ACKNOWLEDGEMET

I am really grateful for the opportunity of this project and sincerely thank my computer teacher, Mrs. Ananthy for contributing her valuable time and effort to help me with this project. Her suggestions and feedback helped me a lot to improve the project quality from the implementation. I would also like to thank our school principal, Mrs. D V Lavanya for providing me the necessary facilities towards the completion of my project on time. Secondly, I would like to thank my parents, family member and friends who supported me and guided me throughout with whose I help, I was able to complete the project on time.

M. Navin Muthu

Xll A

ABSTRACT

A to do list is a list of tasks that need to be completed and organised before a specific time. The goal of this project is to design a simple program for people to keep track of their status, manage their task and how they spend time in the process of doing those tasks and how productive that time is. It can help set some constraints on social media to reduce distraction and track the time we spend working on the to-do items. When we have a better sense of the estimated time we'll need to spend on our tasks, along with the validated time spent on the items for reference or personal/team reviews, we are able to manage our daily routines more efficiently. Our project acts as an organising tool for people who desperately need to manage and organise their task from the comfort of their pockets .We believe this will help many people in their daily lives and create a change of lifestyle.

CONTENTS

REQUIRMENT ANALYSIS

HARDWARE REQUIRMENTS:

- Standard computer
- Keyboard
- Mouse

SOFTWARE REQUIRMENTS:

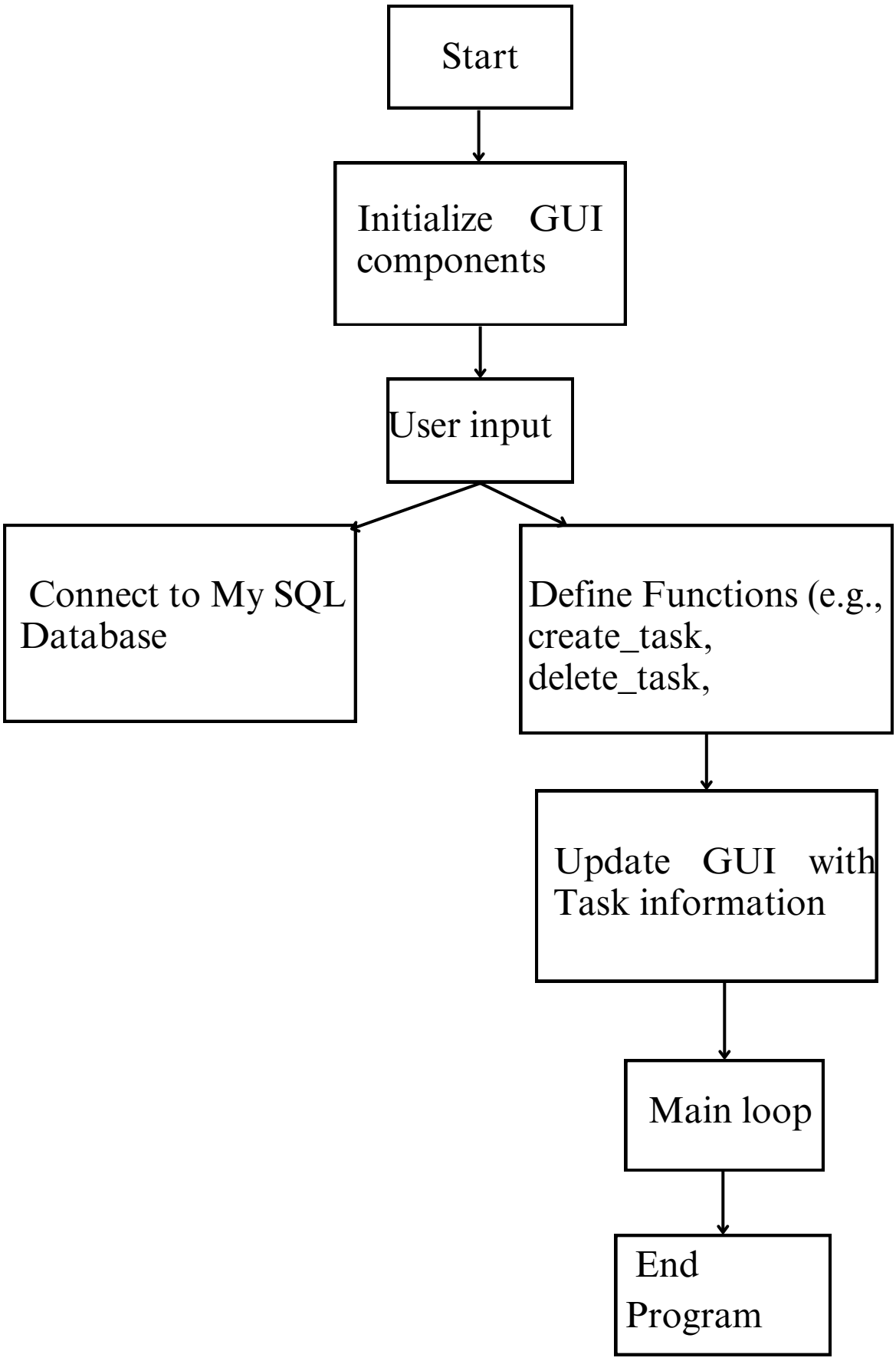
- Windows , Mac or Linux
- Os Python
- Tkinter
- SQL

