```python
import tkinter as tk
from tkinter import ttk, messagebox
from tkcalendar import Calendar
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import os
import csv

CSV_FILE = "marks_data.csv"
COLUMNS = [
    "Module Code", "Module Name", "CW1 Marks", "CW2 Marks", "CW3 Marks",
    "Student ID", "Student Name", "Gender", "Date of Entry"
]

def initialize_csv():
    if not os.path.exists(CSV_FILE):
        with open(CSV_FILE, mode="w", newline="") as file:
            writer = csv.writer(file)
            writer.writerow(COLUMNS)

def load_data():
    if os.path.exists(CSV_FILE):
        return pd.read_csv(CSV_FILE)
    return pd.DataFrame(columns=COLUMNS)

def save_data(df):
    df.to_csv(CSV_FILE, index=False)

def validate_marks(marks):
    try:
        marks = int(marks)
        return 0 <= marks <= 100
    except ValueError:
        return False

def on_enter(e):
    e.widget["background"] = "#346CB0"
    e.widget["foreground"] = "black"

def on_leave(e):
    e.widget["background"] = e.widget.default_bg
    e.widget["foreground"] = e.widget.default_fg

root = tk.Tk()
root.title("Marks Registration System")
root.geometry("1000x700")
root.configure(bg="white")

header_frame = tk.Frame(root, bg="cornflowerblue", height=80)
header_frame.pack(fill="x")
```

```
header_label = tk.Label(
    header_frame,
    text="Welcome to Marks Registration System",
    font=("Arial", 35, "bold"),
    fg="white",
    bg="cornflowerblue"
)
header_label.pack(pady=20)

notebook_frame = tk.Frame(root, bg='white')
notebook_frame.pack(fill="both")

notebook = ttk.Notebook(root)
notebook.pack(fill="both", expand=True)

initialize_csv()

home_tab = ttk.Frame(notebook)
input_tab = ttk.Frame(notebook)
update_tab = ttk.Frame(notebook)
view_tab = ttk.Frame(notebook)
visualization_tab = ttk.Frame(notebook)

notebook.add(home_tab, text="Home")
notebook.add(input_tab, text="Input Marks")
notebook.add(update_tab, text="Update Marks")
notebook.add(view_tab, text="View Marks")
notebook.add(visualization_tab, text="Visualization")

style = ttk.Style()
style.theme_use("default")
style.configure(
    "TNotebook",
    background="white",
    tabposition="n",
    borderwidth=0
)
style.configure(
    "TNotebook.Tab",
    font=("Arial", 12, "bold"),
    background="royalblue",
    foreground="white",
    padding=(10, 5),
)
style.map(
    "TNotebook.Tab",
    background=[("selected", "mediumblue")],
    foreground=[("selected", "white")],
)

style.configure("TFrame", background="white")

def update_home_tab():
    df = load_data()
    student_count = len(df["Student ID"].unique())
```

```
    module_count = len(df["Module Code"].unique())
    student_label.config(text=f"No. of Students: {student_count}")
    module_label.config(text=f"No. of Modules: {module_count}")

student_label = tk.Label(
    home_tab,
    text="No. of Students: 0",
    font=("Arial", 16, "bold"),
    bg="white",
    fg="black"
)
student_label.pack(pady=10)

module_label = tk.Label(
    home_tab,
    text="No. of Modules: 0",
    font=("Arial", 16, "bold"),
    bg="white",
    fg="black"
)
module_label.pack(pady=10)

update_home_tab()



def submit_input():
    module_code = module_code_entry.get()
    module_name = module_name_entry.get()
    cw1 = cw1_entry.get()
    cw2 = cw2_entry.get()
    cw3 = cw3_entry.get()
    student_id = student_id_entry.get()
    student_name = student_name_entry.get()
    gender = gender_var.get()
    date_of_entry = date_entry.get()

    if not (module_code and module_name and cw1 and cw2 and cw3 and student_id and
student_name and date_of_entry):
        messagebox.showerror("Error", "All fields are required.")
        return

    if not (validate_marks(cw1) and validate_marks(cw2) and validate_marks(cw3)):
        messagebox.showerror("Error", "Marks must be between 0 and 100.")
        return

    df = load_data()

    if not df.empty and ((df["Student ID"] == student_id) & (df["Module Code"] ==
module_code)).any():
        messagebox.showerror("Error", "Duplicate entry for this student and module.")
