TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

THAPATHALI CAMPUS

A Project Proposal

On

Productivity Companion

Submitted By:

Prajesh Shrestha (THA076BCT028)

Rishav Subedi (THA076BCT036)

Santosh Pandey (THA076BCT041)

Ujjwal Paudel (THA076BCT048)

Submitted To:

Department of Electronics and Computer Engineering

Thapathali Campus

Kathmandu, Nepal

Under the Supervision of

Er. Saroj Shakya

July 24, 2021

ABSTRACT

This project is intended to incorporate four productivity tools into a single GUI application using C++ and SFML library. Techniques which were found remarkable in boosting the productivity like to-do list, study planner, session tracker and pomodoro technique are included in this app. These techniques are simple yet very effective. To-do list facilitates users to make a list of daily tasks whereas study planner enables them to create an organized list of topics that they wish to lay their hands on in coming days. Session tracker facilitates users to create study sessions and keep track of the time they've invested in each session. Finally, pomodoro timer enables users to set a focused work session of their desired length.

Keywords: C++, SFML, To-do List, Session Tracker, Study Planner, Pomodoro Technique, Timer, Stop Watch.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATION

API	Application Program Interface
CRUD	Create, Read, Update and Delete
GCC	GNU Compiler Collection
GNU	GNU’s Not Unix
GUI	Graphical User Interface
IDE	Integrated Development Environment
SFML	Simple and Fast Multimedia Library
SQL	Structured Query Language
STL	Standard Template Library
UI	User Interface

1 INTRODUCTION

1.1 Background

Being productive and leaping towards perfection always been a topic of high regard in the present world. We students always work to change for better. Several decades into formal education and we have realized that some practices are better than the other in academics. Practices like to-do listing, session tracking, task scheduling, pomodoro technique, music studying etc. are proven to be more effective. Usage of computer to facilitate and implement these practices is becoming more relevant due to increasing technological reach. So, we aim to compile these practices into a single GUI application using C++ and SFML.

1.2 Motivation

Procrastination is a damaging trait which involves ignoring an unpleasant, but likely more important task, in favor of one that is more enjoyable or easier. It can lead to reduced productivity and cause us to miss out on achieving our goals. Poor organization of tasks can lead to procrastination [1] [2]. Using prioritized To-do lists, creating effective schedules, breaking down a work into time intervals separated by short breaks and working on it etc. are few effective measures to stay organized [3]. In our project we intend to integrate these practices into a single application, which would be of great help for not only us but also for many scholars worldwide.

1.3 Problem Definition

All of us, team members, in some way or the other have used many applications for staying organized. We all realized that a single application does not facilitate every technique we wished for. Most applications were available in mobile phones only and some required us to make a payment for accessing the full-fledged application. So, there was a collective need for an application that compiles all the practices that we wished for and in the meantime eradicating all the other problems that we encountered.

1.4 Project Objectives

This project aims to full the following objective:

- To develop an application that compiles To-do List, Session Tracker, Study Planner and Pomo Timer using C++ and SFML.

1.5 Project Application

The project we purpose offers following applications:

- Task organization and completion – Prioritized To-Do List and Session Tracker are of great help in organizing daily tasks. Moreover, investing periodic efforts using Pomo Timer sets us in a track for completing that task at the very least. At the end of the day, we are organized and we accomplish many things.

- Self-tracking – Session Tracker helps us keep track of our involvement in a specific work and then invest efforts accordingly. Similarly, the To-do List reminds us of the targets we've set for the day and also the targets we've accomplished.

1.6 Scope of Project

Productivity tools are handy to everybody worldwide. It is not that this application is limited within scholars only. Everybody willing to stay organized and focused can make a good use of it. This tool can be used by professionals, students, teachers and any individuals or group to boost their productivity and efficiency. On the down side, it is a desktop application and it cannot be used as mobile application.

2 LITERATURE REVIEW

There are many tools that allow individuals, teams, and organizations to perform their essential day-to-day operations faster and more efficiently. These tools can improve productivity and efficiency simply by improving concentration, assisting to prioritize schedules and work to deadlines. These tools assist to manage the workflow and time expenses with the help of task management and time management. Some tools are focused on improving focus and discipline (Pomodoro Technique), some on scheduling tasks and setting ambitious new personal goals (To-do Lists), some on tracking the time spent and achievements gained on certain task (Session Tracker), and some by keeping records of things that are to be learnt (Study Planner). There are many apps and tools designed to improve the productivity based on above mentioned techniques as Forest, Microsoft To-Do, Todo List, Tide etc.

2.1 Forest

This app is designed on the Pomodoro technique to help you focus on the task. This app takes on a creative approach to help you stay focused. Whenever you want to start a study session, you plant a tree in your in-app "forest". As you work, the tree will begin to grow, but if you pick up your phone and leave the app, your tree dies! You can collect and add more trees to your forest the more you focus. The company even plants real trees in dozens of countries! It's a win-win situation for you and the Earth. [4]

2.2 Microsoft To Do

Microsoft To Do is a cloud-based task management application. It allows users to manage their tasks from a smartphone, tablet and computer. Using Microsoft To Do, you can start each day with a clean outlook on the tasks you need to do, across all your devices. Being a student involves balancing assignments, clubs, errands, and everyday tasks, so this app can help you get each task finished. If a task requires multiple steps, you can add subtasks under each one! When you're done, enjoy the satisfaction of checking it off. [5]

2.3 Tide

Tide is an app that offers you the best focus timer with natural white noise. It is the perfect fusion of productivity, wellness, and beautiful design. The app allows you to set up personalized Pomodoro-focus sessions, with a wide variety of nature sounds, white noise, and calm music. If you leave the app, you'll fail the focus section! The app also features sleep, nap, meditation, and beautiful daily quotes to keep you motivated. [6]

These are few examples among the large number of tools readily available in the market focused on the improvement of productivity. However, there is a lack of platform which brings the features of these great apps together for better and effective implementation of all these techniques. So, we came up with the idea of our very own tool, "Productivity Companion" which is the integration of various features from various productivity apps in a single platform. This tool can be used by professionals, students, teachers and any individuals or group to boost their productivity and efficiency. Person using this tool will surely notice a remarkable improvement in their workflow and time management.

3 PROPOSED SYSTEM ARCHITECTURE

The project ‘Productivity Companion’ features numerous time management methods and organizing techniques displayed distinctly in tabs as sub-apps. The main application comprises of four sub-app namely Pomo Timer, Session Tracker, To-do list and Study Planner. All these apps commonly includes various UI components to make the entire app interactive.

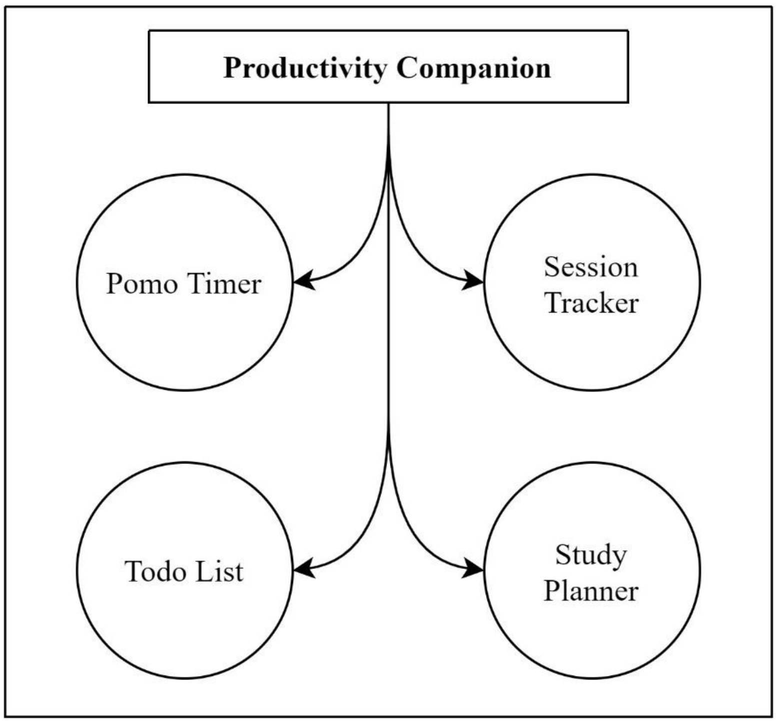


Figure 3.1: Basic Overview

3.1 Individual App System Architecture

3.1.1 Pomo Timer

The prominent *Promodoro Technique* is incorporated via a simple timer with the added feature of rewarding the user for every *pomodoro* (interval). An initial timer of 25 minutes is allowed with higher limit being 120 minutes i.e. 2 hours. A knob that act as a handle sliding in circular path responsive to the user mouse movement is used to set the interval. Points for every *pomodoro* based on the length of interval is rewarded. An additional feature to this app might be the optional legal free copy-righted lo-fi song player to play during the *pomodoro*.

