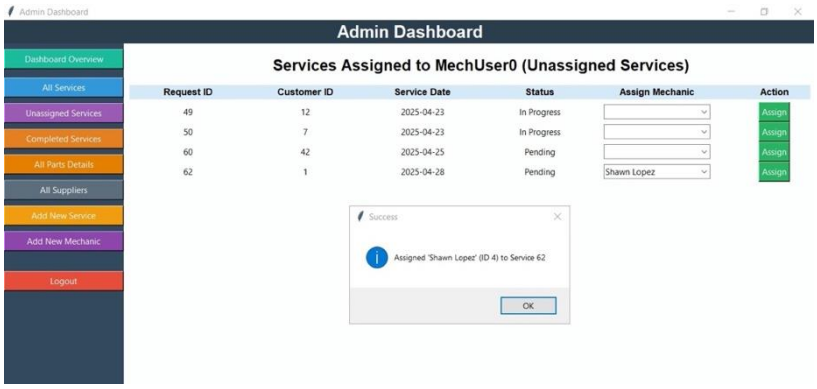



This image showcases the "Admin Dashboard Page" for the garage services platform, providing administrators with a comprehensive overview of key operational metrics. The left-hand navigation menu offers access to various sections, including "Dashboard Overview," "All Services," "Unassigned Services," "Completed Services," "All Parts Details," "All Suppliers," "Add New Service," "Add New Mechanic," and "Logout."

The main "Dashboard Overview" section displays summarized data in distinct tiles: "Customers" (42), "Mechanics" (41), "In Progress Services" (13), "Pending Services" (25), and "Completed Services" (19). This layout allows administrators to quickly grasp the current state of the platform, monitor workload, and navigate to specific areas for more detailed management.

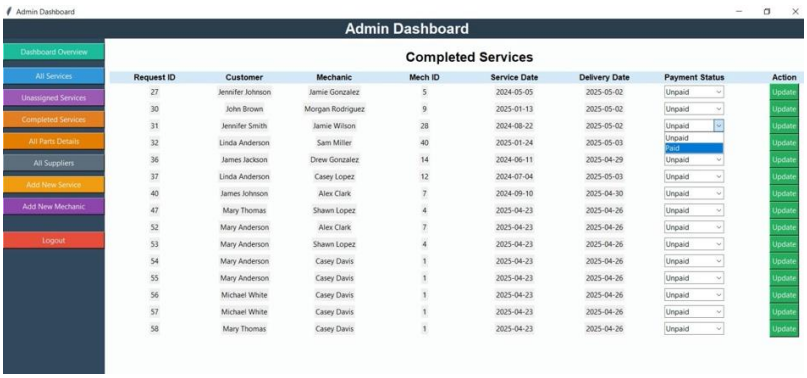
# SCHEDULING ASSIGN MECHANIC



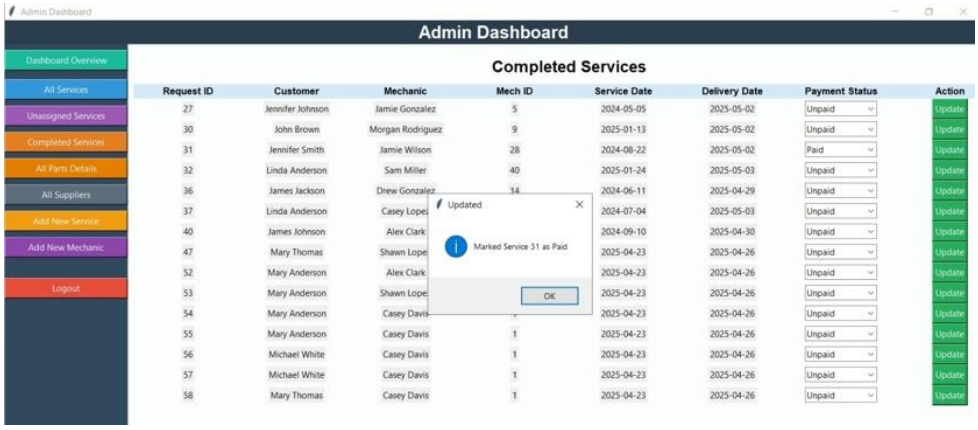

This "Scheduling Assign Mechanic" screen, accessible from the admin dashboard, displays a list of "Unassigned Services" or services assigned to a specific mechanic For each service request, identified by a "Request ID," the table shows the "Customer ID," "Service Date," and current "Status." An "Assign Mechanic" dropdown menu is available for each service, allowing administrators to select a mechanic from the list. An "Assign" button next to each dropdown enables the administrator to finalize the assignment of a mechanic to that service request. This interface streamlines the process of distributing workload among mechanics.

# Completed services screen for the following task

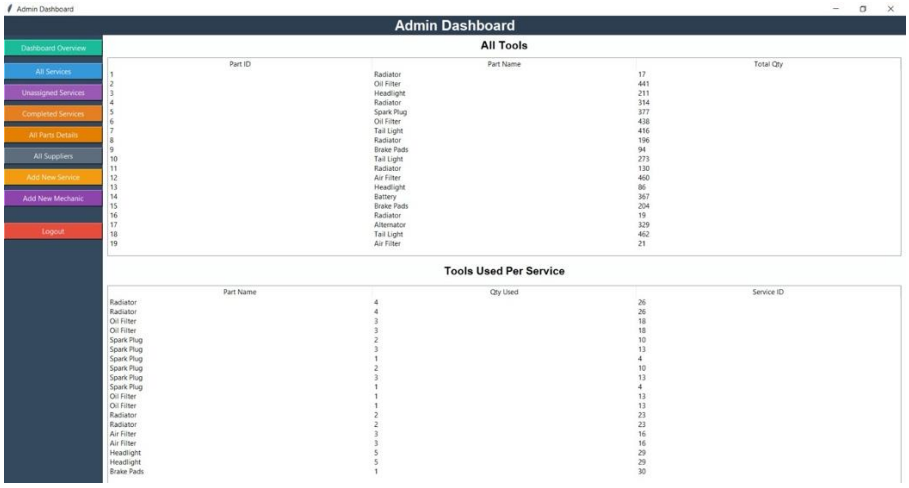



This "Completed Services" screen within the admin dashboard provides a detailed overview of finished service requests. The table lists each completed service with its "Request ID," associated "Customer," assigned "Mechanic" and their "Mech ID," the "Service Date," "Delivery Date," and "Payment Status." An "Action" column is present, suggesting further options for each completed service. In the lower portion of the

image, a pop-up message is visible, indicating that the "Payment Status" for "Request ID 31" has been updated to "Paid" by the administrator. This screen allows administrators to track completed work, payment statuses, and potentially perform post-service actions.




# All Supplies

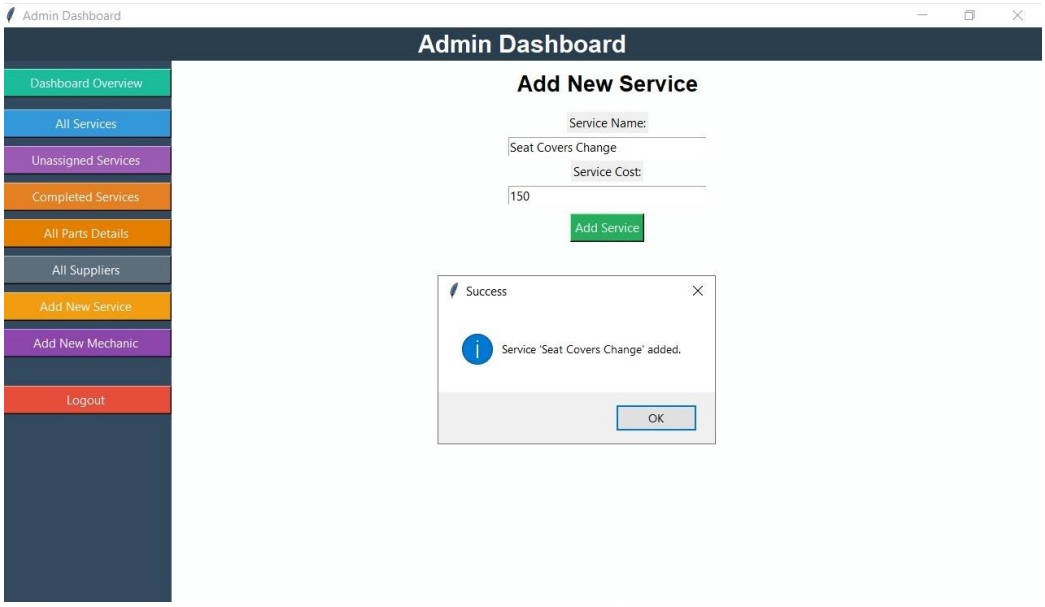



This "All Parts Details" screen within the admin dashboard offers a comprehensive inventory overview. The top section, labeled "All Tools," lists each "Part ID" and "Part Name" along with its "Total Qty" in stock. Below this, the "Tools Used Per Service" section provides a breakdown of which "Part Name" was used, the "Qty Used," and the corresponding "Service ID" for each instance. This detailed view allows administrators to monitor part availability, track usage across services, and manage inventory effectively.