        return

    new_data = {
        "Module Code": module_code,
```

```
        "Module Name": module_name,
        "CW1 Marks": int(cw1),
        "CW2 Marks": int(cw2),
        "CW3 Marks": int(cw3),
        "Student ID": student_id,
        "Student Name": student_name,
        "Gender": gender,
        "Date of Entry": date_of_entry,
    }

    df = pd.concat([df, pd.DataFrame([new_data])], ignore_index=True)
    save_data(df)
    messagebox.showinfo("Success", "Marks have been Registered successfully!")
    reset_input()
    update_home_tab()

def reset_input():
    module_code_entry.delete(0, tk.END)
    module_name_entry.delete(0, tk.END)
    cw1_entry.delete(0, tk.END)
    cw2_entry.delete(0, tk.END)
    cw3_entry.delete(0, tk.END)
    student_id_entry.delete(0, tk.END)
    student_name_entry.delete(0, tk.END)
    gender_var.set("Male")
    date_entry.delete(0, tk.END)

input_tab_content = tk.Frame(input_tab, bg="white")
input_tab_content.grid(row=0, column=0, sticky="n")
input_tab.columnconfigure(0, weight=1)

input_tab_content.grid_columnconfigure(0, weight=1)
input_tab_content.grid_columnconfigure(1, weight=1)

def open_calendar():
    def select_date():
        date_entry.delete(0, tk.END)
        date_entry.insert(0, cal.get_date())
        cal_frame.destroy()

    cal_frame = tk.Frame(input_tab_content, bg="white")
    cal_frame.place(relx=0.5, rely=0, y=date_button.winfo_y() -120, anchor="n")

    cal = Calendar(cal_frame, selectmode="day")
    cal.pack()

    ttk.Button(cal_frame, text="Choose Date", command=select_date).pack(pady=5)

module_code_entry = ttk.Entry(input_tab_content, width=30)
module_name_entry = ttk.Entry(input_tab_content, width=30)
cw1_entry = ttk.Entry(input_tab_content, width=30)
cw2_entry = ttk.Entry(input_tab_content, width=30)
cw3_entry = ttk.Entry(input_tab_content, width=30)
student_id_entry = ttk.Entry(input_tab_content, width=30)
student_name_entry = ttk.Entry(input_tab_content, width=30)
```

```
gender_var = tk.StringVar(value="Male")
gender_menu = ttk.Combobox(input_tab_content, textvariable=gender_var, values=["Male",
"Female"], state="readonly")
date_entry = ttk.Entry(input_tab_content, width=30)

date_button = tk.Button(
    input_tab_content,
    text="Select Date",
    bg="cornflowerblue",
    fg="black",
    activebackground="lightsteelblue",
    activeforeground="black",
    font=("Arial", 10),
    command=open_calendar
)
date_button.default_bg = "cornflowerblue"
date_button.default_fg = "black"
date_button.bind("<Enter>", on_enter)
date_button.bind("<Leave>", on_leave)

submit_button = tk.Button(
    input_tab_content,
    text="Submit",
    bg="cornflowerblue",
    fg="black",
    activebackground="lightsteelblue",
    activeforeground="black",
    font=("Arial", 12),
    command=submit_input
)
submit_button.default_bg = "cornflowerblue"
submit_button.default_fg = "black"
submit_button.bind("<Enter>", on_enter)
submit_button.bind("<Leave>", on_leave)

reset_button = tk.Button(
    input_tab_content,
    text="Reset",
    bg="darkgrey",
    fg="black",
    activebackground="lightgrey",
    activeforeground="white",
    font=("Arial", 12),
    command=reset_input
)
reset_button.default_bg = "darkgrey"
reset_button.default_fg = "black"
reset_button.bind("<Enter>", on_enter)
reset_button.bind("<Leave>", on_leave)

# Layout
widgets = [
    ("Module Code", module_code_entry),
    ("Module Name", module_name_entry),
    ("CW1 Marks", cw1_entry),
    ("CW2 Marks", cw2_entry),
```

```
    ("CW3 Marks", cw3_entry),
    ("Student ID", student_id_entry),
    ("Student Name", student_name_entry),
    ("Gender", gender_menu),
    ("Date of Entry", date_entry),
]

for i, (label_text, widget) in enumerate(widgets):
    tk.Label(
        input_tab_content,
        text=label_text,
        bg="white",
        fg="black",
        font=("Arial", 12, "bold")
    ).grid(row=i, column=0, padx=10, pady=5, sticky="e")
    widget.grid(row=i, column=1, padx=10, pady=5, sticky="w")

# Position Buttons
date_button.grid(row=8, column=2, padx=10, pady=5)
submit_button.grid(row=len(widgets), column=0, padx=10, pady=10)
reset_button.grid(row=len(widgets), column=1, padx=10, pady=10)

# Update Marks Tab
def search_update():
    student_id = update_student_id_entry.get()

    if not student_id:
        messagebox.showerror("Error", "Student ID is required.")