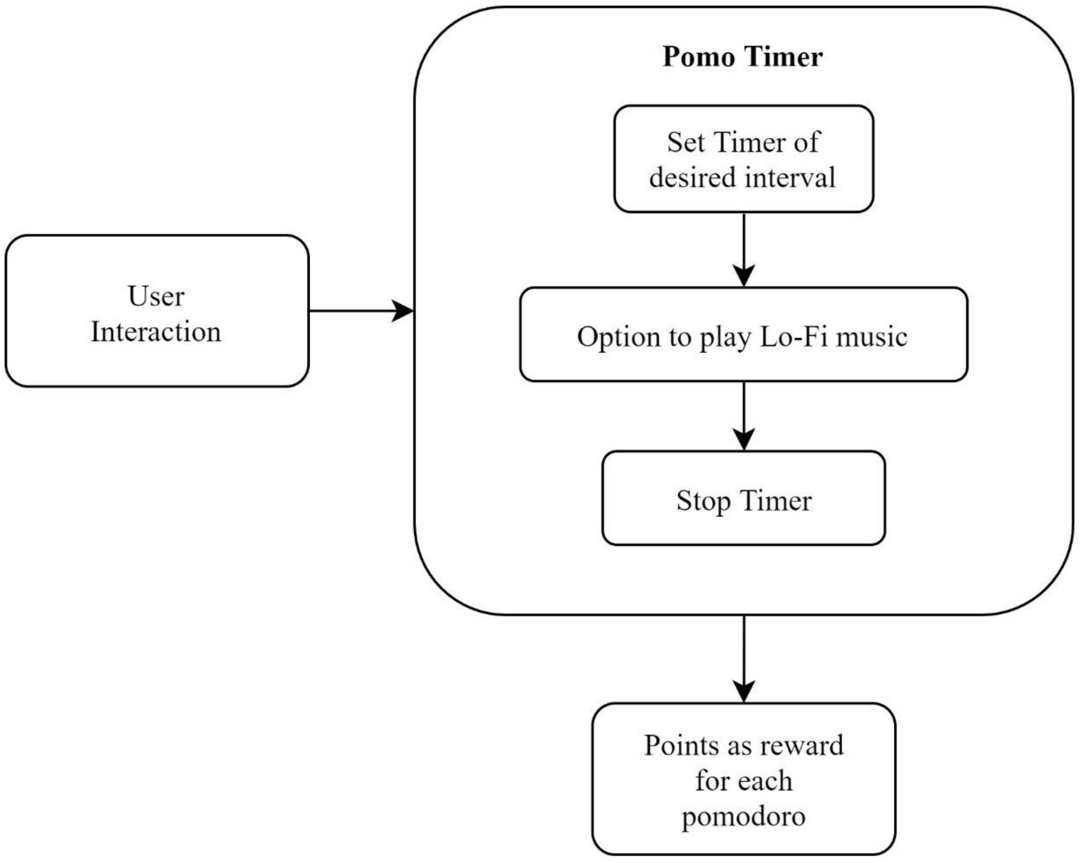

Pomo Timer
Set Timer of desired interval
User Interaction
Option to play Lo-Fi music
Stop Timer
Points as reward for each pomodoro

Figure 3.2: Pomo Timer Architecture

3.1.2 Session Tracker

Session tracking app that keeps track of time stamps and duration for a session. User is provided with the option to manually type and create new session / project in the initial window. The created session tab consist of relevant parameters that best describe the session itself. This session tab opens in a new interactive session window which features a stop-watch that keeps track of the whole time they have invested in the project. On exiting, the model from the related database is updated.

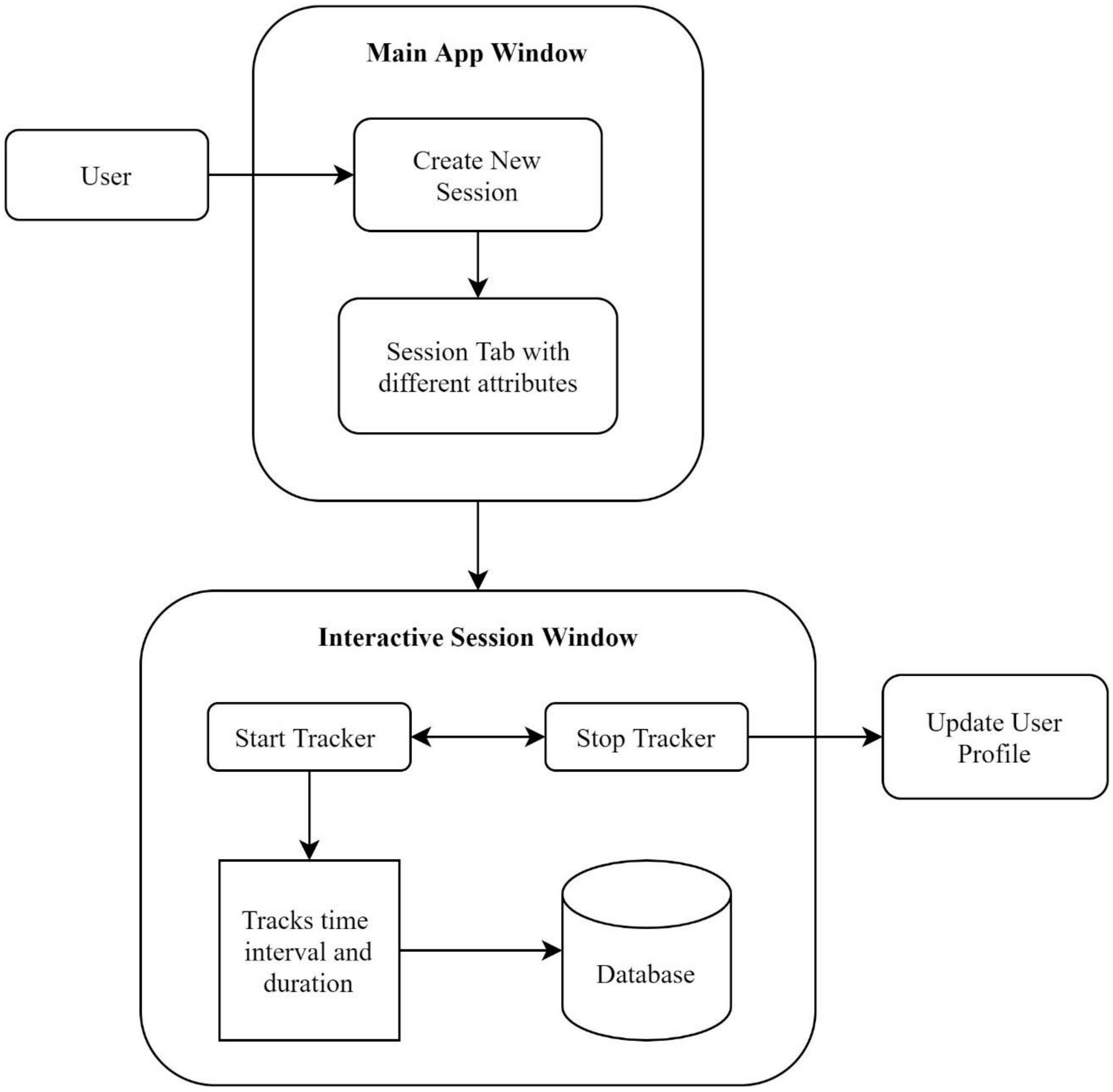

Main App Window
User
Create New Session
Session Tab with different attributes
Interactive Session Window
Start Tracker
Stop Tracker
Update User Profile
Tracks time interval and duration
Database

Figure 3.3: Session Tracker Architecture

3.1.3 To-do List

It is an ordered top-to-down listing of the task. Task added are represented in a div like element which includes editing and remove option. Completion of any task is denoted by a strike through the task. Using CRUD operations on the related model in the local database, the task is manipulated and stored.

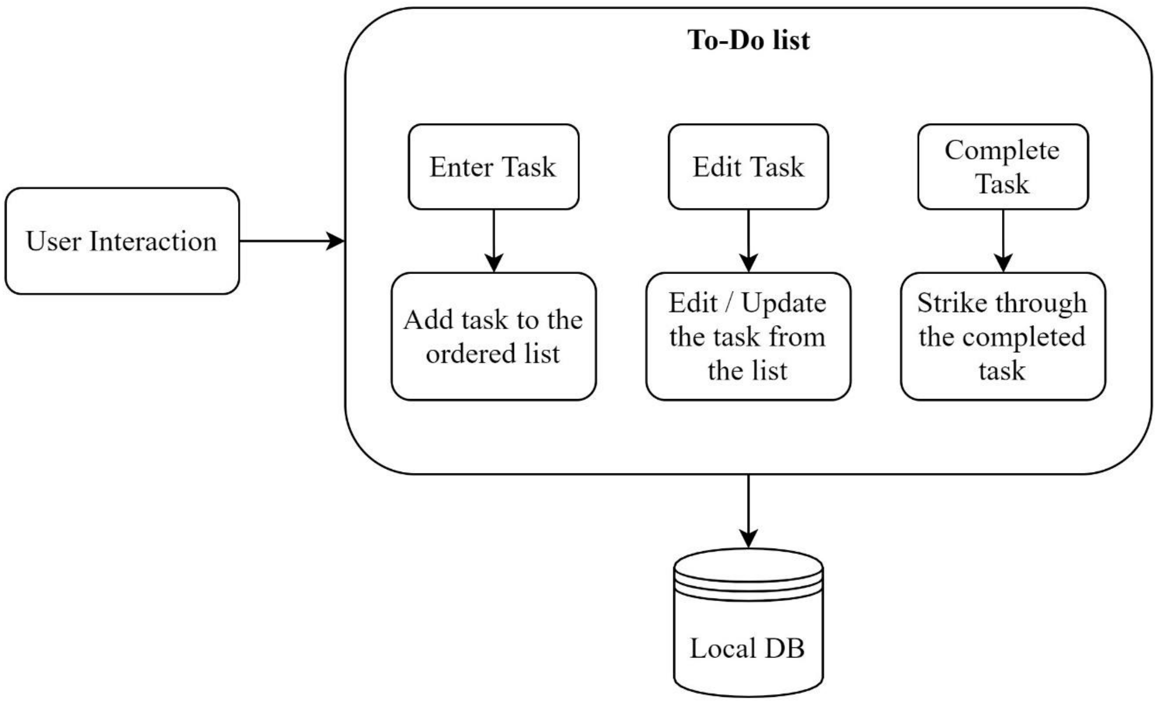

To-Do list
User Interaction
Enter Task
Edit Task
Complete Task
Add task to the ordered list
Edit / Update the task from the list
Strike through the completed task
Local DB

Figure 3.4: To-do List Architecture

3.1.4 Study Planner

The system architecture of ‘Study Planner’ is the amalgamation of both ‘Session planner’ and ‘To-do List’. It is the extended form of To-do List which features separate section for every topic as plan sheet. The plan sheet tab displays the required details related to it. Each plan sheet expands into a separate window where adding, editing / updating of the task is carried out. Completion of any task in the list is designated by a strike through. Using CRUD operations on the related model of the database, the information for all the plan sheet is handled and stored.