APPLICATIONS

- Our project can be used by a common person or other officials and business people.
- Our project can be used in corporate companies for
- employees to schedule their tasks and bring fruitful results to their company.
- Our project can be used as an alternative to
- alarms,calenders and other softwares
- Our project improves the productivity of a person. Can be used by school children to aged people.. It can be used in education institutions for their students.
- It can also be used by anyone as it is user friendly

DESIGN

BLOCK DIAGRAM:-



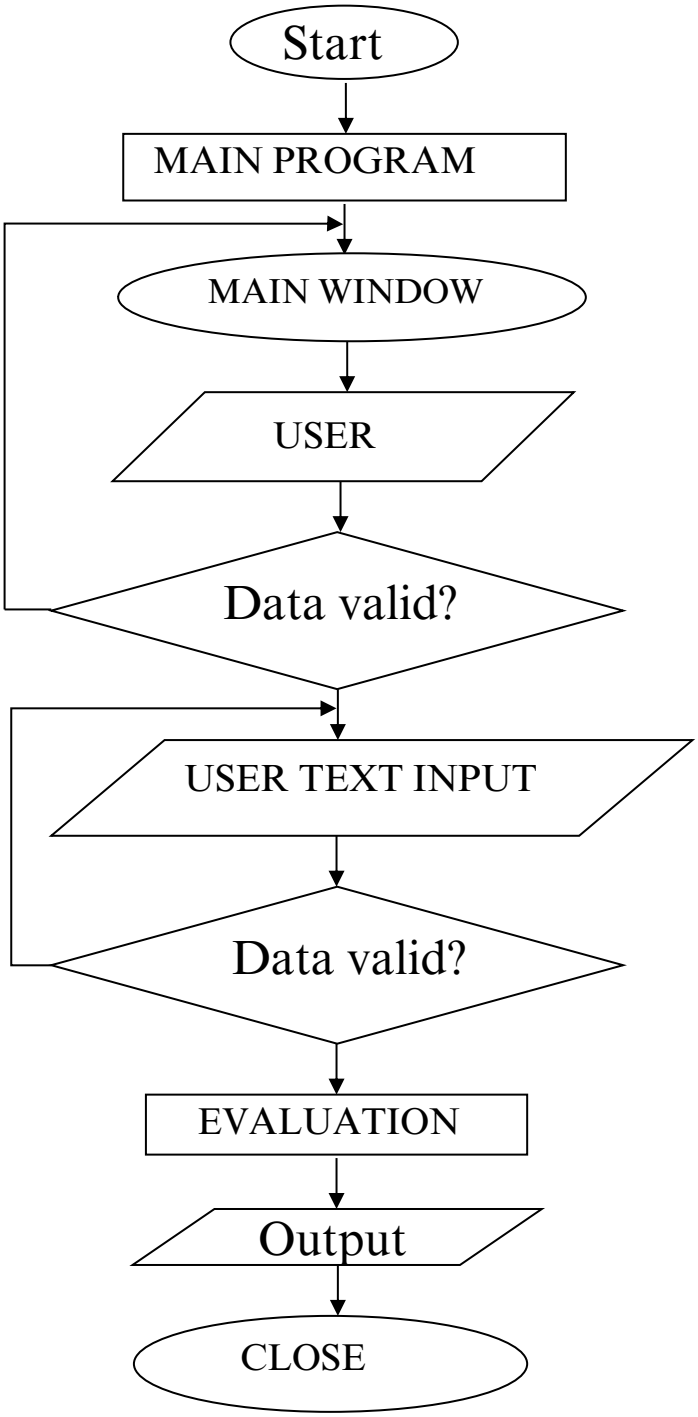
EXPLANATION

First the Main Window is shown where all the entry fields are present. The program recieves the data input by the user and then checks for its validity. It then proceeds to process the input. At this same time , the program also stores the information input by the user in the database. The user can attempt to input the required data of the program and then the program can check the validity of it. It then evaluates the recieved input and also shows the result to the user in the form of a table.