        return

    df = load_data()
    record = df[df["Student ID"] == student_id]

    if record.empty:
        messagebox.showerror("Error", "Record not found.")
        return

    update_student_name_label.grid()
    update_student_name_entry.grid()
    update_module_code_label.grid()
    update_module_code_entry.grid()
    update_cw1_label.grid()
    update_cw1_entry.grid()
    update_cw2_label.grid()
    update_cw2_entry.grid()
    update_cw3_label.grid()
    update_cw3_entry.grid()
    update_button.grid()

    update_student_name_entry.delete(0, tk.END)
    update_module_code_entry.delete(0, tk.END)
    update_cw1_entry.delete(0, tk.END)
    update_cw2_entry.delete(0, tk.END)
    update_cw3_entry.delete(0, tk.END)
```

```
    update_student_name_entry.insert(0, record.iloc[0]["Student Name"])
    update_module_code_entry.insert(0, record.iloc[0]["Module Code"])
    update_cw1_entry.insert(0, record.iloc[0]["CW1 Marks"])
    update_cw2_entry.insert(0, record.iloc[0]["CW2 Marks"])
    update_cw3_entry.insert(0, record.iloc[0]["CW3 Marks"])
    update_student_name_entry.configure(state="readonly")
    update_module_code_entry.configure(state="readonly")

def update_record():
    student_id = update_student_id_entry.get()
    cw1 = update_cw1_entry.get()
    cw2 = update_cw2_entry.get()
    cw3 = update_cw3_entry.get()

    if not (validate_marks(cw1) and validate_marks(cw2) and validate_marks(cw3)):
        messagebox.showerror("Error", "Marks must be between 0 and 100.")
        return

    df = load_data()
    index = df[df["Student ID"] == student_id].index

    if index.empty:
        messagebox.showerror("Error", "Record not found.")
        return

    df.loc[index, "CW1 Marks"] = int(cw1)
    df.loc[index, "CW2 Marks"] = int(cw2)
    df.loc[index, "CW3 Marks"] = int(cw3)
    save_data(df)
    messagebox.showinfo("Success", "Marks has been updated successfully!")
    update_home_tab()

update_tab_content = tk.Frame(update_tab, bg="white")
update_tab_content.grid(row=0, column=0, sticky="n")