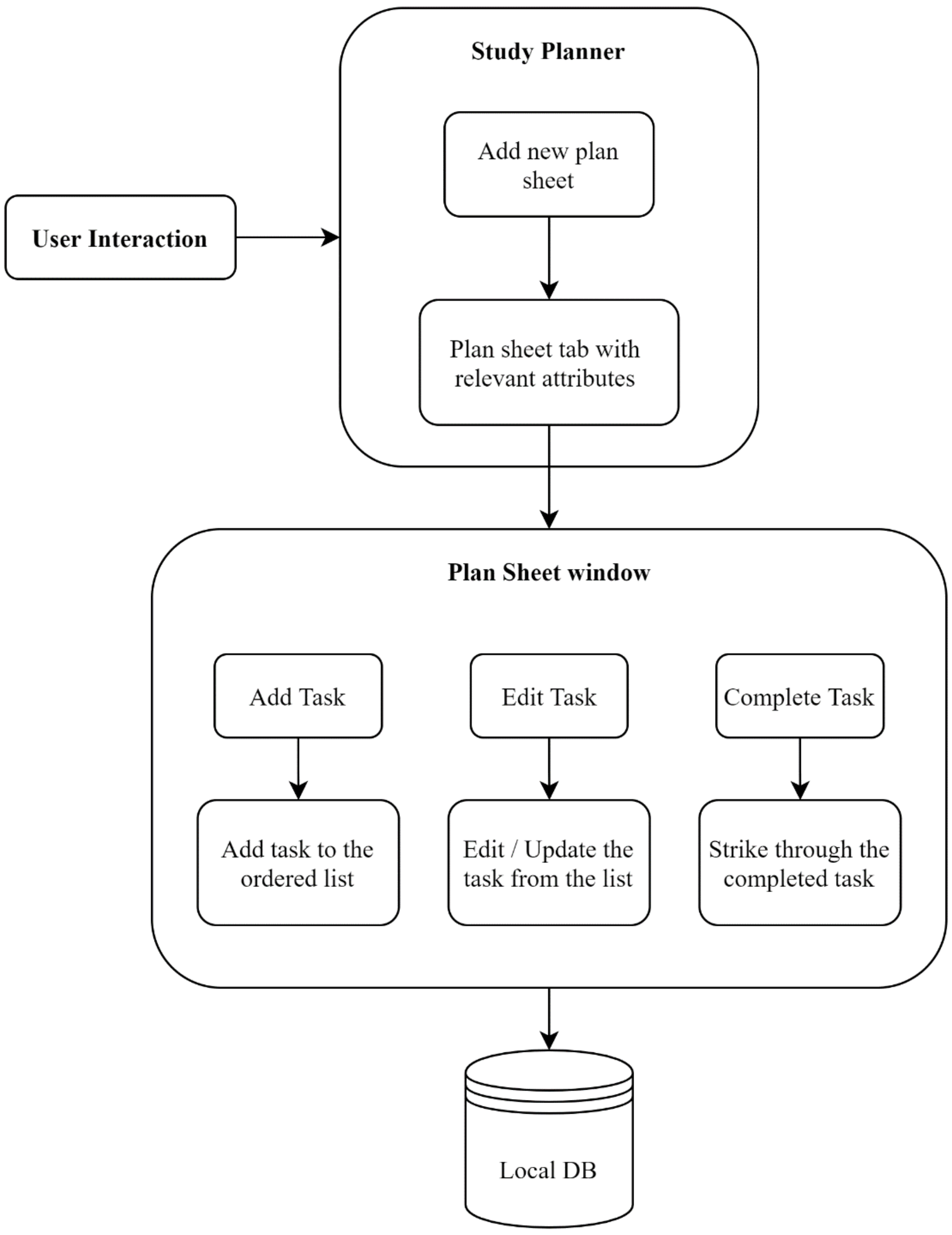

Study Planner
Add new plan sheet
Plan sheet tab with relevant attributes
User Interaction
Plan Sheet window
Add Task
Edit Task
Complete Task
Add task to the ordered list
Edit / Update the task from the list
Strike through the completed task
Local DB

Figure 3.5: Study Planner Architecture

3.2 Data Flow Diagram

The main window displays four different sub-app that the user choose to navigate. The first app being the ‘Pomo Timer’. The only data flow in the Pomo Timer is the points that the user collects for every *Pomodoro* (interval) which is stored as a reward point.

The second tab features ‘Session Tracker’. The data handling in this app will require a simple model that keeps track of the time stamps and the duration along the date. The stored session data is loaded from the stored database. Any modification to the existing session will be updated to the corresponding session data.

The third tab ‘To-do List’ grabs any previous task data from the database and then loads it in the display view. Any modification to the list will be updated to the corresponding table in the database after the tab is closed.

The final tab ‘Study Planner’ loads the plan sheet tabs if any into the initial display window. Each tab is a planner sheet that loads any previous task list in a separate interactive display window. Any modification to the planner sheet will be updated to the corresponding table after the sheet is closed.

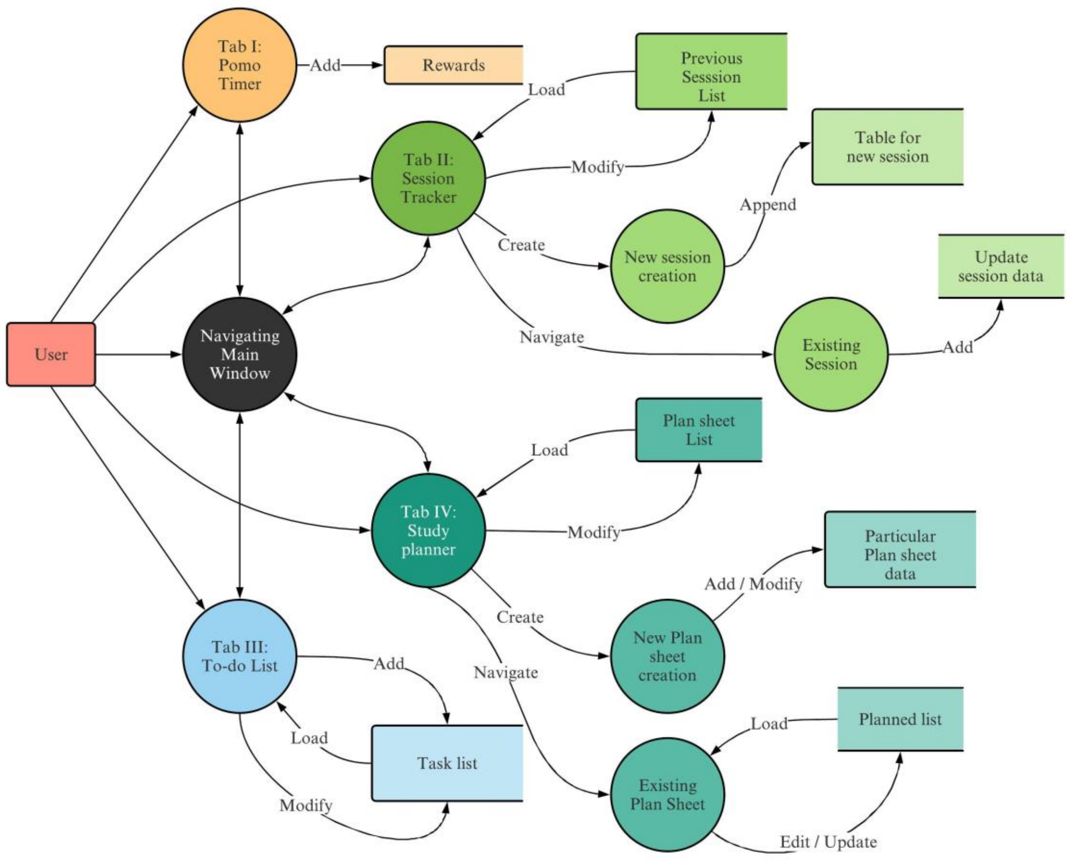


Figure 3.6: Data Flow Diagram for the Proposed System

3.3 Tools and Environment

3.3.1 IDE Used

Visual Studio 2019: Visual studio is used in our project for User Interface Development using some GUI library.

Embarcadero Dev-C++ 6.9: It is used in our project for basic theory testing.

3.3.2 Programming Language and Library Used

C++: C++ standard 17 is used in our project.

SQL: SQL is used for accessing and manipulating database.

SFML: Simple and Fast Multimedia Library (SFML) is used as GUI library for our project.