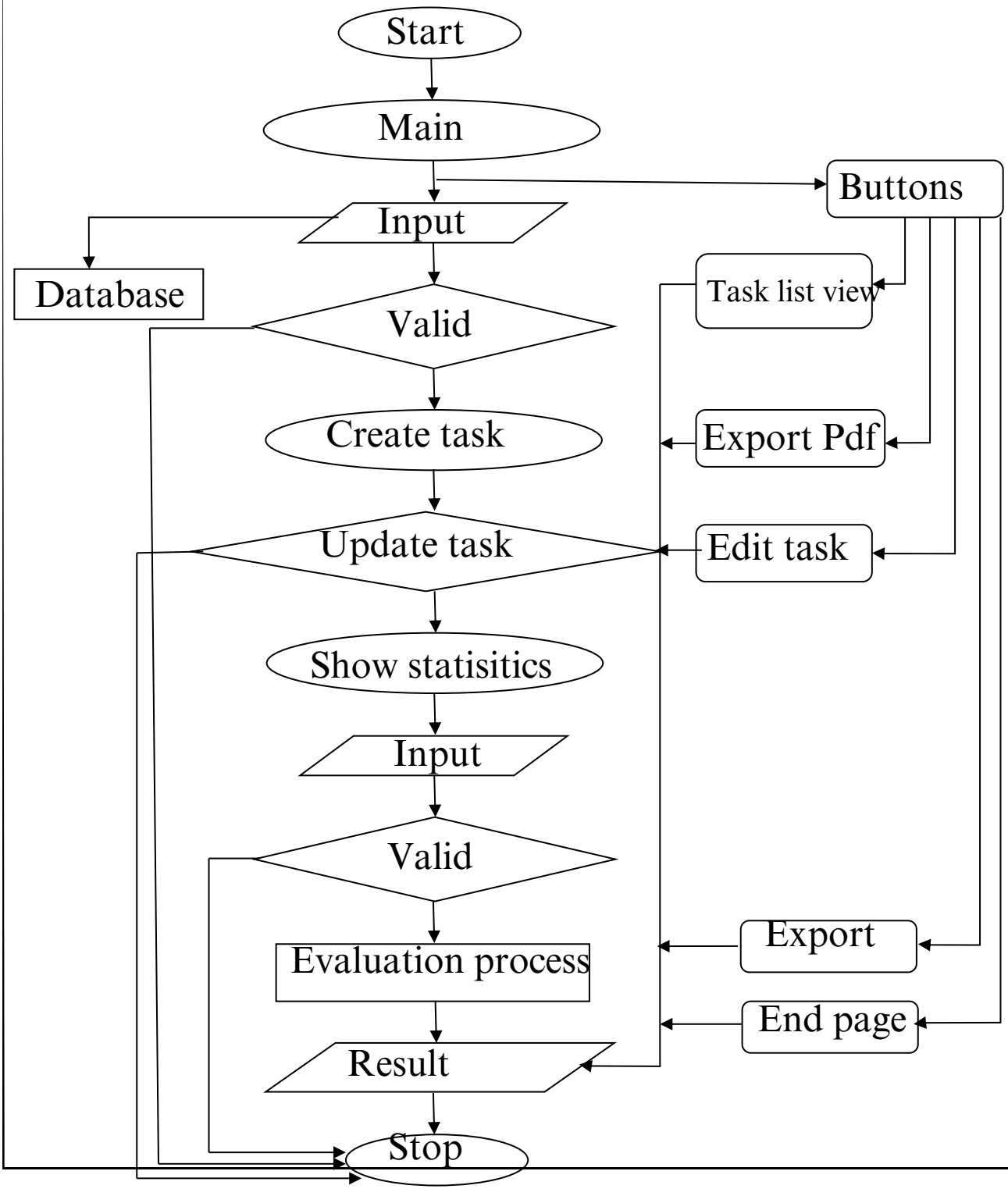
The result is also then stored in the database in the appropriate field. The program then terminates itself .

FLOW CHART

PROJECT FLOW:-



PROGRAM FLOW



ALGORITHM

Step 1 : User enters all their task details and submits it.

Step 2 : The task details are stored in the Database and the user can access other features , simultaneously.

Step 3 : From any of the available options , the user can choose the options they want to apply.

Step 4 : This then applies the features and closes.

Step 5 :delete and edit option is shown along with the options to choose.