update_tab.columnconfigure(0, weight=1)
update_tab.rowconfigure(0, weight=1)

update_tab_content.grid_columnconfigure(0, weight=1)
update_tab_content.grid_columnconfigure(1, weight=1)

update_student_id_label = tk.Label(
    update_tab_content,
    text="Student ID",
    bg="white",
    fg="black",
    font=("Arial", 12, "bold")
)
update_student_id_entry = ttk.Entry(update_tab_content, width=30)
search_button = tk.Button(
    update_tab_content,
    text="Search",
    bg="cornflowerblue",
    fg="black",
    activebackground="lightsteelblue",
    activeforeground="black",
```

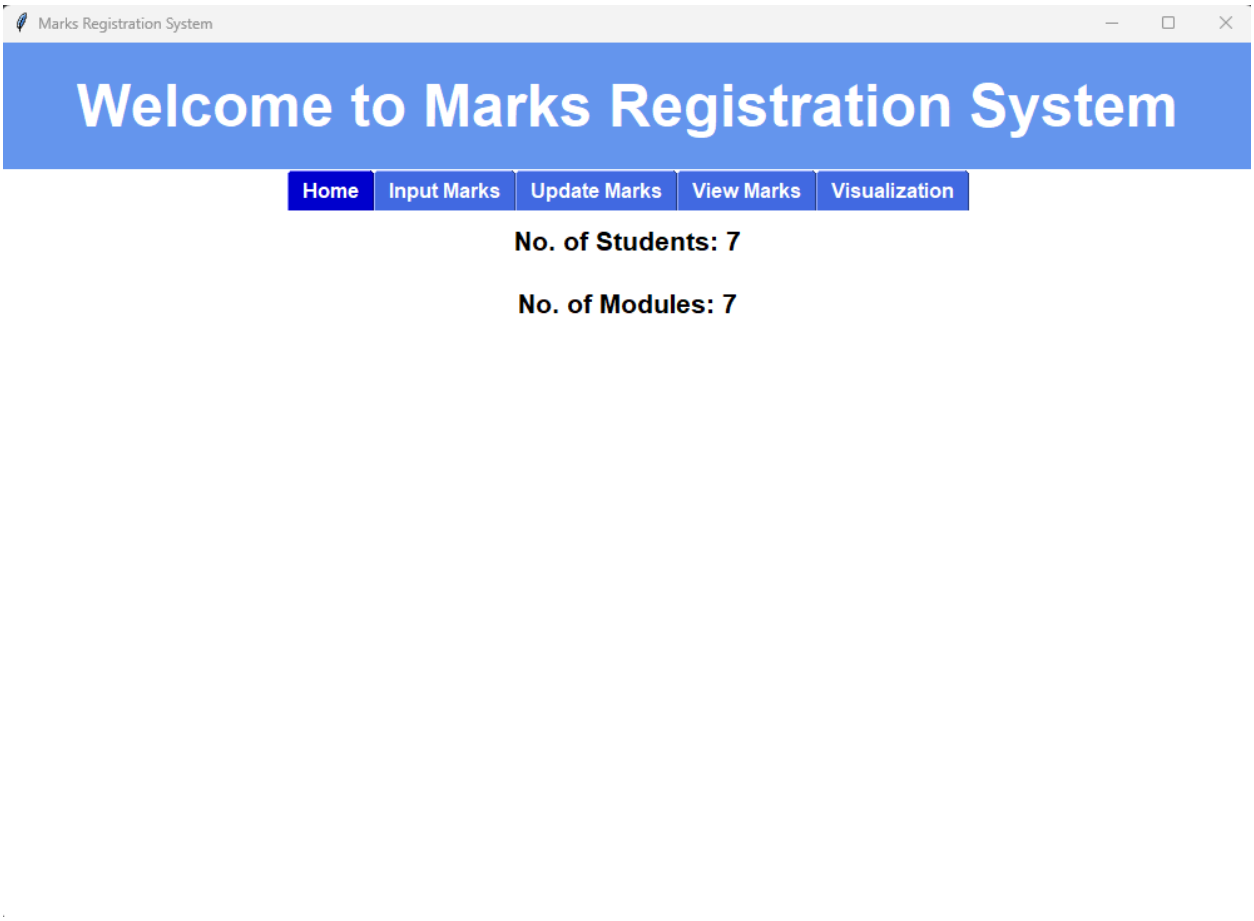
```
    font=("Arial", 12),
    command=search_update
)
search_button.default_bg = "cornflowerblue"
search_button.default_fg = "black"
search_button.bind("<Enter>", on_enter)
search_button.bind("<Leave>", on_leave)

update_student_name_label = tk.Label(
    update_tab_content,
    text="Student Name",
    bg="white",
    fg="black",
    font=("Arial", 12, "bold")
)
update_student_name_entry = ttk.Entry(update_tab_content, width=30)

update_module_code_label = tk.Label(
    update_tab_content,
    text="Module Code",
    bg="white",
    fg="black",
    font=("Arial", 12, "bold")
)
update_module_code_entry = ttk.Entry(update_tab_content, width=30)

update_cw1_label = tk.Label(
    update_tab_content,
    text="CW1 Marks",
    bg="white",
    fg="black",
    font=("Arial", 12, "bold")
)
update_cw1_entry = ttk.Entry(update_tab_content, width=30)

update_cw2_label = tk.Label(
    update_tab_content,
    text="CW2 Marks",
    bg="white",
    fg="black",
    font=("Arial", 12, "bold")
)
update_cw2_entry = ttk.Entry(update_tab_content, width=30)

update_cw3_label = tk.Label(
    update_tab_content,
    text="CW3 Marks",
    bg="white",
    fg="black",
    font=("Arial", 12, "bold")
)
update_cw3_entry = ttk.Entry(update_tab_content, width=30)

update_button = tk.Button(
    update_tab_content,
    text="Update",
```

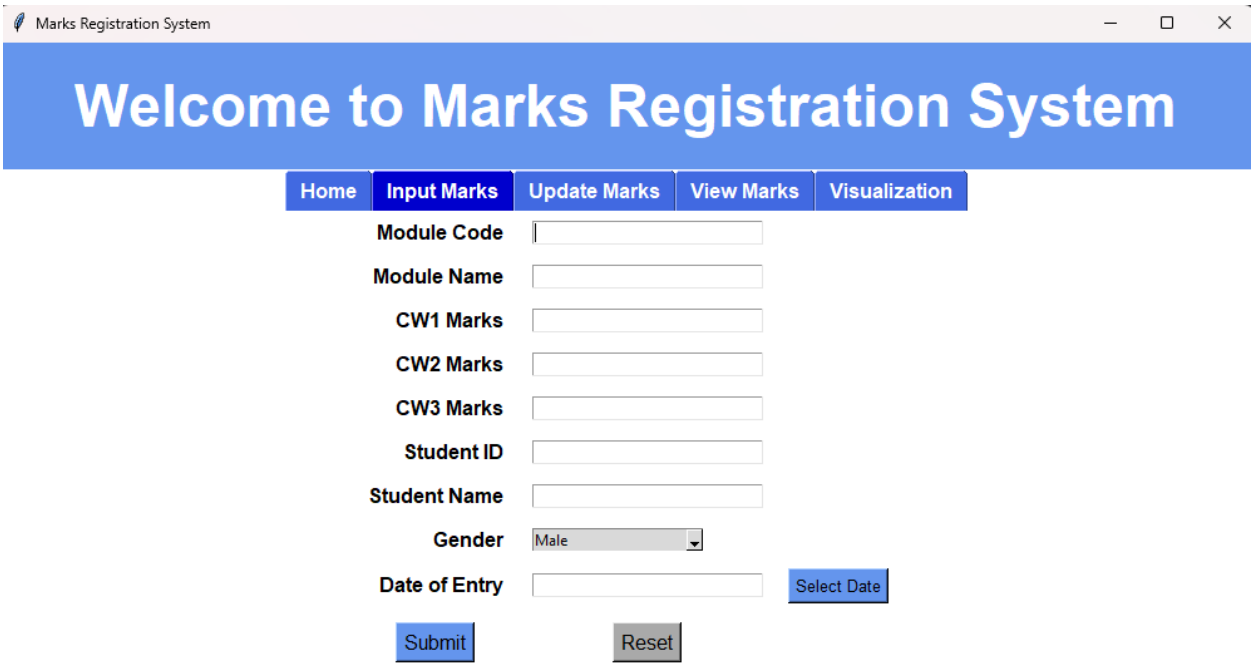