4 METHODOLOGY

4.1 Pomo Timer

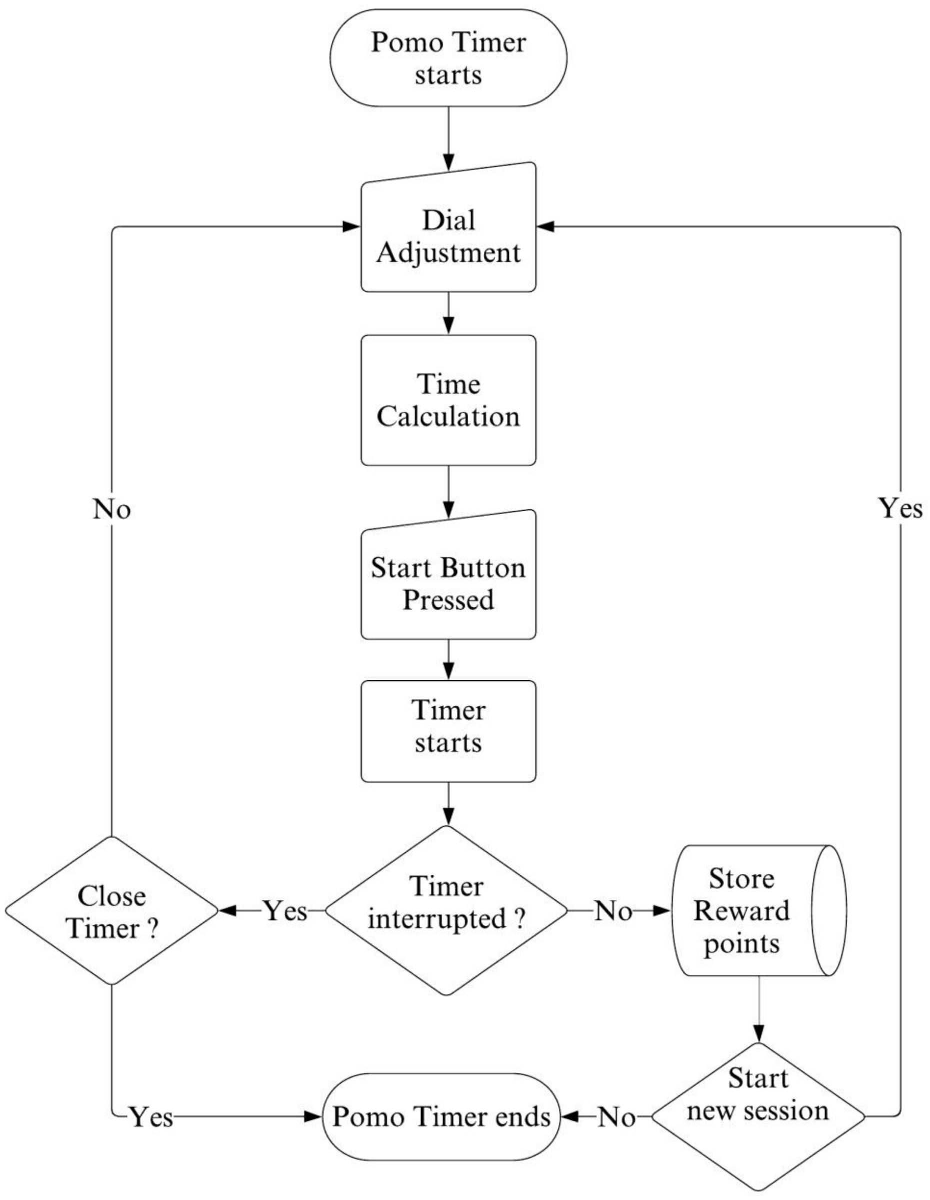


Figure 4.1: Flowchart of Pomo Timer

Pomo timer makes the use of user interactive dial like interface using which the user will be able to set the time for a pomodoro session. The timer starts when the user presses the play button and it ends when user completes the session or user chooses to end the session before the timer completes. User will be rewarded points as per the length of the session provided that they complete it.

4.1.1 Basic Interface

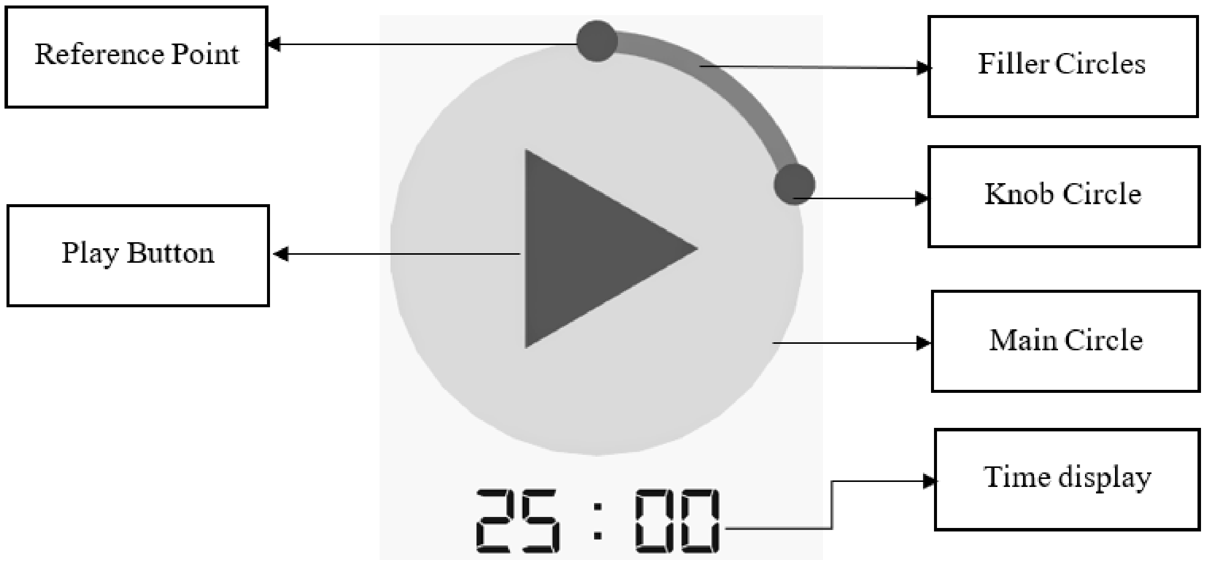


Figure 4.2: Basic Interface

For setting the timer user will be provided with an interactive dial like interface. The dial consists of 3 main components namely: a main circle, filler circles and a knob circle. By dragging the knob circle users will be able to adjust the timer as per their liking which is displayed just below the dial. Meanwhile, the filler circles visually show the point up to which the knob circle is moved from the reference point.

4.1.2 Filler Circles

sf::Vector2f is a utility template class for manipulating 2-dimensional vectors defined with two coordinates (x and y) which are its data members that can be accessed directly. *Sf::Vector3f* is similar to *sf::Vector2f* except for having 3 coordinates(x, y and z). [7]

In the beginning, all the coordinates on the outline of the main circle is calculated using general equation of circle and stored in a C++ vector having datatype as *sf::Vector2f*. Calculation involved is shown below: [8]

If equation of the main circle is:

$$(x-a)^2+(y-b)^2=r^2$$

Where (a,b) is the coordinate of centre of main circle and r is it's radius. (x,y) is any point on the circumference of the circle.

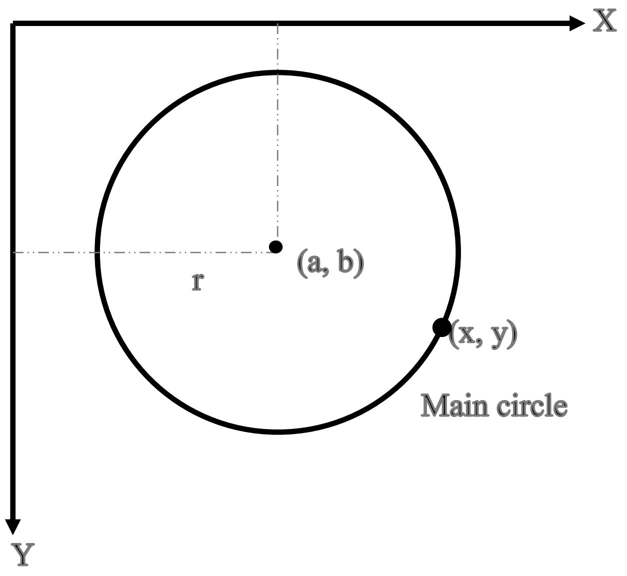


Figure 4.3: Main Circle at Position (a, b)

Upon solving above equation for x, we get

$$x = \sqrt{r^2 - (y - b)^2} + a$$

Value of coordinate 'y' is iterated form top of the main circle to its bottom and corresponding value of 'x' is calculated. (x, y) gives coordinates of points on the circumference on the right half of the main circle. Again, 'y' is iterated from bottom to top of the main circle and corresponding value of 'x' is calculated. (-x+2a, y) gives coordinates of points on the circumference on the left half of the main circle.

Then, all the positions in the outline of the main circle where the knob circle would set time of positive whole numbers i.e., 1,2,3 … 120 are calculated and stored in a C++ vector having datatype of sf::Vector3f where x is time and, y and z are the coordinates of the points. Similar process is repeated for to find out positions where knob circle would set the time value in minutes that is exactly divisible by five i.e., 0,5,10,15, …, 120 is also found out.

Later, by iterating through vector containing all positions on the outline of the main circle, filler circle is drawn in every iterated position. Iteration stops when the iterated value is equal to the coordinates of the knob circle.