# Suppliers with their Contact details

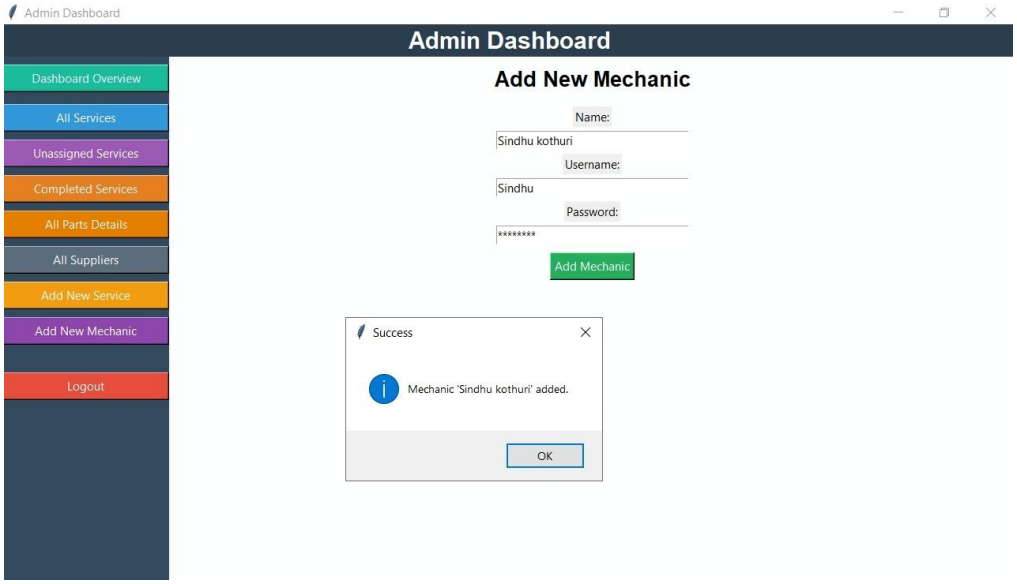
The "Suppliers List" screen within the admin dashboard presents a detailed directory of the platform's suppliers. The table organizes information for each supplier, including their unique "Supplier ID," "Name," "Contact" number, and "Address." This comprehensive list allows administrators to easily access and manage information for all parts and service providers associated with the garage platform.

# New service screen – when customer booked for new service




The "Add New Service" screen, found within the admin dashboard, provides administrators with the functionality to introduce new service offerings to the platform. It features input fields for the "Service Name," where "Seat Covers Change" has been entered, and the "Service Cost," shown here as "150." An "Add Service" button allows the administrator to submit the new service details. Upon successful addition, a confirmation pop-up appears, stating "Service 'Seat Covers Change' added," accompanied by an "OK" button to acknowledge the action.

# New mechanic screen



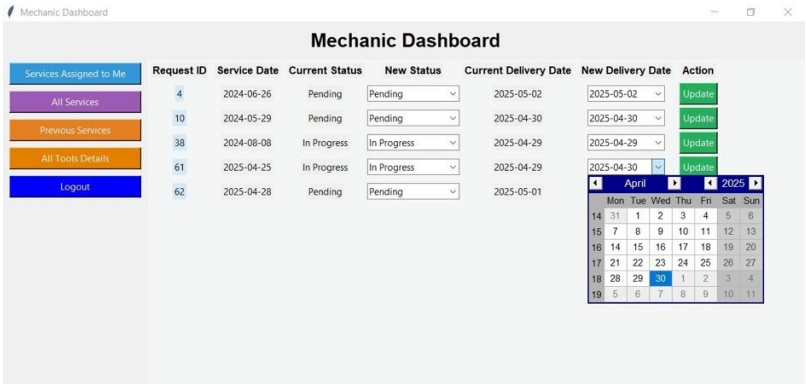

The "Add New Mechanic" screen, accessible through the admin dashboard, enables administrators to register new mechanic profiles on the platform. It includes fields for the

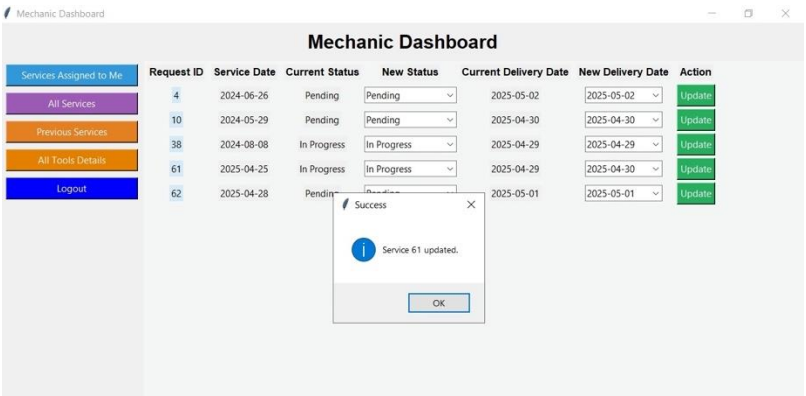
mechanic's "Name" ("Sindhu kothuri" is entered), a designated "Username" ("Sindhu"), and a secure "Password." An "Add Mechanic" button initiates the creation of the new profile. Upon successful registration, a confirmation message appears in a pop-up window, stating "Mechanic 'Sindhu kothuri' added," with an "OK" button to close the notification.

# Mechanic Dashboard


**Mechanic Dashboard** section provides a detailed list of services allocated to the logged-in mechanic. Each task includes key details such as the vehicle information, assigned service date, and the priority level of the service. This ensures that mechanics can efficiently plan their work schedule, prioritize urgent services, and stay organized throughout the day.

# Service assigned for mechanic

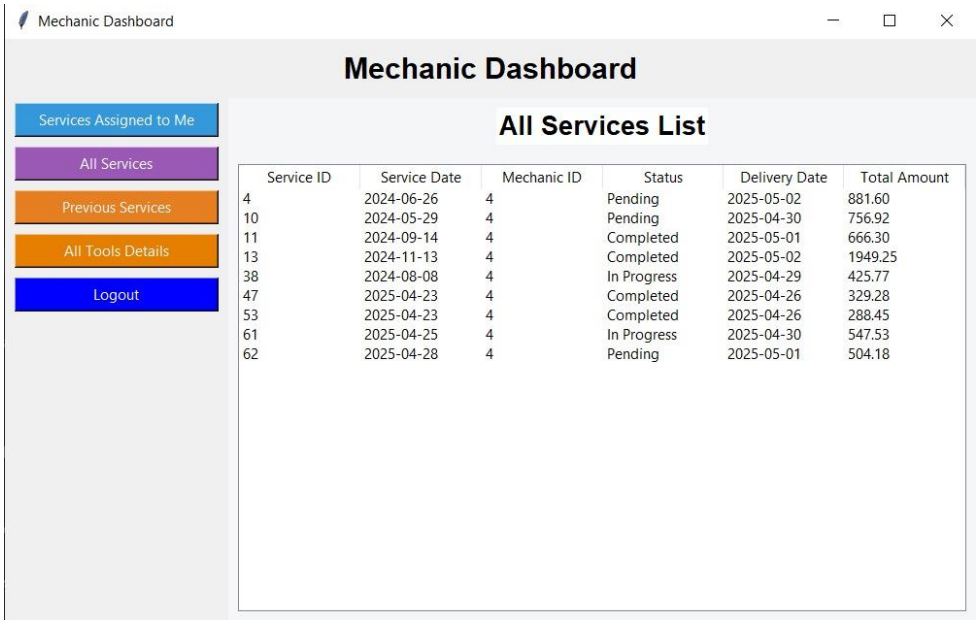



The **New Service Added** feature highlights any newly assigned service tasks for the mechanic. This ensures that mechanics are immediately informed of fresh

assignments without needing to manually check, enabling faster response times and improved service management.