```
    bg="cornflowerblue",
    fg="black",
    activebackground="lightsteelblue",
    activeforeground="black",
    font=("Arial", 12),
    command=update_record
)
update_button.default_bg = "cornflowerblue"
update_button.default_fg = "black"
update_button.bind("<Enter>", on_enter)
update_button.bind("<Leave>", on_leave)

update_student_id_label.grid(row=0, column=0, padx=10, pady=5, sticky="e")
update_student_id_entry.grid(row=0, column=1, padx=10, pady=5, sticky="w")
search_button.grid(row=1, column=1, padx=10, pady=10)

update_student_name_label.grid(row=2, column=0, padx=10, pady=5, sticky="e")
update_student_name_entry.grid(row=2, column=1, padx=10, pady=5, sticky="w")
update_module_code_label.grid(row=3, column=0, padx=10, pady=5, sticky="e")
update_module_code_entry.grid(row=3, column=1, padx=10, pady=5, sticky="w")
update_cw1_label.grid(row=4, column=0, padx=10, pady=5, sticky="e")
update_cw1_entry.grid(row=4, column=1, padx=10, pady=5, sticky="w")
update_cw2_label.grid(row=5, column=0, padx=10, pady=5, sticky="e")
update_cw2_entry.grid(row=5, column=1, padx=10, pady=5, sticky="w")
update_cw3_label.grid(row=6, column=0, padx=10, pady=5, sticky="e")
update_cw3_entry.grid(row=6, column=1, padx=10, pady=5, sticky="w")
update_button.grid(row=7, column=1, padx=10, pady=10)

update_student_name_label.grid_remove()
update_student_name_entry.grid_remove()
update_module_code_label.grid_remove()
update_module_code_entry.grid_remove()
update_cw1_label.grid_remove()
update_cw1_entry.grid_remove()
update_cw2_label.grid_remove()
update_cw2_entry.grid_remove()
update_cw3_label.grid_remove()
update_cw3_entry.grid_remove()
update_button.grid_remove()

def view_records():
    student_id = view_student_id_entry.get()

    if not student_id:
        messagebox.showerror("Error", "Student ID is required.")
        return

    df = load_data()
    records = df[df["Student ID"] == student_id]

    if records.empty:
        messagebox.showinfo("Info", "No records found for the given Student ID.")