4.1.3 Knob Mechanism

Initially the knob circle is placed at the position where the set time for the timer is 25 minutes. Upon dragging, the knob circle is positioned along the outline of the main circle with respect to the position of the mouse pointer. The coordinates of that position are calculated by using a simple geometric calculation that involves the intersection of the line (passing through center of the main circle and the coordinates of the mouse) and the outline of the main circle itself which is illustrated below: [8]

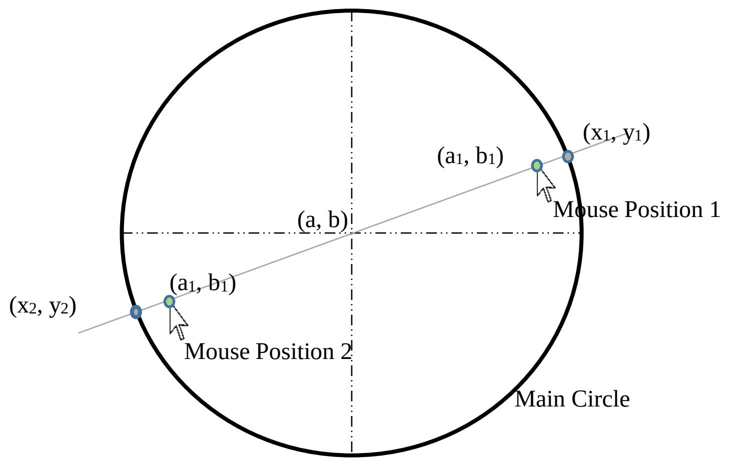


Figure 4.4: Knob Positioning Calculation

As shown in the figure above, although the mouse pointer is at different positions they generate same line passing through the origin.

Eqation of line passing through centre (a,b) and the coordinates of the mouse (a_1,b_1) is given by:

$$y = mx + c$$

Where,

$$m = \frac{b_1 - b}{a_1 - a}$$

Equation of the circle is:

$$(x-a)^2+(y-b)^2=r^2$$

The co-ordinates of the points (x_1,y_1) and (x_2,y_2) are

$$x_1=\frac{a+\ bm-\mathrm{d}m+\sqrt{\delta}}{1+m^2}$$

$$y_1=\frac{d+am+bm^2+m\sqrt{\delta}}{1+m^2}$$

$$x_2=\frac{a+\ bm-\mathrm{d}m-\sqrt{\delta}}{1+m^2}$$

$$y_2=\frac{d+am+bm^2-m\sqrt{\delta}}{1+m^2}$$

Where,

$$\delta=r^2(1+m^2)-(b-ma-d)^2$$

When $a_1 > a$, knob is positioned at (x_1,y_1) or else it is positioned at (x_2,y_2) as shown below:

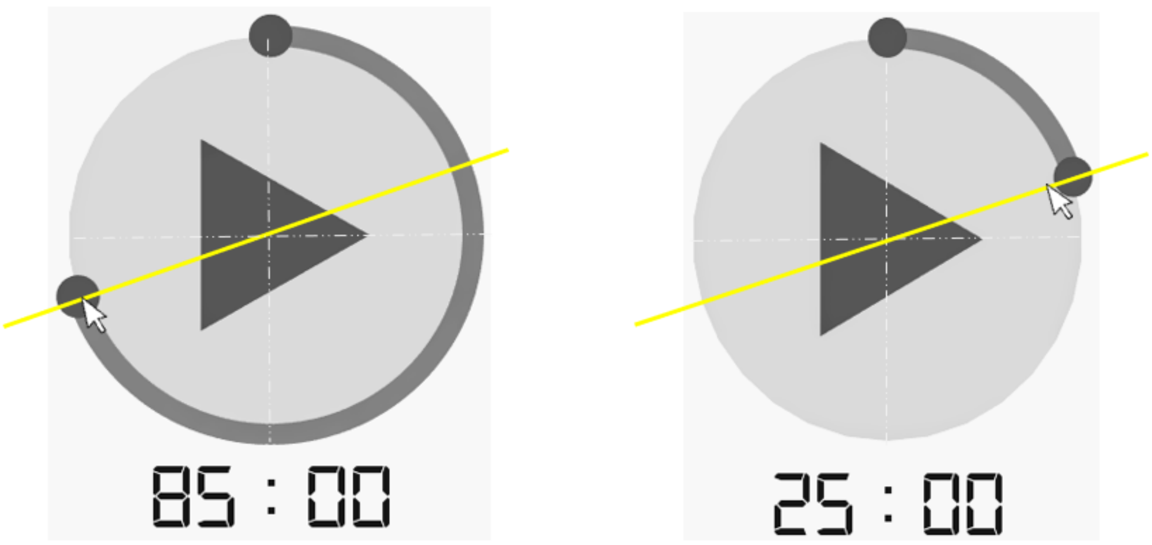


Figure 4.5: Position of the Knob Varied by Mouse Position

When user releases the mouse and stops dragging the knob, its position is set to a point where the time of the timer was last exactly divisible by 5.

4.1.4 Time Calculation

Time that is set by the user by dragging the knob, is calculated mathematically by evaluating time as a function of degrees. Position of the knob determines the time that will be set for the timer. When the knob is moved to its final limit i.e., when the knob makes full rotation around the main circle making an angle of 360 degrees, the timer will be set to 120 minutes. However, the user will only be enabled to set the time in the multiples of five beginning from 25 minutes to 120 minutes. Calculations involved are illustrated below: [8]

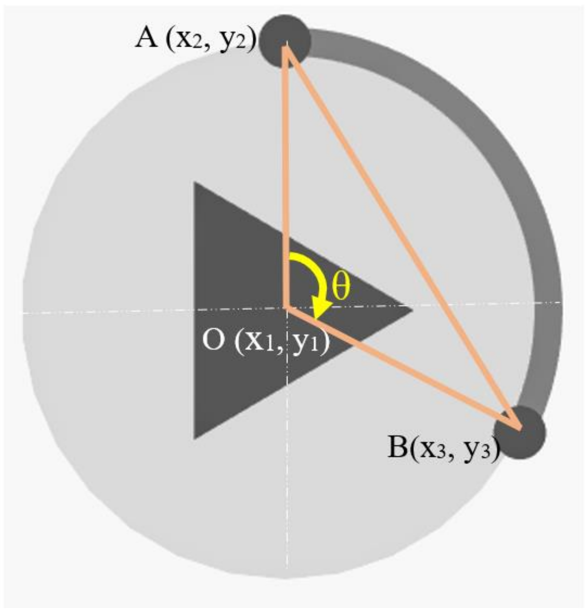


Figure 4.6: Time Calculation

$$Area\ of\ Triangle\ AOB\ (A) = 1/2 \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix}$$

$$Angle(\theta) = sin^{-1}\left(\frac{A \times 2}{r^2}\right)$$

$$Time(T) = \frac{120 \times \theta}{360}$$

4.1.5 Timer Mechanism

SFML has an inbuilt class *sf::Clock*. When an object of the class is created the clock starts. The time elapsed after the object is created can be accessed by using *getElapsedTime()* method. To get the elapsed time in seconds we use *getElapsedTime().asSeconds()* method. Moreover, the clock can be restarted using *restart()* method. [7]

When the user starts timer by clicking the play button, the clock is restarted and the timer starts ticking. Meanwhile play button is replaced by stop button using which user can stop the timer. Initially the timer begins at the time set by the user and eventually it decreases to zero (i.e., 00:00) provided that user doesn't end the session half way by pressing the stop button. Time that the timer displays at any instant is calculated by subtracting elapsed time from the time set by the user in seconds. Meanwhile, the filler circles give the visual representation of the time remaining.

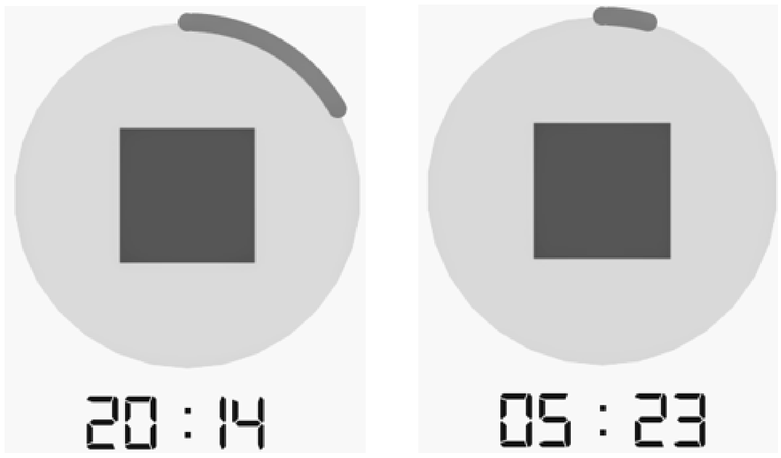


Figure 4.7: Ticking Timer

4.1.6 Lo-fi Music

User will be opted to play soothing music during a pomodoro session. SFML provides *sf::Music* class using which we can load a music file into an object. Music can be played using *play()* method, paused using *pause()* method and stopped using *stop()* method. [7]