Step 6 : After the user enters all the tasks, when the add task button is clicked, the program evaluates the received responses and shows the user their result.

Step 7 : While the result is being shown, the same is also being stored in the database alongside the user's details.

SOURCE CODE

```
import tkinter as tk
from tkinter import ttk
from tkinter import simpledialog, messagebox
from tkcalendar import DateEntry
from ttkthemes import ThemedStyle
from PIL import Image, ImageTk
import mysql.connector
import csv
from reportlab.lib.pagesizes import letter
from reportlab.lib import colors
from reportlab.platypus import SimpleDocTemplate, Table, TableStyle,
Paragraph from reportlab.lib.styles import getSampleStyleSheet
import calendar
import re
from tkinter import Scrollbar

# Create the main application window
root = tk.Tk()
root.title("Task Manager")
# Replace these values with your MySQL credentials
MYSQL_HOST = "localhost"
MYSQL_USER = "root"
MYSQL_PASSWORD = "navin123457"
MYSQL_DATABASE = "task_manager"

def create_task():
title = title_entry.get()
description = description_entry.get("1.0", "end-1c")
due_date = due_date_entry.get()
priority = priority_var.get()
status = status_var.get()

if not title:
messagebox.showerror("Error", "Title cannot be empty.")
return
if not description:
```

```
 messagebox.showerror("Error", "Description cannot be empty.")
return

if not due_date:
messagebox.showerror("Error", "Due date cannot be empty.")
return
# Check if due date is in yyyy-mm-dd format using regular expression
if not re.match(r'^\d{4}-\d{2}-\d{2}$', due_date):
messagebox.showerror("Error", "Due date format should be yyyy-mm-dd.")
return

try:
priority = int(priority)
if priority not in [1, 2, 3]:
raise ValueError
except ValueError:
messagebox.showerror("Error", "Priority must be 1, 2, or 3.")
return
# Connect to MySQL database
connection = mysql.connector.connect(
host=MYSQL_HOST,
user=MYSQL_USER,
password=MYSQL_PASSWORD,
database=MYSQL_DATABASE,
)
# Insert task into the database
cursor = connection.cursor()
query = "INSERT INTO tasks (date_added, title, description, due_date, priority, status) VALUES
(NOW(), %s, %s, %s, %s, %s)"
values = (title, description, due_date, priority, status)
cursor.execute(query, values)
connection.commit()
# Update task list
update_task_list()

# Clear input fields
title_entry.delete(0, tk.END)
description_entry.delete("1.0", tk.END)
due_date_entry.delete(0, tk.END)

# Define priority colors
```

```
PRIORITY_COLORS = {
1: "red", # High priority
2: "yellow", # Medium priority
3: "green", # Low priority
}

def update_task_list():
# Clear existing tasks from the treeview
for item in task_tree.get_children():
task_tree.delete(item)

# Fetch tasks from the database
connection = mysql.connector.connect(
host=MYSQL_HOST,
user=MYSQL_USER,
password=MYSQL_PASSWORD,
database=MYSQL_DATABASE,
)
cursor = connection.cursor()
query = "SELECT id, date_added, title, description, due_date, priority,
status FROM tasks
ORDER BY priority DESC"
cursor.execute(query)
tasks = cursor.fetchall()
# Insert tasks into the treeview with priority color indicators
for task in tasks:
priority = task[5]
priority_color = PRIORITY_COLORS.get(
priority, "black"
) # Default to black if priority is not defined
task_tree.insert("", "end", values=task, tags=(priority_color,))
# Apply tag configuration for priority color indicators
for color in PRIORITY_COLORS.values():
task_tree.tag_configure(color, background=color)

def delete_task():
selected_item = task_tree.selection()
if not selected_item:
return

task_id = task_tree.item(selected_item)["values"][0]
confirmation = messagebox.askyesno(
"Confirm Deletion", "Are you sure you want to delete this task?"
```

```
 )
if confirmation:
# Connect to the database
connection = mysql.connector.connect(
host=MYSQL_HOST,
user=MYSQL_USER,
password=MYSQL_PASSWORD,
database=MYSQL_DATABASE,
)

cursor = connection.cursor()
query = "DELETE FROM tasks WHERE id = %s"
cursor.execute(query, (task_id,))
connection.commit()

# Update task list after deletion
update_task_list()