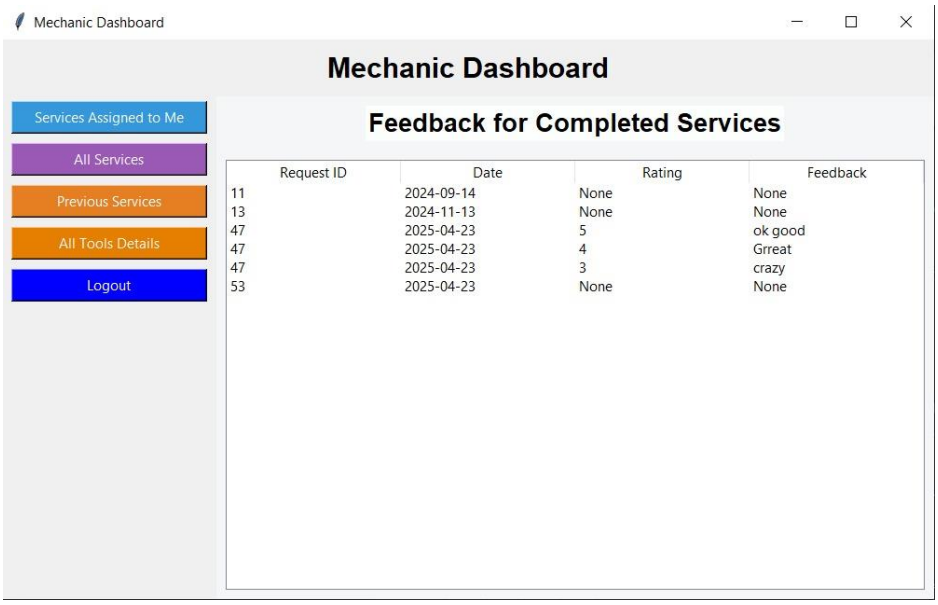
## All service list for Mechanic Dashboard




| Service ID | Service Date | Mechanic ID | Status | Delivery Date | Total Amount |
|---|---|---|---|---|---|
| 4 | 2024-06-26 | 4 | Pending | 2025-05-02 | 881.60 |
| 10 | 2024-05-29 | 4 | Pending | 2025-04-30 | 756.92 |
| 11 | 2024-09-14 | 4 | Completed | 2025-05-01 | 666.30 |
| 13 | 2024-11-13 | 4 | Completed | 2025-05-02 | 1949.25 |
| 38 | 2024-08-08 | 4 | In Progress | 2025-04-29 | 425.77 |
| 47 | 2025-04-23 | 4 | Completed | 2025-04-26 | 329.28 |
| 53 | 2025-04-23 | 4 | Completed | 2025-04-26 | 288.45 |
| 61 | 2025-04-25 | 4 | In Progress | 2025-04-30 | 547.53 |
| 62 | 2025-04-28 | 4 | Pending | 2025-05-01 | 504.18 |

The **All Service List for Mechanic** section displays a complete history of all tasks assigned to the mechanic, including both pending and completed services. The interface provides filtering options that allow mechanics to easily sort through services based on status, service date, or priority. This helps mechanics maintain full visibility over their workload and past service performance.

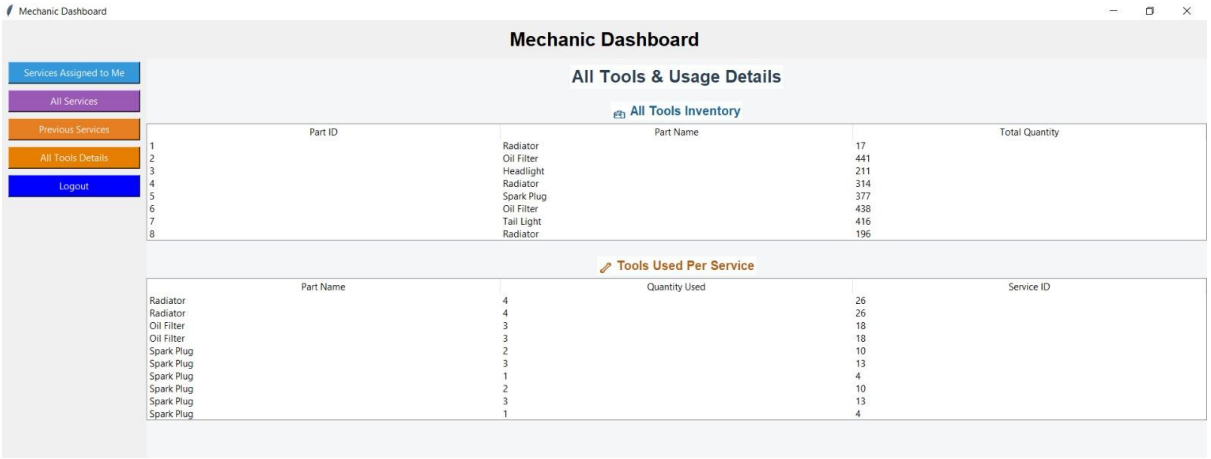
## Feedback for Completed Service




| Request ID | Date | Rating | Feedback |
|---|---|---|---|
| 11 | 2024-09-14 | None | None |
| 13 | 2024-11-13 | None | None |
| 47 | 2025-04-23 | 5 | ok good |
| 47 | 2025-04-23 | 4 | Grreat |
| 47 | 2025-04-23 | 3 | crazy |
| 53 | 2025-04-23 | None | None |

In the **Feedback for Completed Service** section, mechanics can view customer ratings and feedback provided after a service is completed. This helps mechanics track their performance and identify areas for improvement. Positive feedback encourages high service standards, while constructive criticism provides an opportunity for skill development and better customer service.

# Mechanic Tools and usage details




The **Mechanic Tools and Usage Details** section is designed for managing and tracking the tools issued to mechanics for specific service tasks. It maintains a list of all tools assigned to the mechanic, monitors the usage of each tool during services, and ensures that tool return records are updated once a service is completed. This feature supports efficient inventory management and helps in preventing tool loss or misuse.