4.2 Session Tracker

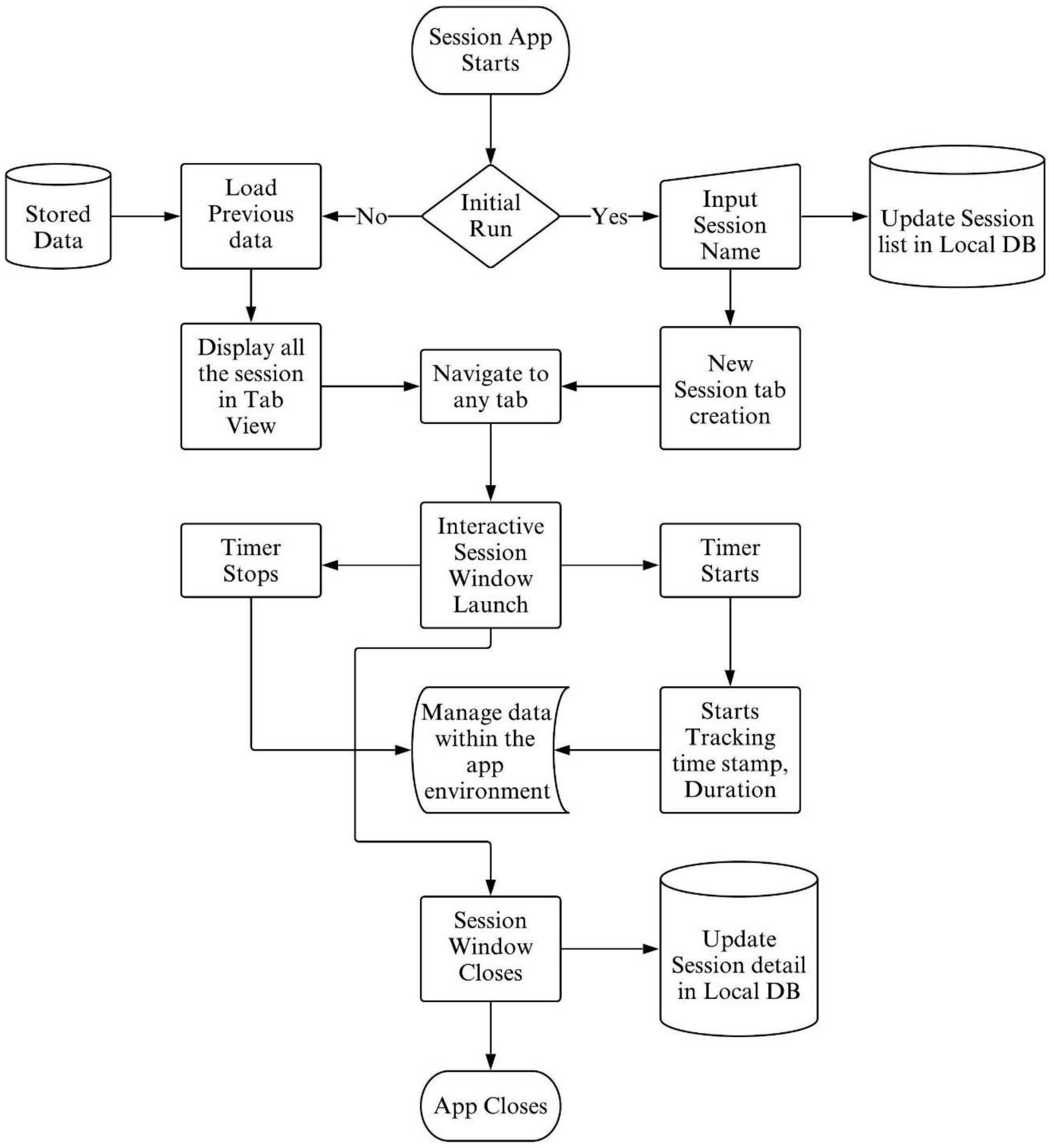

Session App Starts
Stored Data
Load Previous data
No
Initial Run
Yes
Input Session Name
Update Session list in Local DB
Display all the session in Tab View
Navigate to any tab
New Session tab creation
Timer Stops
Interactive Session Window Launch
Timer Starts
Manage data within the app environment
Starts Tracking time stamp, Duration
Session Window Closes
Update Session detail in Local DB
App Closes

Figure 4.8: Flowchart of Session Tracker

The working of the whole ‘Session Tracker’ app is going to be based upon working with local database. Basic operations such as creating, reading, updating and deleting (CRUD) on the associated database is executed for the flow of any data and information in this app. Two scenarios exist for the app based upon whether the app is being initially run or not. If the app contains no prior stored data, in the instant the app is launched, two tables will be created in the associated database. On adding new session to the session tab, the session properties will be stored in the local database. The session listed is going to appear in their own separate session tab. On clicking the particular session tab, a new interactive session window will render in the main window. This window will contain a button that toggles between starting and stopping the session. On starting the session, the stop watch appears and will start counting every millisecond. On stopping the session, the time stamp and the duration of that particular session will be stored within the program. On exiting the app, the data temporarily stored in the container will be transferred to the associated local database.

4.2.1 Main Window Interface

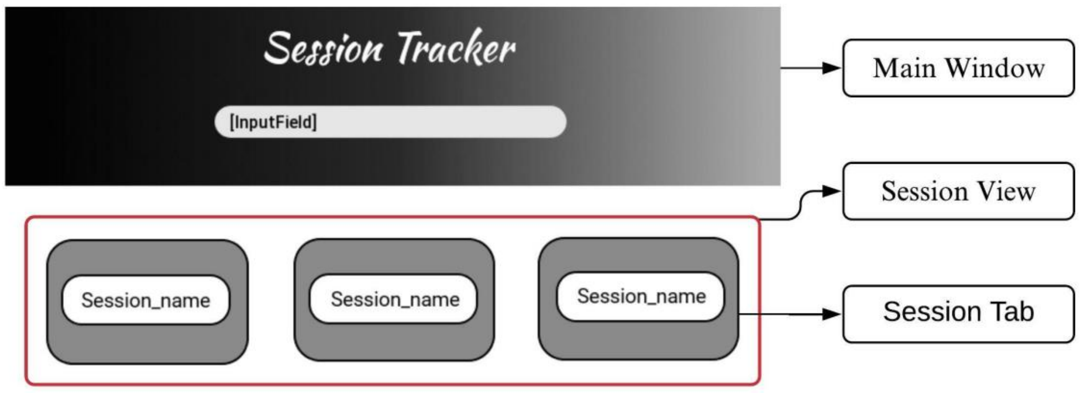


Figure 4.9: Main Window Layout

4.2.1.1 Main Window

SFML provides two different classes for rendering the window. The main window will be using *sf::RenderWindow*. *sf::RenderWindow* provides useful methods such as polling any type of event within the window, extracting different properties of the window, clearing and updating the window, etc. The main window design contains the app title, an input field and a button that toggles and the session view. On clicking the 'Add new session' button, the input field appears. On adding the session, a new session tab in the session view will appear. All the required UI components will be built separately inheriting from SFML classes. [7]

4.2.1.2 Session View

Viewing section will contain all the session tabs. Every session tab will be equally spaced within the view. On exceeding the size of the view, the scroll effect will appear for the session view. For the implementation of scroll effect, *sf::view* with view port size as the session view will be set. There will be a maximum limit of the number of tabs that can be contained within the view. [7]

4.2.1.3 Session Tab

Tab is going to be implemented using the round cornered rectangle and a button within it. Each tab will have an event handler that detects the clicking of the tab button. The tab will then expand to the window size as new session window. The tab might contain associated information of the tab such as creation date and the stop watch info if any.

4.2.2 Session Window Interface

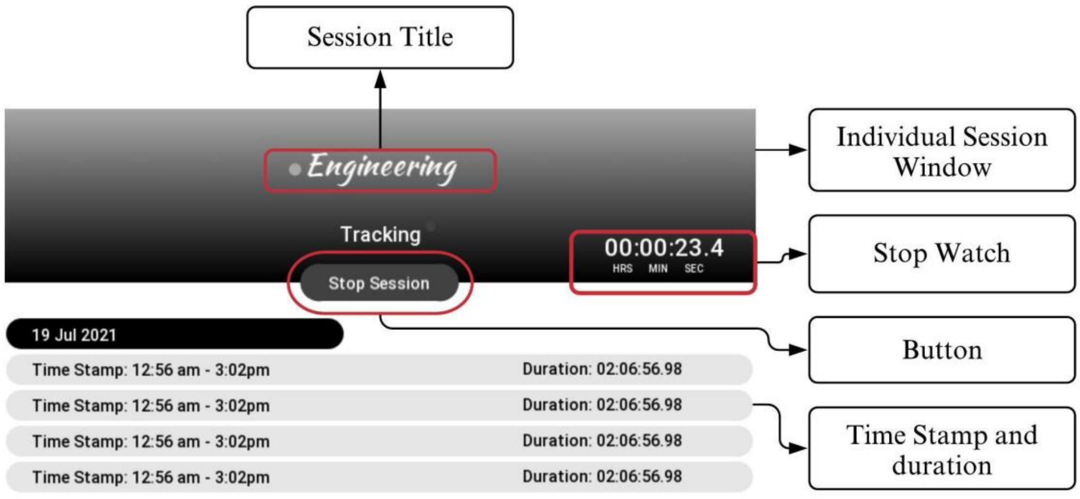


Figure 4.10: Session Window Layout

4.2.2.1 Individual Session Window

New session window will be rendered in the same main window. This window will contain the main aspect of the app. Its design contains the session name, stop watch, a button that toggles between start and stop session and the session details. This window will have the same dimension as the main window. The graphical design of the session window will contain the gradient background which contains the session name, stop watch and the button. The second part of the design will contain the session details in a tabular format.

4.2.2.2 Stop Watch

Stop watch records every millisecond of time using *sf::Clock* and its methods: *getElaspsedTime(), asSeconds()* and *asMilliSeconds()*. A conversion function will then convert the seconds into hour and minute. The stop watch will display the recorded time in [hour: minute: second. millisecond] format. This stop watch will be connected to the button. On clicking the start session button, the stop watch appears and starts counting every second of time and updates the watch in the window. [7]

4.2.2.3 Toggle Button

The button will act as toggle button between ‘Start Session’ and ‘Stop Session’. On clicking the start session, the button will be replaced by the stop session and the stop watch starts. On clicking the stop session, the stop watch will stop and disappear and the detail of that particular session will be added to the session detail view.