root.bind("<Control-d>", delete_task)

def search_task():
keyword = search_entry.get()
priority_value = priority_var_search.get()
if priority_value and priority_value.isdigit():
priority = int(priority_value)
else:
priority = 3 # Assign a default value if priority is not selected or not a valid digit

# Connect to the database
connection = mysql.connector.connect(
host=MYSQL_HOST,
user=MYSQL_USER,
password=MYSQL_PASSWORD,
database=MYSQL_DATABASE,
)
cursor = connection.cursor()
# Search tasks based on keyword and priority
query = "SELECT id, date_added, title, description, due_date, priority, status FROM tasks
WHERE title LIKE %s AND priority = %s ORDER BY priority DESC"
cursor.execute(query, (f"%{keyword}%", priority))
tasks = cursor.fetchall()
# Clear existing tasks from the treeview
```

```
 for item in task_tree.get_children():
task_tree.delete(item)

# Insert searched tasks into the treeview
for task in tasks:
task_tree.insert("", "end", values=task)

def clear_search():
search_entry.delete(0, tk.END)
priority_var_search.set(0)
update_task_list()

def fetch_tasks_from_database():
try:
# Connect to the database
connection = mysql.connector.connect(
host=MYSQL_HOST,
user=MYSQL_USER,
password=MYSQL_PASSWORD,
database=MYSQL_DATABASE,
)
# Fetch tasks from the database
cursor = connection.cursor()
query = "SELECT id, date_added, title, description, due_date, priority,
status FROM tasks
ORDER BY priority DESC"
cursor.execute(query)
tasks = cursor.fetchall()

# Close the cursor and connection
cursor.close()
connection.close()
return tasks
except mysql.connector.Error as e:
print(f"Error fetching tasks from the database: {e}")
return []

def export_to_csv():
tasks = fetch_tasks_from_database()

# Convert the list of tuples to a list of dictionaries
task_dicts = []
for task in tasks:
```

```
 task_dict = {
"ID": task[0],
"Date Added": task[1],
"Title": task[2],
"Description": task[3],
"Due Date": task[4],
"Priority": task[5],
"Status": task[6],
}
task_dicts.append(task_dict)

# Write the list of dictionaries to CSV
with open("task_manager_export.csv", "w", newline="") as csvfile:
fieldnames = [
"ID",
"Date Added",
"Title",
"Description",
"Due Date",
"Priority",
"Status",
]
writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
writer.writeheader()
writer.writerows(task_dicts)

def update_task_status(task_id, new_status):
# Connect to MySQL database
connection = mysql.connector.connect(
host=MYSQL_HOST,
user=MYSQL_USER,
password=MYSQL_PASSWORD,
database=MYSQL_DATABASE,
)

# Update task status in the database
cursor = connection.cursor()
query = "UPDATE tasks SET status = %s WHERE id = %s"
values = (new_status, task_id)
cursor.execute(query, values)
connection.commit()

# Close the cursor and connection
cursor.close()
connection.close()
```

```
# ... (your existing code for creating the task_tree)

def mark_as_completed():
selected_item = task_tree.selection()
if not selected_item:
return

task_id = task_tree.item(selected_item)["values"][0]
update_task_status(task_id, "Completed")
update_task_list()

def show_date_picker():
selected_date = DateEntry(root, date_pattern="yyyy-mm-dd")
selected_date.grid(row=2, column=2, padx=5, pady=5)
def set_selected_date():
due_date_entry.delete(0, tk.END)
due_date_entry.insert(0, selected_date.get())
selected_date.grid_forget()

ok_button = tk.Button(root, text="OK", command=set_selected_date)
ok_button.grid(row=2, column=3, padx=5, pady=5)


def mark_as_completed():
selected_item = task_tree.selection()
if not selected_item:
return

# Ask for confirmation
confirmation = messagebox.askyesno(
"Confirm Mark as Completed",
"Are you sure you want to mark this task as completed?",
)
if confirmation:
task_id = task_tree.item(selected_item)["values"][0]
update_task_status(task_id, "Completed")
update_task_list()

root.bind("<Control-c>", mark_as_completed)
```

```
def export_to_pdf():
tasks = fetch_tasks_from_database()

# Create a PDF document
doc = SimpleDocTemplate("task_manager_export.pdf", pagesize=letter)
elements = []
# Create a table to hold task data
data = [