# Appendix:

## Login Page:

```
import tkinter as tk
from tkinter import messagebox
from PIL import Image, ImageTk
import mysql.connector
from RegisterPage import RegisterPage
from main import MainApp
from Admindashboard import AdminDashboard
from customer_dashboard import CustomerDashboard
from Mechanic_dashboard import MechanicDashboard
import os
ADMIN_CREDENTIALS = {"admin": "admin123"}
def get_db_connection():
    return mysql.connector.connect(
        host='141.209.241.91',
        port=3306,
        user='sp2025bis698g6',
        password='warm',
        database='sp2025bis698g6s'
    )
def get_user_role(username, password):
    if username in ADMIN_CREDENTIALS and ADMIN_CREDENTIALS[username] == password:
        return "Admin", None
    conn = get_db_connection()
    cursor = conn.cursor(dictionary=True)
    cursor.execute("SELECT Mechanic_ID FROM MECHANIC WHERE Mechanic_User_Name = %s AND
Mechanic_Password = %s", (username, password))
    mech = cursor.fetchone()
    if mech:
        return "Mechanic", mech["Mechanic_ID"]

    cursor.execute("SELECT Customer_ID FROM CUSTOMER WHERE Email = %s AND Password = %s",
(username, password))
    cust = cursor.fetchone()
    if cust:
        return "Customer", cust["Customer_ID"]

    conn.close()
    return None, None

class LoginPage(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("Login")
        self.geometry("1000x800")
        self.configure(bg="#ECF0F1")

        # Logo Section
        try:
            current_directory = os.path.dirname(os.path.abspath(__file__))
            logo_path = os.path.join(current_directory, "logo.png")
            logo_image = Image.open(logo_path)
            logo_image = logo_image.resize((350, 350), Image.LANCZOS)
            self.logo = ImageTk.PhotoImage(logo_image)
            tk.Label(self, image=self.logo, bg="#ECF0F1").pack(pady=(30, 10))
        except Exception as e:
            print(f"Logo load failed: {e}")

        # Login Form
        tk.Label(self, text="Login", font=("Arial", 22, "bold"), bg="#ECF0F1").pack(pady=10)
        tk.Label(self, text="Email or Username", bg="#ECF0F1", font=("Arial", 12)).pack()
```

```
        self.username_entry = tk.Entry(self, font=("Arial", 12), width=30)
        self.username_entry.pack(pady=5)

        tk.Label(self, text="Password", bg="#ECF0F1", font=("Arial", 12)).pack()
        self.password_entry = tk.Entry(self, show="*", font=("Arial", 12), width=30)
        self.password_entry.pack(pady=5)

        tk.Button(self, text="Login", bg="#3498DB", fg="white", font=("Arial", 12, "bold"), width=15,
                command=self.login).pack(pady=15)
        tk.Button(self, text="Register", bg="#2ECC71", fg="white", font=("Arial", 11),
command=self.open_register_page).pack()

    def open_register_page(self):
        self.destroy()
        RegisterPage()

    def login(self):
        username = self.username_entry.get()
        password = self.password_entry.get()
        role, user_id = get_user_role(username, password)

        if role == "Customer":
            messagebox.showinfo("Login Successful", f"Welcome, {username} ({role})")
            self.destroy()
            CustomerDashboard(customer_id=user_id).mainloop()

        elif role == "Mechanic":
            messagebox.showinfo("Login Successful", f"Welcome, {username} ({role})")
            self.destroy()
            root = tk.Tk()
            root.title("Mechanic Dashboard")
            MechanicDashboard(root, controller=SimpleController(root), role=role, mechanic_id=user_id)
            root.mainloop()

        elif role == "Admin":
            messagebox.showinfo("Login Successful", f"Welcome, {username} (Admin)")
            self.destroy()
            root = tk.Tk()
            AdminDashboard(root)
            root.mainloop()

        else:
            messagebox.showerror("Login Failed", "Invalid username or password.")

class SimpleController:
    def __init__(self, parent):
        self.parent = parent

    def add_back_button(self, dashboard):
        tk.Button(self.parent, text="Back", command=self.go_back).pack(pady=5)

    def go_back(self):
        self.parent.destroy()
        LoginPage().mainloop()

if __name__ == "__main__":
    LoginPage().mainloop()
```