4.2.2.4 Session Detail

The header file *time.h* provides suitable methods to get the system date and time. Using *time.h*, the date and time for a particular session will be tracked. This data will temporarily be stored within the program in STL containers like vector. All these data will be then rendered in list format and the information of the tracked record will be displayed within a designed component. The session details will be displayed via *sf::view* for the scrolling effect. *sf::View* is a class entity defined in SFML that defines which part of the window is to be shown using target view port size. On exiting the app, the data will be then transferred to the local database. [7]

4.2.3 UI Components

SFML offers only basic elements. For any type of UI responsive and interactive component such as button and input field, different components provided by SFML and their functionality must be incorporated together.

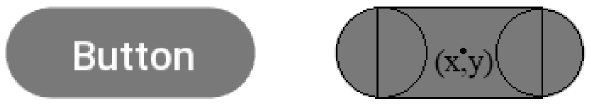


Figure 4.11: Button Geometry

As shown above, making a button requires one rectangle and two circles. The rectangle is an object of *sf::RectangleShape* class whereas the circle are objects of *sf::CircleShape* class. Horizontally and vertically centered text in the button is a text object of *sf::Text* class. The triggering event for the button will be done through a wrapper function. *std::function* is a general-purpose polymorphic function wrapper. Any associated function in the form of lambda expression will be bind to the button using *std::function*. Any real-type mouse event is polled using *sf::Mouse* class. Using this event the wrapped function is called and the desired action is achieved. [7]

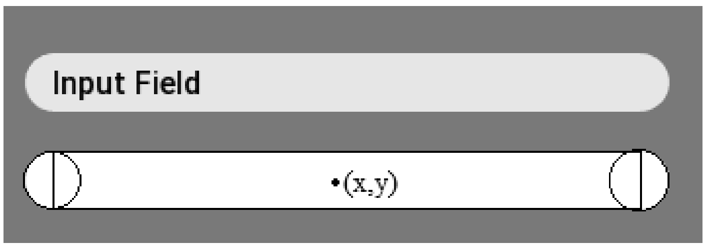


Figure 4.12: Input Text Field Geometry

The basic design and the shape of the input text field is same as that of the button. It provides additional feature of typing the text within the field. Here, real-time keyboard input is detected using *sf::Keyboard* class. These common UI components will be separately coded in a different user defined header file inheriting different classes from SFML. [7]

4.3 To-Do List

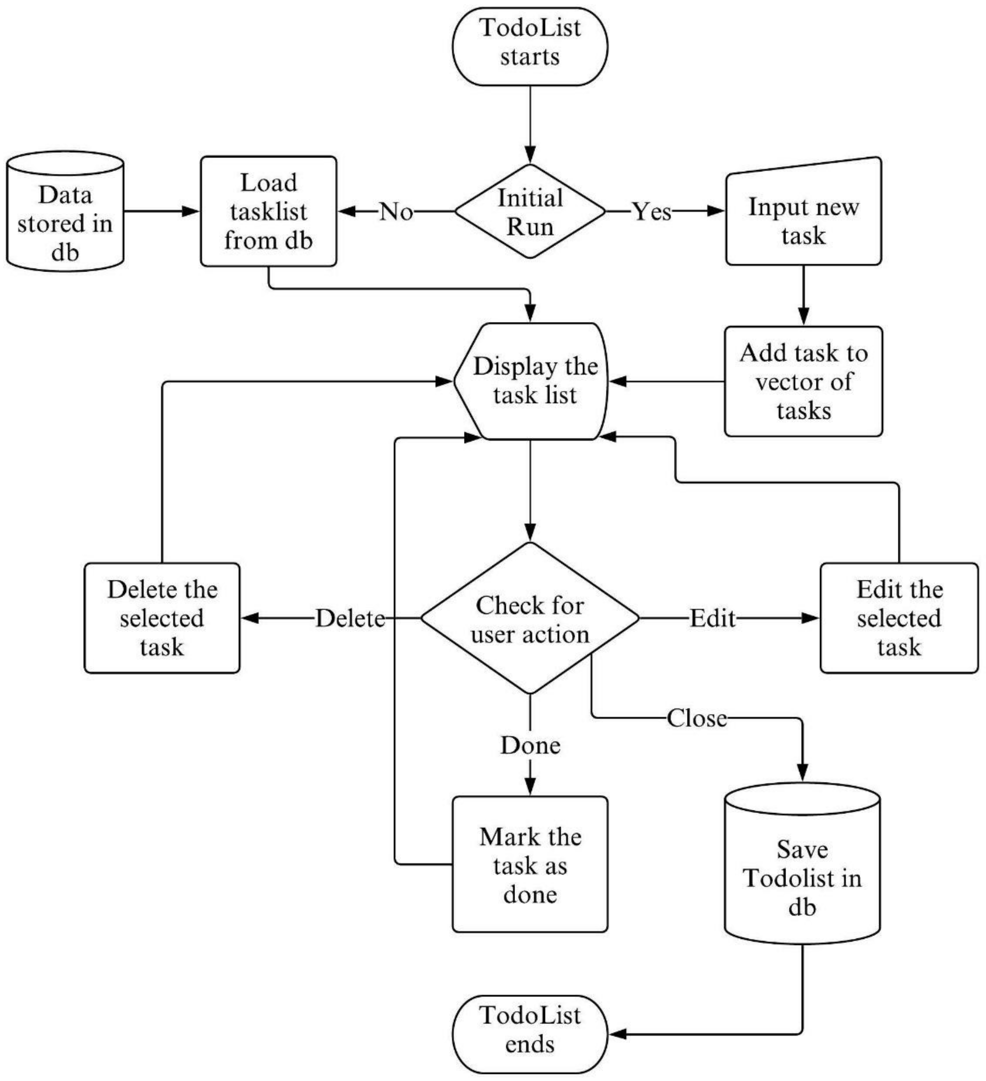


Figure 4.13: Flowchart of To-do List

The proposed flowchart of the program is shown above which shows how To-do List works. This will be implemented with the help of user defined functions and classes. A separate class to take input from the user and store the task is used. And the tasks are stored in a vector of the same class. The task list is printed by iterating the vector. There are buttons to mark done, edit, delete the task for each task and the user action is checked through a function which directs the operations as per the user action and respective action is carried out. Previously added task are loaded from the database when the program is started and the added tasks are saved to the database when the program is closed.

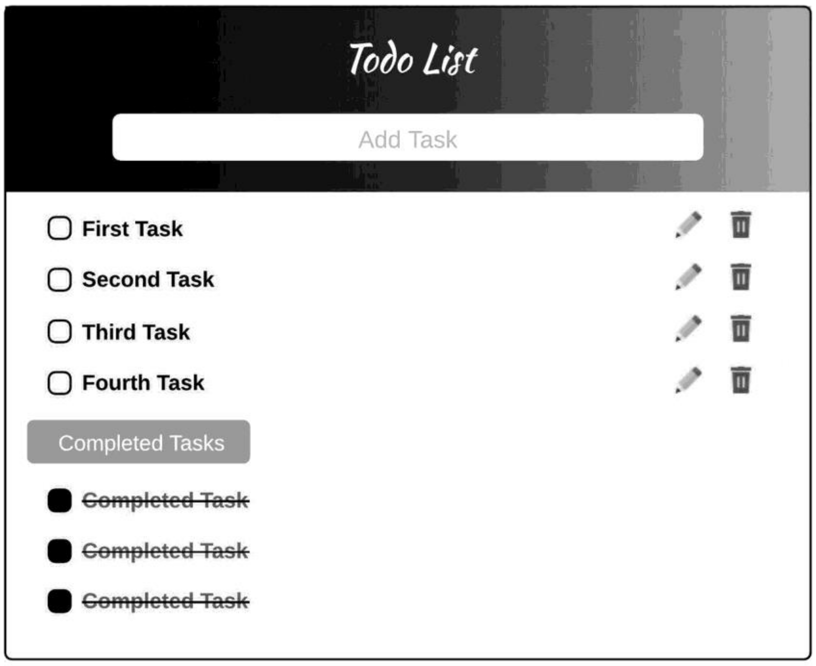


Figure 4.14: Proposed Design for To-do List

4.3.1 Loading To-do List from Database

When the app is launched, first it will check whether there are tasks added previously or not in the database and if any tasks exist, they're loaded into the program with the help of loader function into a vector container and then displayed in the window along with the input text field to add new task. If there were no previous task, we simply display the window without any task and input field to add the task. The loader function interacts with the SQLite3 API to read the data from the Database.

4.3.2 Adding New Task

To add the task in our To-do List we have an Inputfield at the top of our task list where user can enter the name of the task and press enter to add the task to the list. For this we'll use the *sf::Text* object available in SFML to store the task and give output in window and *sf::Event* object to check if the user is entering the text. The input area itself is an object of *Button Class* (user defined class) which needs to be created utilizing the previously available classes like *sf::Text*, *sf::Shape* (rectangle, circle), *sf::Event* etc. to get the desired result. [7]

4.3.3 Editing Tasks

Every task in the To-do List has its own edit button to provide user an option to edit the task if they want to modify the task they previously added. The edit button is also an object of *Button Class.* When edit button is pressed it'll load the task into the input area at the top where user can edit the task and press enter to save the changes.

4.3.4 Deleting Tasks

If the users need not to perform the task they added previously, every task has its own delete button to enable user to delete the unwanted task. Delete button is similar to edit button but instead of editing it deletes the task.

4.3.5 Marking Completed Tasks

There is a small check button to mark the completed task for every task in the list. It is also similar to edit and delete button but marks the task as done and draws strike through line on the task.