        return

    for row in tree.get_children():
```

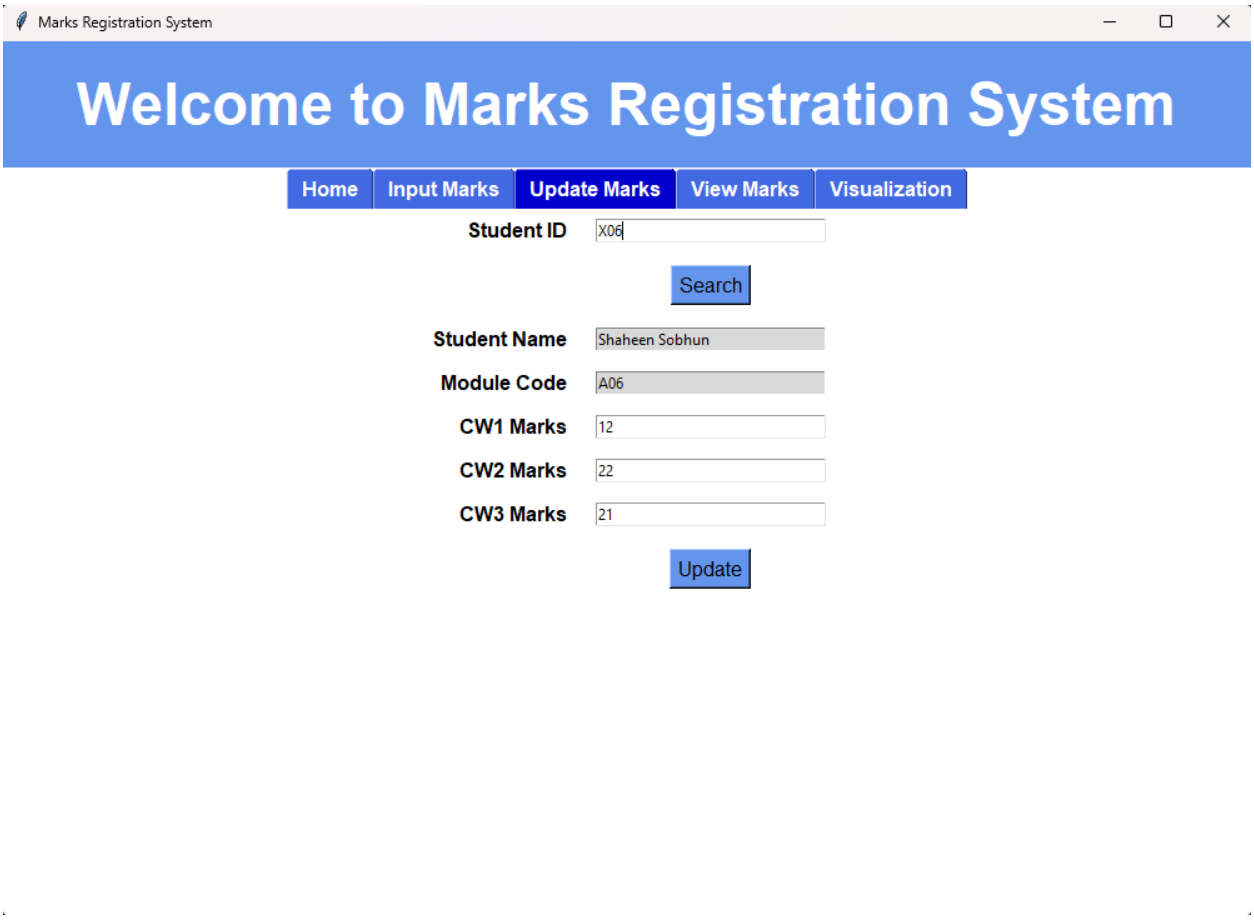
```
        tree.delete(row)

    for _, record in records.iterrows():
        total = record["CW1 Marks"] + record["CW2 Marks"] + record["CW3 Marks"]
        tree.insert("", tk.END, values=(
            record["Module Code"],
            record["Student Name"],
            record["CW1 Marks"],
            record["CW2 Marks"],
            record["CW3 Marks"],
            total
        ))

view_tab_content = tk.Frame(view_tab, bg="white")
view_tab_content.grid(row=0, column=0, sticky="n")
view_tab.columnconfigure(0, weight=1)
view_tab.rowconfigure(0, weight=1)

view_tab_content.grid_columnconfigure(0, weight=1)
view_tab_content.grid_columnconfigure(1, weight=1)

view_student_id_label = tk.Label(
    view_tab_content,
    text="Student ID",
    bg="white",
    fg="black",
    font=("Arial", 12, "bold")
)
view_student_id_entry = ttk.Entry(view_tab_content, width=30)
view_button = tk.Button(
    view_tab_content,
    text="View",
    bg="cornflowerblue",
    fg="black",
    activebackground="lightsteelblue",
    activeforeground="black",
    font=("Arial", 12),
    command=view_records
)
view_button.default_bg = "cornflowerblue"
view_button.default_fg = "black"
view_button.bind("<Enter>", on_enter)
view_button.bind("<Leave>", on_leave)

view_student_id_label.grid(row=0, column=0, padx=10, pady=5, sticky="e")
view_student_id_entry.grid(row=0, column=1, padx=10, pady=5, sticky="w")
view_button.grid(row=1, column=1, padx=10, pady=10, sticky="w")

style = ttk.Style()
style.configure(
    "Treeview.Heading",
   font=("Arial", 12, "bold"),
   background="blue",
   foreground="white",
   padding=(5, 5)
)
```

```
style.configure(
    "Treeview",
    background="white",
   foreground="black",
   rowheight=25,
   fieldbackground="white"
)