## Customer Page:

```
import tkinter as tk
from tkinter import ttk
from tkinter import messagebox
import mysql.connector
from datetime import datetime, timedelta

class CustomerDashboard(tk.Tk):
    def __init__(self, customer_id):
        super().__init__()
        self.customer_id = customer_id
        self.title("Customer Dashboard")
        self.geometry("900x500")
        self.configure(bg="#F9F9F9")

        # Left panel
        left_frame = tk.Frame(self, width=200, bg="#DCE1E3")
        left_frame.pack(side="left", fill="both")

        tk.Label(left_frame, text="Dashboard", font=("Arial", 16, "bold"), bg="#DCE1E3").pack(pady=30)
        tk.Button(left_frame, text="Schedule a Service", width=20, command=self.schedule_service,
bg="#3498DB").pack(pady=10)
        tk.Button(left_frame, text="View Previous Services", width=20, command=self.view_previous_services,
bg="#2ECC71").pack(pady=10)
        tk.Button(left_frame, text="Current Service Status", width=20, command=self.check_current_status,
bg="#F1C40F").pack(pady=10)
        tk.Button(left_frame, text="Log Out", width=20, command=self.logout, bg="#E74C3C",
fg="white").pack(pady=10)

        # Right panel
        self.right_frame = tk.Frame(self, bg="#FFFFFF")
        self.right_frame.pack(side="right", fill="both", expand=True)

    def logout(self):
        """Cleanly exit the dashboard."""
        confirm = messagebox.askyesno("Confirm Logout", "Are you sure you want to log out?")
        if confirm:
            self.destroy()
            from Login import LoginPage  # <- local import avoids circular dependency
            LoginPage().mainloop()

    def connect_db(self):
        return mysql.connector.connect(
            host='141.209.241.91',
            port=3306,
            user='sp2025bis698g6',
            password='warm',
            database='sp2025bis698g6s'
        )

    def clear_right_frame(self):
        for widget in self.right_frame.winfo_children():
            widget.destroy()

    def schedule_service(self):
        self.clear_right_frame()

        tk.Label(self.right_frame, text="Schedule a Service", font=("Arial", 16, "bold"),
bg="#FFFFFF").pack(pady=10)

        tk.Label(self.right_frame, text="Select Services", font=("Arial", 12), bg="#FFFFFF").pack(pady=5)

        conn = self.connect_db()
        cursor = conn.cursor()
```

```
        cursor.execute("SELECT Service_ID, ServiceName, Service_Cost FROM SERVICES")
        self.services = cursor.fetchall()
        conn.close()

        self.service_map = {i: (sid, cost) for i, (sid, name, cost) in enumerate(self.services)}

        self.service_listbox = tk.Listbox(self.right_frame, selectmode="multiple", height=30, width=100,
exportselection=0)
        for _, name, cost in self.services:
            self.service_listbox.insert("end", f"{name} - ${cost}")
        self.service_listbox.pack(padx=20, pady=5)

        tk.Label(self.right_frame, text="Optional Notes:", bg="#FFFFFF").pack(pady=(10, 0))
        self.notes_entry = tk.Text(self.right_frame, height=4, width=60)
        self.notes_entry.pack(pady=(0, 10))

        tk.Button(self.right_frame, text="Proceed", bg="#2ECC71",
command=self.proceed_service).pack(pady=10)

    def proceed_service(self):
        selected_indices = self.service_listbox.curselection()
        if not selected_indices:
            messagebox.showwarning("No Service Selected", "Please select at least one service.")
            return

        selected_services = [self.service_map[i] for i in selected_indices]
        total_amount = sum(cost for _, cost in selected_services)
        service_date = datetime.today().date()
        delivery_date = service_date + timedelta(days=3)
        notes = self.notes_entry.get("1.0", "end").strip()

        conn = self.connect_db()
        cursor = conn.cursor()

        cursor.execute("""
            INSERT INTO SERVICE_REQUEST (Customer_ID, Total_Amount, Service_Date, Service_Notes,
Mechanic_ID, Service_Status, Delivery_Date)
            VALUES (%s, %s, %s, %s, %s, %s, %s)
        """, (self.customer_id, total_amount, service_date, notes, 1, "Pending", delivery_date))
        service_request_id = cursor.lastrowid

        for sid, cost in selected_services:
            cursor.execute("""
                INSERT INTO SERVICE_SELECTION (Service_ID, Customer_ID, Service_Cost)
                VALUES (%s, %s, %s)
            """, (sid, self.customer_id, cost))

        conn.commit()
        conn.close()

        messagebox.showinfo("Service Scheduled", f"Service scheduled!\nDelivery Date: {delivery_date}\nTotal:
${total_amount:.2f}")
        self.check_current_status()
         # Reset the form

    def view_previous_services(self):
        self.clear_right_frame()
        tk.Label(self.right_frame, text="Previous Services", font=("Arial", 16, "bold"),
bg="#FFFFFF").pack(pady=10)

        container = tk.Frame(self.right_frame, bg="#FFFFFF")
        container.pack(fill="both", expand=True, padx=10, pady=10)

        columns = ("Service_Date", "Total_Amount", "Service_Notes", "Service_Status", "Delivery_Date",
"Service_Request_ID")
        self.tree = ttk.Treeview(container, columns=columns, show="headings", height=15)
```

```
        for col in columns:
            self.tree.heading(col, text=col.replace("_", " "))
            self.tree.column(col, anchor="center", width=140)

        self.tree.pack(side="left", fill="both", expand=True)

        scrollbar = ttk.Scrollbar(container, orient="vertical", command=self.tree.yview)
        self.tree.configure(yscrollcommand=scrollbar.set)
        scrollbar.pack(side="right", fill="y")

        try:
            conn = self.connect_db()
            cursor = conn.cursor(dictionary=True)
            cursor.execute("""
                SELECT * FROM SERVICE_REQUEST
                WHERE Customer_ID = %s AND Service_Status = 'Completed'
                ORDER BY Service_Date DESC
            """, (self.customer_id,))
            services = cursor.fetchall()
            conn.close()

            if not services:
                self.tree.insert("", "end", values=("No records", "", "", "", "", ""))
            else:
                for svc in services:
                    self.tree.insert("", "end", values=(
                        svc['Service_Date'],
                        f"${svc['Total_Amount']:.2f}",
                        svc['Service_Notes'][:50] + ("..." if len(svc['Service_Notes']) > 50 else ""),
                        svc['Service_Status'],
                        svc['Delivery_Date'],
                        svc['Service_Request_ID']
                    ))

            tk.Button(self.right_frame, text="Give Feedback", bg="#27AE60", fg="white",
                    command=lambda: self.open_feedback_form(self.tree)).pack(pady=10)

        except Exception as e:
            messagebox.showerror("Error", str(e))

    def open_feedback_form(self, tree):
        selected = tree.selection()
        if not selected:
            messagebox.showwarning("Warning", "Please select a service to give feedback on.")
            return

        item = tree.item(selected[0])
        values = item["values"]
        service_request_id = values[5]

        feedback_window = tk.Toplevel(self)
        feedback_window.title("Give Feedback")
        feedback_window.geometry("400x350")
        feedback_window.configure(bg="#F4F6F7")

        tk.Label(feedback_window, text="Rate your experience:", bg="#F4F6F7", font=("Arial",
12)).pack(pady=(10, 5))

        # Star rating UI
        star_frame = tk.Frame(feedback_window, bg="#F4F6F7")
        star_frame.pack()
        rating_var = tk.IntVar(value=0)
        stars = []

        def update_stars(selected_rating):
            rating_var.set(selected_rating)
            for i in range(5):
```

```
            stars[i].config(text="★" if i < selected_rating else "☆")

        for i in range(5):
            btn = tk.Button(star_frame, text="☆", font=("Arial", 20), bd=0, bg="#F4F6F7", command=lambda
r=i+1: update_stars(r))
            btn.grid(row=0, column=i, padx=5)
            stars.append(btn)

        # Feedback box
        tk.Label(feedback_window, text="Feedback:", bg="#F4F6F7").pack(pady=(20, 5))
        feedback_text = tk.Text(feedback_window, height=5, width=40)
        feedback_text.pack()

        # Pre-fill if feedback exists
        try:
            conn = self.connect_db()
            cursor = conn.cursor(dictionary=True)
            cursor.execute("""
                SELECT Rating, Customer_feedback FROM FEEDBACK
                WHERE Customer_ID = %s AND Service_Request_ID = %s
            """, (self.customer_id, service_request_id))
            existing = cursor.fetchone()
            conn.close()

            if existing:
                update_stars(existing["Rating"])
                feedback_text.insert("1.0", existing["Customer_feedback"])

        except Exception as e:
            messagebox.showerror("Load Feedback Error", str(e))

        def submit_feedback():
            rating = rating_var.get()
            feedback = feedback_text.get("1.0", "end").strip()

            if rating == 0:
                messagebox.showerror("Error", "Please select a star rating.")
                return

            try:
                conn = self.connect_db()
                cursor = conn.cursor(dictionary=True)

                cursor.execute("""
                    SELECT Feedback_ID FROM FEEDBACK
                    WHERE Customer_ID = %s AND Service_Request_ID = %s
                """, (self.customer_id, service_request_id))
                existing = cursor.fetchone()

                if existing:
                    cursor.execute("""
                        UPDATE FEEDBACK
                        SET Customer_feedback = %s, Rating = %s
                        WHERE Feedback_ID = %s
                    """, (feedback, rating, existing["Feedback_ID"]))
                    messagebox.showinfo("Updated", "Your feedback has been updated.")
                else:
                    cursor.execute("""
                        INSERT INTO FEEDBACK (Service_Request_ID, Customer_feedback, Rating, Customer_ID)
                        VALUES (%s, %s, %s, %s)
                    """, (service_request_id, feedback, rating, self.customer_id))
                    messagebox.showinfo("Success", "Feedback submitted successfully!")

                conn.commit()
                conn.close()
                feedback_window.destroy()
```

```
        except Exception as e:
            messagebox.showerror("Database Error", str(e))
    tk.Button(feedback_window, text="Submit", command=submit_feedback, bg="#2ECC71",
fg="white").pack(pady=15)
  def check_current_status(self):
    self.clear_right_frame()
    tk.Label(self.right_frame, text="Current Service Status", font=("Arial", 16, "bold"),
bg="#FFFFFF").pack(pady=10)

    columns = ("Service Date", "Total Cost", "Notes", "Status", "Delivery Date")
    tree = ttk.Treeview(self.right_frame, columns=columns, show="headings", height=5)

    for col in columns:
        tree.heading(col, text=col)
        tree.column(col, width=160, anchor="center")

    tree.pack(padx=10, pady=20, fill="x")

    try:
        conn = self.connect_db()
        cursor = conn.cursor(dictionary=True)
        cursor.execute("""
            SELECT * FROM SERVICE_REQUEST
            WHERE Customer_ID = %s AND Service_Status IN ('Pending', 'In Progress')
            ORDER BY Service_Date DESC
        """, (self.customer_id,))
        services = cursor.fetchall()
        conn.close()

        if not services:
            tree.insert("", "end", values=("No records", "", "", "", ""))
        else:
            for svc in services:
                tree.insert("", "end", values=(
                    svc['Service_Date'],
                    f"${svc['Total_Amount']:.2f}",
                    svc['Service_Notes'][:50] + ("..." if len(svc['Service_Notes']) > 50 else ""),
                    svc['Service_Status'],
                    svc['Delivery_Date']
                ))

    except Exception as e:
        messagebox.showerror("Error", str(e))
  def check_current_status(self):
    self.clear_right_frame()

    tk.Label(self.right_frame, text="Current Service Status", font=("Arial", 16, "bold"),
bg="#FFFFFF").pack(pady=10)

    columns = ("Service Date", "Total Cost", "Notes", "Status","Delivery Date")
    tree = ttk.Treeview(self.right_frame, columns=columns, show="headings", height=5)
    for col in columns:
        tree.heading(col, text=col)
        tree.column(col, width=200, anchor="center")
    tree.pack(padx=10, pady=20, fill="x")

    try:
        conn = self.connect_db()
        cursor = conn.cursor(dictionary=True)
        cursor.execute("""
            SELECT * FROM SERVICE_REQUEST
            WHERE Customer_ID = %s AND Service_Status IN ('Pending' , 'In Progress')
            ORDER BY Service_Date DESC
        """, (self.customer_id,))
        services = cursor.fetchall()
        conn.close()
```

```
            if not services:
                tree.insert("", "end", values=("No records", "", "", "", ""))
            else:
                for svc in services:
                    tree.insert("", "end", values=(
                        svc['Service_Date'],
                        f"${svc['Total_Amount']:.2f}",
                        svc['Service_Notes'][:50] + ("..." if len(svc['Service_Notes']) > 50 else ""),
                        svc['Service_Status'],
                        svc['Delivery_Date']
                    ))

        except Exception as e:
            messagebox.showerror("Error", str(e))
# For testing only
if __name__ == "__main__":
    app = CustomerDashboard(customer_id=1)
    app.mainloop()
```