4.3.6 Showing Due Tasks

If the task are not completed within the day they're marked as due and displayed under the Previous Dues List in the To-do List above the tasks that are not overdue.

4.3.7 Displaying Tasks

When the task is loaded from the database they're loaded into a vector container of tasks which works as the list of all the added tasks. The tasks are categorized under two groups i.e. overdue or not overdue. The overdue tasks are displayed under the heading 'Previous Dues' and rest under the heading 'My To-dos'.

4.4 Study Planner

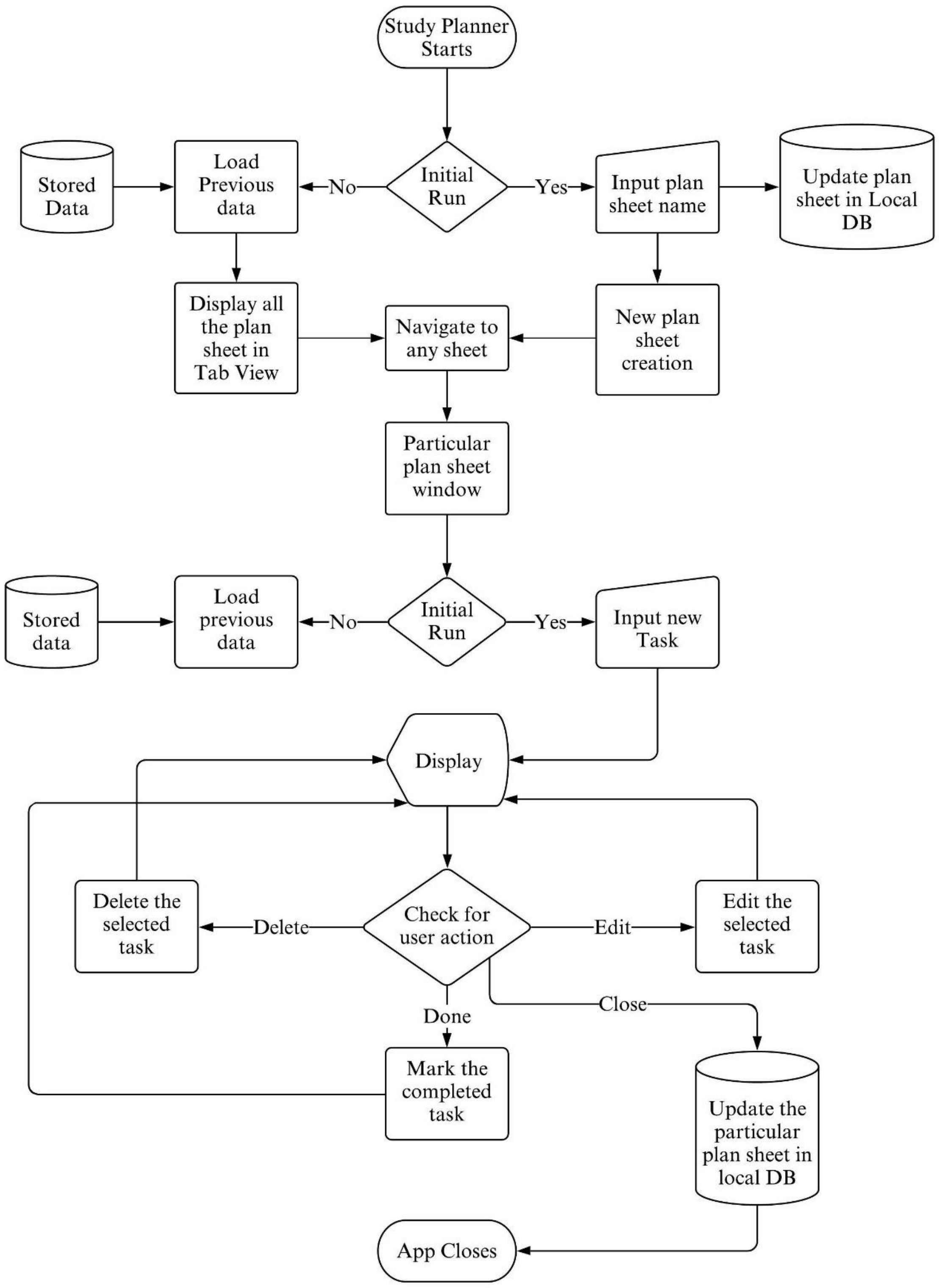


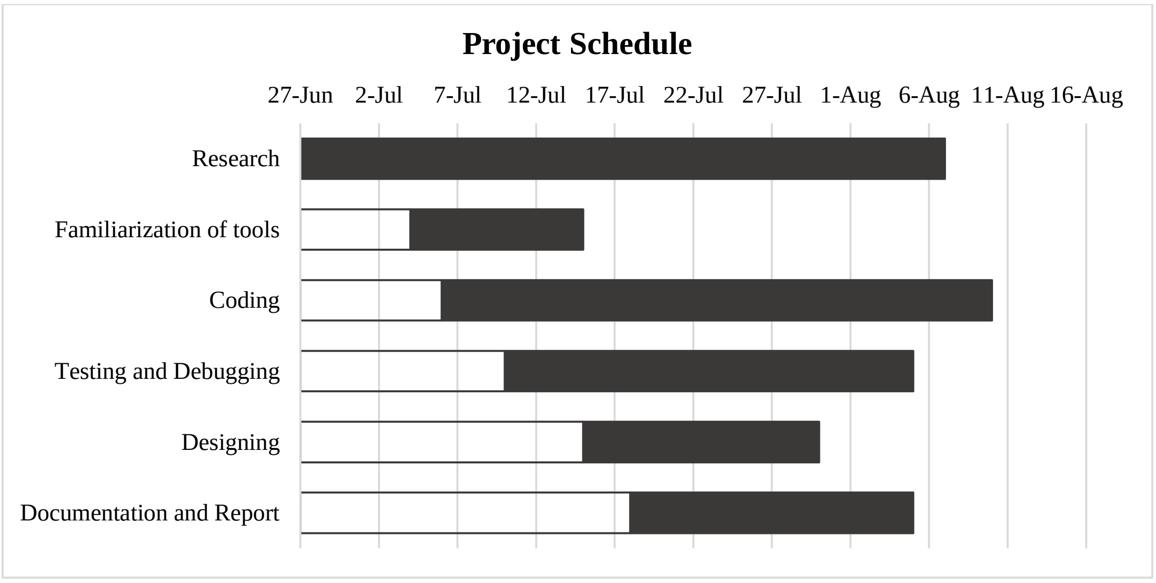
Figure 4.15: Flowchart of Study Planner

The layout and the operation involved will be amalgamation of To-do List and Session Tracker. For the initial launch of the app, two tables will be created in the database. The main window will contain an input field where the name for the plan sheet will be entered. This created plan sheet will then appear in the viewing area. Navigating to any of the plan sheet will expand the plan sheet window in the same working window.

Plan sheet window will contain an input field and the listing area. Every task added to the sheet will be stored and managed in a vector container. All the task from the vector will be then displayed in the listing area with the option to edit and delete the task. The completed task will be denoted with a strike through. The local database will be updated at the event of closing the plan sheet. The main window interface of this app is similar to that of Session Tracker. The plan sheet window layout will be similar to the layout of To-do List.

5 TIME ESTIMATION

Table 5.1: Time Estimation Gantt chart



6 FEASIBILITY ANALYSIS

6.1 Economic Feasibility

This project does not cost much as it is college project done by the students to get more familiar and efficient in their coding with C++. All the tools, libraries and services (*Git and GitHub, SFML, SQLite3 API, Lucid App, Flowchart Maker, MS teams etc.)* used for the development are freely available.

6.2 Technical Feasibility

This application is developed with the help of standard C++ libraries and additional libraries like SFML, API of SQLite3 for database management and Git and GitHub for collaboration and version control.

6.3 Operational Feasibility

Application requires 64-bit operating system with a compatible compiler for C++ (GNU g++).

6.3.1 Software Requirements

For the program to run computer must meet the following software specifications:

- Operating System: - WINDOWS 98 or newer.
- Application Software: - Visual Studio or Dev C++ with g++ or any C++ compiler.

6.3.2 Hardware Requirements

For the program to run computer must meet the following hardware specifications:

- Intel core2 Duo 1333 MHz or higher.
- 4MB caches memory or higher.
- 128 GB HDD and 1GB RAM or higher.
- Any Graphics card supporting OpenGL 1.1.

REFERENCES

[1] A. Dolle, "The only article you need to read about Productivity," Getmailbird, 16 06 2015. [Online]. Available: https://www.getmailbird.com/the-only-article-you-need-to-read-about-productive/.

[2] S. Pogue, "How To Be More Productive: 18 Top Tips To Help You (Don’t Miss The Last One!)," Workzone, 16 12 2020. [Online]. Available: https://www.workzone.com/blog/how-to-be-more-productive/.

[3] F. Cirillo, The pomodoro technique (the pomodoro), Berlin: FC Garage, 2006.

[4] "Forest Stay focused, be present," Forest , [Online]. Available: https://www.forestapp.cc/.

[5] "Microsoft To Do," Microsoft, [Online]. Available: https://todo.microsoft.com/tasks/.

[6] "Tide homepage," Tide, 2017. [Online]. Available: https://tide.fm/en_US/.

[7] L. Gomila, "Simple and Fast Multimedia Library," SFML, [Online]. Available: https://www.sfml-dev.org/.

[8] L. P. Eisenhart, Coordinate Geometry, New York: Dover Publications, Inc, 1960.