style.map(
    "Treeview",
    background=[("selected", "lightsteelblue")],
   foreground=[("selected", "black")]
)

columns = ("Module Code", "Student Name", "CW1 Marks", "CW2 Marks", "CW3 Marks", "Total
Marks")
tree = ttk.Treeview(view_tab_content, columns=columns, show="headings",
style="Treeview")

for col in columns:
    tree.heading(col, text=col, anchor='center')
    tree.column(col, width=150, anchor='center')

tree.grid(row=2, column=0, columnspan=2, pady=10)

def visualization():
    df = load_data()

    if df.empty:
        messagebox.showinfo("Info", "No data available for visualization.")
        return

    for widget in visualization_tab.winfo_children():
        widget.destroy()

    custom_colors = ['blue', 'mediumblue', 'royalblue']

    visualization_frame = tk.Frame(visualization_tab, bg="white")
    visualization_frame.grid(row=0, column=0, padx=10, pady=10, sticky="nsew")
    visualization_tab.grid_columnconfigure(0, weight=1)
   visualization_tab.grid_rowconfigure(0, weight=1)

   visualization_frame.grid_columnconfigure(0, weight=1)
   visualization_frame.grid_columnconfigure(1, weight=1)

   total_cw1 = df["CW1 Marks"].sum()
    total_cw2 = df["CW2 Marks"].sum()
    total_cw3 = df["CW3 Marks"].sum()

    pie_fig, pie_ax = plt.subplots(figsize=(5, 5))
    pie_ax.pie(
        [total_cw1, total_cw2, total_cw3],
        labels=["CW1", "CW2", "CW3"],
        autopct="%1.1f%%",
```

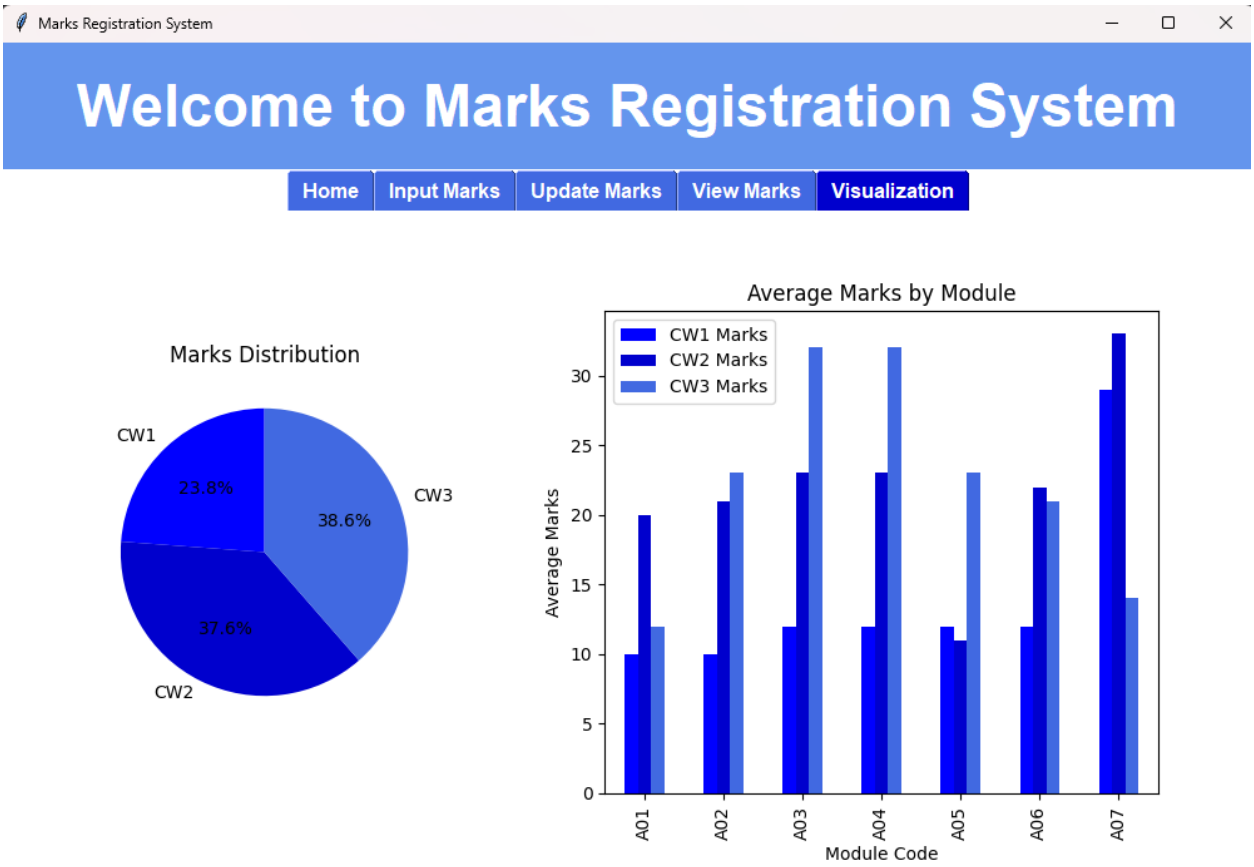
```
        startangle=90,
        colors=custom_colors
    )
    pie_ax.set_title("Marks Distribution")

    pie_canvas = FigureCanvasTkAgg(pie_fig, master=visualization_frame)
    pie_canvas.draw()
    pie_canvas.get_tk_widget().grid(row=0, column=0, padx=10, pady=10, sticky="n")

    avg_marks = df.groupby("Module Code")[["CW1 Marks", "CW2 Marks", "CW3
Marks"]].mean()