## Admin Login:

```
import tkinter as tk
from tkinter import ttk, messagebox
import mysql.connector

def get_db_connection():
    return mysql.connector.connect(
        host='141.209.241.91',
        port=3306,
        user='sp2025bis698g6',
        password='warm',
        database='sp2025bis698g6s'
    )

class AdminDashboard:
    def __init__(self, root):
        self.root = root
        self.root.geometry("1100x650")
        self.root.title("Admin Dashboard")
        self.root.configure(bg="#ECF0F1")

        tk.Label(root, text="Admin Dashboard", font=("Arial", 18, "bold"), bg="#2C3E50",
fg="white").pack(fill=tk.X)

        button_frame = tk.Frame(root, bg="#34495E", width=200)
        button_frame.pack(side=tk.LEFT, fill=tk.Y)

        self.content_frame = tk.Frame(root, bg="#FDFEFE")
        self.content_frame.pack(side=tk.RIGHT, fill=tk.BOTH, expand=True)

        # Navigation Buttons
        tk.Button(button_frame, text="Dashboard Overview", bg="#1ABC9C", fg="white", width=25,
                command=self.show_metrics).pack(pady=10)
        tk.Button(button_frame, text="All Services", bg="#3498DB", fg="white", width=25,
                command=self.show_all_services).pack(pady=5)
        tk.Button(button_frame, text="Unassigned Services", bg="#9B59B6", fg="white", width=25,
                command=self.show_unassigned_services).pack(pady=5)
        tk.Button(button_frame, text="Completed Services", bg="#E67E22", fg="white", width=25,
                command=self.show_completed_services).pack(pady=5)
        tk.Button(button_frame, text="All Parts Details", bg="#E67E00", fg="white", width=25,
                command=self.show_parts_details).pack(pady=5)
        tk.Button(button_frame, text="All Suppliers", bg="#5D6D7E", fg="white", width=25,
                command=self.show_suppliers).pack(pady=5)
        tk.Button(button_frame, text="Add New Service", bg="#F39C12", fg="white", width=25,
```

```
        command=self.show_add_service_form).pack(pady=5)
    tk.Button(button_frame, text="Add New Mechanic", bg="#8E44AD", fg="white", width=25,
        command=self.show_add_mechanic_form).pack(pady=5)
    tk.Button(button_frame, text="Logout", bg="#E74C3C", fg="white", width=25,
        command=self.logout).pack(pady=30)


    self.show_metrics()

  def logout(self):
    self.root.destroy()
    from Login import LoginPage
    LoginPage().mainloop()


  def clear_content(self):
    for widget in self.content_frame.winfo_children():
        widget.destroy()

  def show_metrics(self):
    self.clear_content()

    tk.Label(self.content_frame, text="Dashboard Overview", font=("Arial", 18, "bold"), bg="#FDFEFE",
fg="#2C3E50").pack(pady=10)

    conn = get_db_connection()
    cursor = conn.cursor()

    cursor.execute("SELECT COUNT(*) FROM CUSTOMER")
    customers = cursor.fetchone()[0]

    cursor.execute("SELECT COUNT(*) FROM MECHANIC")
    mechanics = cursor.fetchone()[0]

    cursor.execute("SELECT COUNT(*) FROM SERVICE_REQUEST WHERE Service_Status = 'In
Progress'")
    in_progress = cursor.fetchone()[0]

    cursor.execute("SELECT COUNT(*) FROM SERVICE_REQUEST WHERE Service_Status = 'Pending'")
    pending = cursor.fetchone()[0]

    cursor.execute("SELECT COUNT(*) FROM SERVICE_REQUEST WHERE Service_Status =
'Completed'")
    completed = cursor.fetchone()[0]

    conn.close()

    stats = [
        ("👥 Customers", customers, "#3498DB"),
        ("🔧 Mechanics", mechanics, "#9B59B6"),
        ("🚗 In Progress Services", in_progress, "#E67E22"),
        ("🕒 Pending Services", pending, "#F1C40F"),
        ("✅ Completed Services", completed, "#2ECC71")
    ]

    cards_frame = tk.Frame(self.content_frame, bg="#FDFEFE")
    cards_frame.pack(pady=10)

    for idx, (label, value, color) in enumerate(stats):
        box = tk.Frame(cards_frame, bg=color, width=180, height=100, bd=2, relief=tk.RIDGE)
        box.grid(row=0, column=idx, padx=10, pady=10)
        box.grid_propagate(False)

        tk.Label(box, text=label, font=("Arial", 10, "bold"), bg=color, fg="white", wraplength=150,
justify="center").pack(pady=5)
        tk.Label(box, text=str(value), font=("Arial", 16, "bold"), bg=color, fg="white").pack()
```

```
    def show_all_services(self):
        self.clear_content()
        conn = get_db_connection()
        cursor = conn.cursor()
        cursor.execute("SELECT Service_ID, ServiceName, Service_Cost FROM SERVICES")
        data = cursor.fetchall()
        conn.close()
        self.display_table(["ID", "Service Name", "Cost"], data)

    def show_unassigned_services(self):
        self.clear_content()

        # Add a frame that will use grid layout, and pack it inside the content_frame
        grid_frame = tk.Frame(self.content_frame, bg="#FDFEFE")
        grid_frame.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

        tk.Label(grid_frame, text="Services Assigned to MechUser0 (Unassigned Services)", font=("Arial", 16,
"bold"),
                 bg="#FDFEFE").grid(row=0, column=0, columnspan=6, pady=10)

        conn = get_db_connection()
        cursor = conn.cursor(dictionary=True)

        # Services where Mechanic_ID = 1 (MechUser0)
        cursor.execute("""
            SELECT sr.Service_Request_ID, sr.Customer_ID, sr.Service_Date, sr.Service_Status
            FROM SERVICE_REQUEST sr
            WHERE sr.Mechanic_ID = 1
            AND sr.Service_Status IN ('Pending', 'In Progress')
        """)
        rows = cursor.fetchall()

        if not rows:
            tk.Label(grid_frame, text="No unassigned services.", bg="#FDFEFE", font=("Arial", 12)).grid(row=1,
column=0, columnspan=6)
            return

        # Get list of mechanics (excluding MechUser0)
        cursor.execute("SELECT Name FROM MECHANIC WHERE Mechanic_ID != 1")
        mechanic_names = [r["Name"] for r in cursor.fetchall()]
        conn.close()

        headers = ["Request ID", "Customer ID", "Service Date", "Status", "Assign Mechanic", "Action"]
        for col, header in enumerate(headers):
            tk.Label(grid_frame, text=header, font=("Arial", 10, "bold"), bg="#D6EAF8", width=20).grid(row=2,
column=col, pady=5)

        for row_idx, row in enumerate(rows, start=3):
            req_id = row["Service_Request_ID"]

            tk.Label(grid_frame, text=req_id, bg="#FDFEFE").grid(row=row_idx, column=0)
            tk.Label(grid_frame, text=row["Customer_ID"], bg="#FDFEFE").grid(row=row_idx, column=1)
            tk.Label(grid_frame, text=row["Service_Date"], bg="#FDFEFE").grid(row=row_idx, column=2)
            tk.Label(grid_frame, text=row["Service_Status"], bg="#FDFEFE").grid(row=row_idx, column=3)

            # Create unique StringVar per dropdown
            mech_var = tk.StringVar()
            dropdown = ttk.Combobox(grid_frame, textvariable=mech_var, values=mechanic_names, width=20,
state="readonly")
            dropdown.grid(row=row_idx, column=4)

            # Fix closure issue by using mech_var here
            def make_assigner(req_id=req_id, name_var=mech_var):
                def assign():
                    selected_name = name_var.get()
```

```
                if not selected_name:
                    messagebox.showerror("Missing", "Please select a mechanic.")
                    return
                conn2 = get_db_connection()
                cur2 = conn2.cursor()
                cur2.execute("SELECT Mechanic_ID FROM MECHANIC WHERE Name = %s",
(selected_name,))
                result = cur2.fetchone()
                if result:
                    new_id = result[0]
                    cur2.execute("UPDATE SERVICE_REQUEST SET Mechanic_ID = %s WHERE
Service_Request_ID = %s", (new_id, req_id))
                    conn2.commit()
                    cur2.close()
                    conn2.close()
                    messagebox.showinfo("Success", f"Assigned '{selected_name}' (ID {new_id}) to Service
{req_id}")
                    self.clear_content()
                    self.show_unassigned_services()
                else:
                    messagebox.showerror("Error", "Mechanic not found.")
                    conn2.close()

            return assign

        tk.Button(grid_frame, text="Assign", bg="#28B463", fg="white",
command=make_assigner()).grid(row=row_idx, column=5, padx=5)


    conn.close()

  def show_completed_services(self):
    self.clear_content()

    grid_frame = tk.Frame(self.content_frame, bg="#FDFEFE")
    grid_frame.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

    tk.Label(grid_frame, text="Completed Services", font=("Arial", 16, "bold"), bg="#FDFEFE").grid(row=0,
column=0, columnspan=8, pady=10)

    conn = get_db_connection()
    cursor = conn.cursor(dictionary=True)
    cursor.execute("""
        SELECT sr.Service_Request_ID, sr.Service_Date, sr.Delivery_Date, sr.Mechanic_ID, sr.Service_Status,
            c.Name AS Customer_Name, m.Name AS Mechanic_Name
        FROM SERVICE_REQUEST sr
        JOIN CUSTOMER c ON sr.Customer_ID = c.Customer_ID
        JOIN MECHANIC m ON sr.Mechanic_ID = m.Mechanic_ID
        WHERE sr.Service_Status = 'Completed' AND (sr.Payment_Status IS NULL OR sr.Payment_Status !=
'Paid')
    """)
    services = cursor.fetchall()
    conn.close()

    headers = ["Request ID", "Customer", "Mechanic", "Mech ID", "Service Date", "Delivery Date", "Payment
Status", "Action"]
    for col, h in enumerate(headers):
        tk.Label(grid_frame, text=h, font=("Arial", 10, "bold"), bg="#D6EAF8", width=18).grid(row=1,
column=col)

    for row_idx, s in enumerate(services, start=2):
        tk.Label(grid_frame, text=s["Service_Request_ID"]).grid(row=row_idx, column=0)
        tk.Label(grid_frame, text=s["Customer_Name"]).grid(row=row_idx, column=1)
        tk.Label(grid_frame, text=s["Mechanic_Name"]).grid(row=row_idx, column=2)
        tk.Label(grid_frame, text=s["Mechanic_ID"]).grid(row=row_idx, column=3)
        tk.Label(grid_frame, text=s["Service_Date"]).grid(row=row_idx, column=4)
        tk.Label(grid_frame, text=s["Delivery_Date"]).grid(row=row_idx, column=5)
```

```
        status_var = tk.StringVar(value="Unpaid")  # default
        dropdown = ttk.Combobox(grid_frame, textvariable=status_var, values=["Unpaid", "Paid"], width=12,
state="readonly")
        dropdown.grid(row=row_idx, column=6)

        def make_updater(req_id=s["Service_Request_ID"], status_var=status_var):
            def update():
                new_status = status_var.get()
                try:
                    conn2 = get_db_connection()
                    cur2 = conn2.cursor()
                    cur2.execute("UPDATE SERVICE_REQUEST SET Payment_Status = %s WHERE
Service_Request_ID = %s", (new_status, req_id))
                    conn2.commit()
                    conn2.close()
                    messagebox.showinfo("Updated", f"Marked Service {req_id} as {new_status}")
                    self.show_completed_services()
                except Exception as e:
                    messagebox.showerror("Error", str(e))
            return update

        tk.Button(grid_frame, text="Update", bg="#27AE60", fg="white",
command=make_updater()).grid(row=row_idx, column=7)


    def show_parts_details(self):
        self.clear_content()
        conn = get_db_connection()
        cursor = conn.cursor()

        tk.Label(self.content_frame, text="All Tools", font=("Arial", 14, "bold"), bg="#FDFEFE").pack(pady=5)
        cursor.execute("SELECT Part_ID, Part_name, Total_quantity FROM PARTS")
        parts = cursor.fetchall()
        self.display_table(["Part ID", "Part Name", "Total Qty"], parts)

        tk.Label(self.content_frame, text="Tools Used Per Service", font=("Arial", 14, "bold"),
bg="#FDFEFE").pack(pady=10)
        cursor.execute("""
            SELECT p.Part_name, pu.Quantity_used, pu.Service_ID
            FROM PARTS_USED pu
            JOIN PARTS p ON pu.Part_ID = p.Part_ID
        """)
        usage = cursor.fetchall()
        conn.close()
        self.display_table(["Part Name", "Qty Used", "Service ID"], usage)

    def show_suppliers(self):
        self.clear_content()
        tk.Label(self.content_frame, text="Suppliers List", font=("Arial", 16, "bold"),
bg="#FDFEFE").pack(pady=10)
        conn = get_db_connection()
        cursor = conn.cursor()
        cursor.execute("SELECT Supplier_ID, Name, Contact_Number, Address FROM SUPPLIER")
        data = cursor.fetchall()
        conn.close()
        self.display_table(["Supplier ID", "Name", "Contact", "Address"], data)

    def display_table(self, columns, data):
        tree = ttk.Treeview(self.content_frame, columns=columns, show='headings')
        for col in columns:
            tree.heading(col, text=col)
            tree.column(col, width=180)
        for row in data:
            tree.insert('', tk.END, values=row)
        tree.pack(fill=tk.BOTH, expand=True, padx=10, pady=5)
```

```
    def show_add_service_form(self):
        self.clear_content()
        tk.Label(self.content_frame, text="Add New Service", font=("Arial", 16, "bold"),
bg="#FDFEFE").pack(pady=10)

        name_var = tk.StringVar()
        cost_var = tk.StringVar()

        tk.Label(self.content_frame, text="Service Name:").pack(pady=5)
        tk.Entry(self.content_frame, textvariable=name_var, width=30).pack()

        tk.Label(self.content_frame, text="Service Cost:").pack(pady=5)
        tk.Entry(self.content_frame, textvariable=cost_var, width=30).pack()

        def save_service():
            name = name_var.get().strip()
            try:
                cost = float(cost_var.get())
                conn = get_db_connection()
                cursor = conn.cursor()
                cursor.execute("INSERT INTO SERVICES (ServiceName, Service_Cost) VALUES (%s, %s)", (name,
cost))
                conn.commit()
                conn.close()
                messagebox.showinfo("Success", f"Service '{name}' added.")
                self.show_all_services()
            except ValueError:
                messagebox.showerror("Invalid", "Service cost must be a number.")
            except Exception as e:
                messagebox.showerror("Error", str(e))

        tk.Button(self.content_frame, text="Add Service", bg="#27AE60", fg="white",
command=save_service).pack(pady=10)

    def show_add_mechanic_form(self):
        self.clear_content()
        tk.Label(self.content_frame, text="Add New Mechanic", font=("Arial", 16, "bold"),
bg="#FDFEFE").pack(pady=10)

        name_var = tk.StringVar()
        username_var = tk.StringVar()
        password_var = tk.StringVar()

        tk.Label(self.content_frame, text="Name:").pack(pady=5)
        tk.Entry(self.content_frame, textvariable=name_var, width=30).pack()

        tk.Label(self.content_frame, text="Username:").pack(pady=5)
        tk.Entry(self.content_frame, textvariable=username_var, width=30).pack()

        tk.Label(self.content_frame, text="Password:").pack(pady=5)
        tk.Entry(self.content_frame, textvariable=password_var, width=30, show="*").pack()

        def save_mechanic():
            name = name_var.get().strip()
            user = username_var.get().strip()
            pwd = password_var.get().strip()

            if not all([name, user, pwd]):
                messagebox.showerror("Invalid", "All fields are required.")
                return

            try:
                conn = get_db_connection()
                cursor = conn.cursor()
                cursor.execute("INSERT INTO MECHANIC (Name, Mechanic_User_Name, Mechanic_Password)
VALUES (%s, %s, %s)",
                               (name, user, pwd))
```

```
            conn.commit()
            conn.close()
            messagebox.showinfo("Success", f"Mechanic '{name}' added.")
        except Exception as e:
            messagebox.showerror("Error", str(e))

    tk.Button(self.content_frame, text="Add Mechanic", bg="#27AE60", fg="white",
command=save_mechanic).pack(pady=10)



if __name__ == "__main__":
    root = tk.Tk()
    AdminDashboard(root)  # Replace with actual mechanic ID
    root.mainloop()
```