["ID", "Date Added", "Title", "Description", "Due Date", "Priority", "Status"]
]
for task in tasks:
data.append(task)

table = Table(data)
table.setStyle(
TableStyle(
[
("BACKGROUND", (0, 0), (-1, 0), colors.grey),
("TEXTCOLOR", (0, 0), (-1, 0), colors.whitesmoke),
("ALIGN", (0, 0), (-1, -1), "CENTER"),
("FONTNAME", (0, 0), (-1, 0), "Helvetica-Bold"),
("BOTTOMPADDING", (0, 0), (-1, 0), 12),
("BACKGROUND", (0, 1), (-1, -1), colors.beige),
("GRID", (0, 0), (-1, -1), 1, colors.black),
]
)
)

# Add the table to the PDF document
elements.append(table)

# Build the PDF document
doc.build(elements)

def display_statistics():
connection = mysql.connector.connect(
host=MYSQL_HOST,
user=MYSQL_USER,
password=MYSQL_PASSWORD,
database=MYSQL_DATABASE,
)
cursor = connection.cursor()
query = "SELECT COUNT(*) FROM tasks"
cursor.execute(query)
total_tasks = cursor.fetchone()[0]
```

```
query = "SELECT COUNT(*) FROM tasks WHERE status = 'Completed'"
cursor.execute(query)
completed_tasks = cursor.fetchone()[0]

query = "SELECT COUNT(*) FROM tasks WHERE status = 'In Progress'"
cursor.execute(query)
in_progress_tasks = cursor.fetchone()[0]
cursor.close()
connection.close()

statistics_text = f"Total Tasks: {total_tasks}\nCompleted Tasks: {completed_tasks}\nIn Progress
Tasks: {in_progress_tasks}"
messagebox.showinfo("Task Statistics", statistics_text)

def edit_selected_task():
selected_item = task_tree.selection()
if not selected_item:
return

task_id = task_tree.item(selected_item)["values"][0]
connection = mysql.connector.connect(
host=MYSQL_HOST,
user=MYSQL_USER,
password=MYSQL_PASSWORD,
database=MYSQL_DATABASE,
)

cursor = connection.cursor()
query = "SELECT * FROM tasks WHERE id = %s"
cursor.execute(query, (task_id,))
task_data = cursor.fetchone()
cursor.close()
connection.close()

if task_data:
# Open a dialog box for editing task fields
edit_dialog = tk.Toplevel(root)
edit_dialog.title("Edit Task")

# Create and place input fields for each task attribute
tk.Label(edit_dialog, text="Title:").grid(row=0, column=0, padx=5, pady=5)
title_entry_edit = tk.Entry(edit_dialog)
title_entry_edit.insert(0, task_data[2])
title_entry_edit.grid(row=0, column=1, padx=5, pady=5)
```

```
tk.Label(edit_dialog, text="Description:").grid(row=1, column=0, padx=5, pady=5)
description_text_edit = tk.Text(edit_dialog, height=5, width=30)
description_text_edit.insert("1.0", task_data[3])
description_text_edit.grid(row=1, column=1, padx=5, pady=5)

tk.Label(edit_dialog, text="Due Date:").grid(row=2, column=0, padx=5, pady=5)
due_date_entry_edit = tk.Entry(edit_dialog)
due_date_entry_edit.insert(0, task_data[4])
due_date_entry_edit.grid(row=2, column=1, padx=5, pady=5)

tk.Label(edit_dialog, text="Priority:").grid(row=3, column=0, padx=5, pady=5)
priority_var_edit = tk.StringVar(edit_dialog)
priority_combobox_edit = ttk.Combobox(
edit_dialog, textvariable=priority_var_edit, values=[1, 2, 3]
)
priority_combobox_edit.set(task_data[5])
priority_combobox_edit.grid(row=3, column=1, padx=5, pady=5)
tk.Label(edit_dialog, text="Status:").grid(row=4, column=0, padx=5, pady=5)
status_var_edit = tk.StringVar(edit_dialog)
status_combobox_edit = ttk.Combobox(
edit_dialog,
textvariable=status_var_edit,
values=["Not Started", "In Progress", "Completed"],
)
status_combobox_edit.set(task_data[6])
status_combobox_edit.grid(row=4, column=1, padx=5, pady=5)
# Update the task when "Save" button is clicked
def save_edited_task():
new_title = title_entry_edit.get()
new_description = description_text_edit.get("1.0", tk.END)
new_due_date = due_date_entry_edit.get()
new_priority = priority_var_edit.get()
new_status = status_var_edit.get()
connection = mysql.connector.connect(
host=MYSQL_HOST,
user=MYSQL_USER,
password=MYSQL_PASSWORD,
database=MYSQL_DATABASE,
)

cursor = connection.cursor()
query = "UPDATE tasks SET title=%s, description=%s, due_date=%s, priority=%s,
status=%s WHERE id=%s"
cursor.execute(
```

```
 query,
(
new_title,
new_description,
new_due_date,
new_priority,
new_status,
task_id,
),
)
connection.commit()
cursor.close()
connection.close()

edit_dialog.destroy() # Close the edit dialog