    bar_fig, bar_ax = plt.subplots(figsize=(7, 5))
    avg_marks.plot(kind="bar", ax=bar_ax, color=custom_colors)
    bar_ax.set_title("Average Marks by Module")
    bar_ax.set_ylabel("Average Marks")
    bar_ax.set_xlabel("Module Code")

    bar_canvas = FigureCanvasTkAgg(bar_fig, master=visualization_frame)
    bar_canvas.draw()
    bar_canvas.get_tk_widget().grid(row=0, column=1, padx=10, pady=10, sticky="n")



visualization()

try:
    root.mainloop()
except KeyboardInterrupt:
    pass
```


Marks Registration System
Welcome to Marks Registration System
Home
Input Marks
Update Marks
View Marks
Visualization
No. of Students: 7
No. of Modules: 7


Marks Registration System
Welcome to Marks Registration System
Home
Input Marks
Update Marks
View Marks
Visualization
Module Code
Module Name
CW1 Marks
CW2 Marks
CW3 Marks
Student ID
Student Name
Gender
Male
Date of Entry
Select Date
Submit
Reset


Marks Registration System
Welcome to Marks Registration System
Home
Input Marks
Update Marks
View Marks
Visualization
Student ID
X06
Search
Student Name
Shaheen Sobhun
Module Code
A06
CW1 Marks
12
CW2 Marks
22
CW3 Marks
21
Update


Marks Registration System
Welcome to Marks Registration System
Home
Input Marks
Update Marks
View Marks
Visualization
Student ID
X01
View
Module Code
Student Name
CW1 Marks
CW2 Marks
CW3 Marks
Total Marks
A01
Seif Baichoo
10
20
12
42



marks_data.csv

```
Module Code,Module Name,CW1 Marks,CW2 Marks,CW3 Marks,Student ID,Student Name,Gender,Date of Entry
A01,ICT,10,20,12,X01,Seif Baichoo,Male,07/02/2024
A02,IOT,10,21,23,X02,Ajmhul Baicho0,Male,25/05/2023
A03,BDA,12,23,32,X03,Farah Baichoo,Female,1/8/25
A04,Python,12,23,32,X04,Widaad Googoolee,Female,1/11/25
A05,Networking,12,11,23,X05,Haroon Rasheed,Male,9/13/24
A06,Electronics,12,22,21,X06,Shaheen Sobhun,Female,7/19/24
A07,DCY9,29,33,14,X07,Sara Young,Female,4/20/25
```