## Mechanic Dashboard

```
import tkinter as tk
from tkinter import ttk, messagebox
from tkcalendar import DateEntry
import mysql.connector
from datetime import datetime

def get_db_connection():
    return mysql.connector.connect(
        host='141.209.241.91',
        port=3306,
        user='sp2025bis698g6',
        password='warm',
        database='sp2025bis698g6s'
    )

class MechanicDashboard:
    def __init__(self, root, controller=None, role=None, mechanic_id=None):
        self.root = root
        self.controller = controller
        self.role = role
        self.mechanic_id = mechanic_id

        self.root.geometry("1000x600")
        self.root.title("Mechanic Dashboard")

        tk.Label(root, text="Mechanic Dashboard", font=("Arial", 18, "bold")).pack(pady=10)

        button_frame = tk.Frame(root)
        button_frame.pack(side=tk.LEFT, fill=tk.Y, padx=10)

        self.content_frame = tk.Frame(root, bg="#F4F6F7")
        self.content_frame.pack(side=tk.RIGHT, fill=tk.BOTH, expand=True)

        tk.Button(button_frame, text="Services Assigned to Me",bg="#3498DB", fg="white", width=25,
command=self.show_assigned_services).pack(pady=5)
        tk.Button(button_frame, text="All Services",bg="#9B59B6", fg="white", width=25,
command=self.show_all_services).pack(pady=5)
        tk.Button(button_frame, text="Previous Services",bg="#E67E22", fg="white", width=25,
command=self.show_completed_services).pack(pady=5)
        tk.Button(button_frame, text="All Tools Details",bg="#E67E00", fg="white", width=25,
command=self.show_tools_details).pack(pady=5)
        tk.Button(button_frame, text="Logout", bg="Blue", fg="white", width=25,
command=self.logout).pack(pady=5)


    def clear_content(self):
        for widget in self.content_frame.winfo_children():
            widget.destroy()
```

```
    def display_table(self, columns, data):
        tree = ttk.Treeview(self.content_frame, columns=columns, show='headings')
        for col in columns:
            tree.heading(col, text=col)
            tree.column(col, width=150)
        for row in data:
            tree.insert(', tk.END, values=row)
        tree.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

    def show_all_services(self):
        self.clear_content()
        tk.Label(self.content_frame, text="All Services List", font=("Arial", 16, "bold"),
bg="#FDFEFE").pack(pady=10)
        conn = get_db_connection()
        cursor = conn.cursor()
        cursor.execute("""
            SELECT Service_Request_ID, Service_Date, Mechanic_ID, Service_Status, Delivery_Date,
Total_Amount
            FROM SERVICE_REQUEST
            WHERE Mechanic_ID = %s
        """, (self.mechanic_id,))
        rows = cursor.fetchall()
        conn.close()

        self.display_table(
            ["Service ID", "Service Date", "Mechanic ID", "Status", "Delivery Date", "Total Amount"],
            rows
        )

    def logout(self):
        from Login import LoginPage
        self.root.destroy()
        LoginPage().mainloop()

    def show_assigned_services(self):
        self.clear_content()
        #tk.Label(self.content_frame, text="Assigned Services to Me", font=("Arial", 16, "bold"),
bg="#FDFEFE").grid(row=0, column=0, columnspan=7, pady=10)
        conn = get_db_connection()
        cursor = conn.cursor(dictionary=True)

        cursor.execute("""
            SELECT Service_Request_ID, Service_Date, Service_Status, Delivery_Date
            FROM SERVICE_REQUEST
            WHERE Mechanic_ID = %s AND Service_Status IN ('Pending', 'In Progress')
        """, (self.mechanic_id,))
        services = cursor.fetchall()
        #print(f"Fetched {len(services)} assigned services for mechanic {self.mechanic_id}")
        cursor.close()
        conn.close()

        if not services:
            tk.Label(self.content_frame, text="No assigned services to update.").pack()
            return

        headers = ["Request ID", "Service Date", "Current Status", "New Status", "Current Delivery Date", "New
Delivery Date", "Action"]
        for col_idx, header in enumerate(headers):
            tk.Label(self.content_frame, text=header, font=("Arial", 10, "bold")).grid(row=0, column=col_idx,
padx=5, pady=5)

        status_options = ["Pending", "In Progress", "Completed"]

        for row_idx, service in enumerate(services, start=1):
            req_id = service["Service_Request_ID"]
            #tk.Label(self.content_frame, text=req_id).grid(row=row_idx, column=0,bg="#D6EAF8")
```

```
        tk.Label(self.content_frame, text=req_id, bg="#D6EAF8").grid(row=row_idx, column=0)
        tk.Label(self.content_frame, text=service["Service_Date"]).grid(row=row_idx, column=1)
        tk.Label(self.content_frame, text=service["Service_Status"]).grid(row=row_idx, column=2)

        # New status dropdown
        new_status_var = tk.StringVar(value=service["Service_Status"])
        status_menu = ttk.Combobox(self.content_frame, textvariable=new_status_var, values=status_options,
state="readonly", width=15)
        status_menu.grid(row=row_idx, column=3)

        # Current delivery date
        tk.Label(self.content_frame, text=str(service["Delivery_Date"])).grid(row=row_idx, column=4)

        # New delivery date picker
        date_picker = DateEntry(self.content_frame, width=12, background='darkblue', foreground='white',
                        borderwidth=2, date_pattern='yyyy-mm-dd')
        date_picker.set_date(service["Delivery_Date"])
        date_picker.grid(row=row_idx, column=5)

        # Update button
        def make_updater(req_id=req_id, status_var=new_status_var, date_widget=date_picker):
            def update():
                new_status = status_var.get()
                new_date = date_widget.get_date().strftime("%Y-%m-%d")

                try:
                    conn = get_db_connection()
                    cursor = conn.cursor()
                    cursor.execute("""
                        UPDATE SERVICE_REQUEST
                        SET Service_Status = %s, Delivery_Date = %s
                        WHERE Service_Request_ID = %s
                    """, (new_status, new_date, req_id))
                    conn.commit()
                    cursor.close()
                    conn.close()
                    messagebox.showinfo("Success", f"Service {req_id} updated.")
                    self.show_assigned_services()
                except Exception as e:
                    messagebox.showerror("Update Error", str(e))

            return update

        #tk.Button(self.content_frame, text="Update", command=make_updater()).grid(row=row_idx,
column=6, padx=5)
        tk.Button(self.content_frame, text="Update",
command=make_updater(),bg="#27AE60",fg="white",activebackground="#2ECC71",activeforeground="white
").grid(row=row_idx, column=6, padx=5, pady=2)

    def show_completed_services(self):
        self.clear_content()
        tk.Label(self.content_frame, text="Feedback for Completed Services", font=("Arial", 16, "bold"),
bg="#FDFEFE").pack(pady=10)
        conn = get_db_connection()
        cursor = conn.cursor()
        cursor.execute("""
            SELECT sr.Service_Request_ID, sr.Service_Date, f.Rating, f.Customer_feedback
            FROM SERVICE_REQUEST sr
            LEFT JOIN FEEDBACK f ON sr.Service_Request_ID = f.Service_Request_ID
            WHERE sr.Mechanic_ID = %s AND sr.Service_Status = 'Completed'
        """, (self.mechanic_id,))
        rows = cursor.fetchall()
        conn.close()

        if not rows:
            tk.Label(self.content_frame, text="No completed services found.", font=("Arial", 12),
bg="#FDFEFE").pack(pady=20)
```

```
            return

        self.display_table(["Request ID", "Date", "Rating", "Feedback"], rows)

    def show_tools_details(self):
        self.clear_content()

        heading = tk.Label(self.content_frame, text="All Tools & Usage Details", font=("Arial", 16, "bold"),
bg="#FDFEFE", fg="#2C3E50")
        heading.pack(pady=10)

        conn = get_db_connection()
        cursor = conn.cursor()

        # Top Table: All Tools from PARTS
        cursor.execute("""
            SELECT Part_ID, Part_name, Total_quantity FROM PARTS
        """)
        parts = cursor.fetchall()

        top_label = tk.Label(self.content_frame, text="🧰 All Tools Inventory", font=("Arial", 12, "bold"),
fg="#1F618D", bg="#FDFEFE")
        top_label.pack(pady=(10, 0))
        top_table = ttk.Treeview(self.content_frame, columns=["Part ID", "Part Name", "Total Quantity"],
show="headings", height=8)
        for col in ["Part ID", "Part Name", "Total Quantity"]:
            top_table.heading(col, text=col)
            top_table.column(col, width=150)
        for row in parts:
            top_table.insert("", tk.END, values=row)
        top_table.pack(pady=5, fill=tk.X)

        # Bottom Table: Tools used per service
        cursor.execute("""
            SELECT p.Part_name, pu.Quantity_used, pu.Service_ID
            FROM PARTS_USED pu
            JOIN PARTS p ON pu.Part_ID = p.Part_ID
        """)
        parts_used = cursor.fetchall()
        conn.close()

        bottom_label = tk.Label(self.content_frame, text="🔧 Tools Used Per Service", font=("Arial", 12, "bold"),
fg="#AF601A", bg="#FDFEFE")
        bottom_label.pack(pady=(20, 0))
        bottom_table = ttk.Treeview(self.content_frame, columns=["Part Name", "Quantity Used", "Service ID"],
show="headings", height=10)
        for col in ["Part Name", "Quantity Used", "Service ID"]:
            bottom_table.heading(col, text=col)
            bottom_table.column(col, width=160)
        for row in parts_used:
            bottom_table.insert("", tk.END, values=row)
        bottom_table.pack(pady=5, fill=tk.X)
# Example of launching the dashboard directly
if __name__ == "__main__":
    root = tk.Tk()
    MechanicDashboard(root, mechanic_id=27)  # Replace with actual mechanic ID
    root.mainloop()
```