update_task_list() # Update the task list in the main window

# Add a "Save" button to save the edited task
save_button = tk.Button(edit_dialog, text="Save", command=save_edited_task)
save_button.grid(row=5, columnspan=2, padx=5, pady=10)

def export_to_csv_popup():
confirmation = messagebox.askyesno(
"Confirm CSV Export", "Are you sure you want to export tasks to CSV?"
)
if confirmation:
export_to_csv()
messagebox.showinfo("CSV Export", "Tasks exported to CSV successfully.")

def export_to_csv_shortcut(event):
export_to_csv_popup()

root.bind("<Control-s>", export_to_csv_shortcut)

def export_to_pdf_popup():
export_to_pdf()
messagebox.showinfo("PDF Export", "Tasks exported to PDF successfully.")

def export_to_pdf_shortcut(event):
export_to_pdf_popup()

root.bind("<Control-p>", export_to_pdf_shortcut)
```

```
# Apply the themed style
style = ThemedStyle(root)
style.set_theme("plastik") # Choose the "plastik" theme

# Set custom fonts
font_title = ("Helvetica", 18, "bold")
font_label = ("Helvetica", 12)
font_button = ("Helvetica", 12, "bold")

# Center the window on the screen
window_width = 800
window_height = 600
screen_width = root.winfo_screenwidth()
screen_height = root.winfo_screenheight()
x = (screen_width - window_width) // 2
y = (screen_height - window_height) // 2
root.geometry(f"{window_width}x{window_height}+{x}+{y}")

# Load and set the background image
background_image = Image.open("Untitled design (6).jpg")
background_image = ImageTk.PhotoImage(background_image)
background_label = tk.Label(root, image=background_image)
background_label.place(x=0, y=0, relwidth=1, relheight=1)




# Task Input Section
input_frame = ttk.Frame(root)
input_frame.place(relx=0.5, rely=0.1, anchor=tk.CENTER)

title_label = tk.Label(root, text="Title:")
title_label.grid(row=0, column=0, padx=5, pady=5)
title_entry = tk.Entry(root)
title_entry.grid(row=0, column=1, padx=5, pady=5)
```

```
description_label = tk.Label(root, text="Description:")
description_label.grid(row=1, column=0, padx=5, pady=5)
description_entry = tk.Text(root, height=5, width=30)
description_entry.grid(row=1, column=1, padx=5, pady=5)

due_date_label = tk.Label(root, text="Due Date:")
due_date_label.grid(row=2, column=0, padx=5, pady=5)
due_date_entry = tk.Entry(root)
due_date_entry.grid(row=2, column=1, padx=5, pady=5)

priority_label = tk.Label(root, text="Priority:")
priority_label.grid(row=3, column=0, padx=5, pady=5)
priority_var = tk.StringVar(root)
priority_combobox = ttk.Combobox(root, textvariable=priority_var, values=[1, 2, 3])
priority_combobox.grid(row=3, column=1, padx=5, pady=5)

status_label = tk.Label(root, text="Status:")
status_label.grid(row=4, column=0, padx=5, pady=5)
status_var = tk.StringVar(root)
status_combobox = ttk.Combobox(
root, textvariable=status_var, values=["Not Started", "In Progress", "Completed"]
)
status_combobox.grid(row=4, column=1, padx=5, pady=5)

add_button = tk.Button(root, text="Add Task", command=create_task)
add_button.grid(row=5, column=0, columnspan=2, padx=5, pady=10)

mark_as_completed_button = tk.Button(
root, text="Mark as Completed", command=mark_as_completed
)
mark_as_completed_button.grid(row=7, column=1, padx=5, pady=5)

date_picker_button = tk.Button(root, text="Select Due Date", command=show_date_picker)
date_picker_button.grid(row=2, column=2, padx=5, pady=5)

# Add a button to export tasks to PDF
export_pdf_button = tk.Button(root, text="Export to PDF", command=export_to_pdf_popup)
export_pdf_button.grid(row=12, column=0, padx=5, pady=5)
# show statistics
stats_button = tk.Button(root, text="Show Statistics", command=display_statistics)
stats_button.grid(row=13, column=2, padx=5, pady=5)
```

```
# Add "Edit Task" Button
edit_task_button = tk.Button(root, text="Edit Task", command=edit_selected_task) edit_task_button.grid(row=14, column=1, padx=5, pady=5)

# Add a button to export tasks to CSV
export_csv_button = tk.Button(root, text="Export to CSV", command=export_to_csv_popup)
export_csv_button.grid(row=13, column=0, padx=5, pady=5)

# Task List View Section
list_frame = ttk.Frame(root)
list_frame.place(relx=0.5, rely=0.35, anchor=tk.CENTER)

task_tree = ttk.Treeview(
root,
columns=(
"ID",
"Date Added",
"Title",
"Description",
"Due Date",
"Priority",
"Status",
),
)

# Create a vertical scrollbar for the Treeview
scrollbar = Scrollbar(root, orient="vertical", command=task_tree.yview)
task_tree.configure(yscrollcommand=scrollbar.set)
# Place the scrollbar on the right side of the Treeview
scrollbar.grid(row=6, column=2, sticky="ns")

task_tree.heading("#1", text="ID")
task_tree.heading("#2", text="Date Added")
task_tree.heading("#3", text="Title")
task_tree.heading("#4", text="Description")
task_tree.heading("#5", text="Due Date")
task_tree.heading("#6", text="Priority")
task_tree.heading("#7", text="Status")
task_tree.column("#1", anchor="center", width=40)
task_tree.column("#2", anchor="center", width=80)
task_tree.column("#3", anchor="w", width=120)
task_tree.column("#4", anchor="w", width=200)
task_tree.column("#5", anchor="center", width=80)
task_tree.column("#6", anchor="center", width=60)
```

```
task_tree.column("#7", anchor="center", width=100)
task_tree.grid(row=6, column=0, columnspan=2, padx=5, pady=5)

# Task Deletion Buttons
delete_button = tk.Button(root, text="Delete Task",
command=delete_task)
delete_button.grid(row=7, column=0, padx=5, pady=5)

# Search Functionality Section
search_frame = ttk.Frame(root)
search_frame.place(relx=0.5, rely=0.92, anchor=tk.CENTER)

search_label = tk.Label(root, text="Search Keyword:")
search_label.grid(row=8, column=0, padx=5, pady=5)
search_entry = tk.Entry(root)
search_entry.grid(row=8, column=1, padx=5, pady=5)

priority_label_search = tk.Label(root, text="Priority:")
priority_label_search.grid(row=9, column=0, padx=5, pady=5)
priority_var_search = tk.StringVar(root)
priority_combobox_search = ttk.Combobox(
root, textvariable=priority_var_search, values=[0, 1, 2, 3]
)
priority_combobox_search.grid(row=9, column=1, padx=5, pady=5)

search_button = tk.Button(root, text="Search", command=search_task)
search_button.grid(row=10, column=0, columnspan=2, padx=5, pady=5)

clear_search_button = tk.Button(root, text="Clear Search",
command=clear_search)
clear_search_button.grid(row=11, column=0, columnspan=2, padx=5,
pady=5)

# export csv
export_csv_button = tk.Button(root, text="Export to CSV",
command=export_to_csv)
export_csv_button.grid(row=12, column=0, columnspan=2, padx=5,
pady=5)

# Delete Task Shortcut (Ctrl + D)
def delete_selected(event):
delete_task()

root.bind("<Control-d>", delete_selected)
```

```
# Mark as Completed Shortcut (Ctrl + C)
def mark_completed(event):
mark_as_completed()

root.bind("<Control-c>", mark_completed)
# Set window icon
window_icon = Image.open("download (1).png")
window_icon = ImageTk.PhotoImage(window_icon)
root.iconphoto(True, window_icon)

# Set window background color
root.configure(bg="#f0f0f0")

# Update task list when the application starts
update_task_list()

root.mainloop()
```

EXECUTION SCREENSHOTS

	ID	Date Added	Title	Description	Due Date	Priority	Status
2	29	19-08-2023	Studying	Chemistry-	20-08-2023	3	In Progress
3	30	19-08-2023	Cycling	Physical Ex	19-08-2023	1	Not Started

FUTURE ENHANCEMENT

There is a lot of scope for improvement and betterment of this project. Some are :

- Improving the Graphical User Interface to facilitate the ease of access for users.
- New features like voice based task addition.
- Seperate pages for default tasks and advanced time managers can be added
- Can be converted to Mobile Application.

BIBLIOGRAPHY AND REFERENCES

Real Python:

https://realpython.com/python-gui-tkinter/Javatpoint :

https://www.javatpoint.com/python-tkinter

StackOverflow :

https://stackoverflow.com/questions/41211060/how-to-add-scrollbar-to-tkinter

https://stackoverflow.com/questions/33770312/tkinter-place-widgets-from-a-list-using-a-loop

TutorialsPoint :

https://www.tutorialspoint.com/python/tk_scrollbar.htm

SECONDARY REFERENCES :-

- Computer science with Python – Sumita Arora
- NCERT Computer science book
- Practical Computer Science Class 12
- Python reference books