

**A REPORT
ON
Automation of Test Application using PyAutoGUI**

Submitted by,

**Ms. Shilpa Nagaraj – 20211CAI0161
Mr. Nikith Murali – 20211CAI0093**

Under the guidance of,

Mr. Santhosh Kumar K L

in partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

**COMPUTER SCIENCE AND ENGINEERING
(Artificial Intelligence and Machine Learning)**

At



PRESIDENCY UNIVERSITY

BENGALURU

MAY 2025

PRESIDENCY UNIVERSITY

PRESIDENCY SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

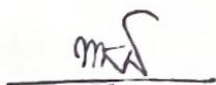
This is to certify that the Internship report “AUTOMATION OF TEST APPLICATION using PyAutoGUI” being submitted by “SHILPA NAGARAJ, NIKITH MURALI” bearing roll number “20211CAI0161, 20211CAI0093” in partial fulfillment of the requirement for the award of the degree of Bachelor of Technology in Computer Science and Engineering is a bonafide work carried out under my supervision.



Mr. SANTHOSH KUMAR K L
Assistant Professor
Presidency school of CSE(PSCS)
Presidency University



Dr. ZAFAR ALI KHAN N
Professor & HoD
Presidency school of CSE(PSCS)
Presidency University



Dr. MYDHILI K NAIR
Professor & Associate Dean
Presidency school of CSE(PSCS)
Presidency University



Dr. MD. SAMEERUDDIN KHAN
Pro-Vice Chancellor - Engineering
Dean –PSCS & PSIS
Presidency University

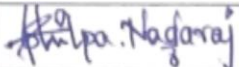
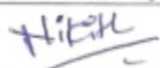
PRESIDENCY UNIVERSITY

PRESIDENCY SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

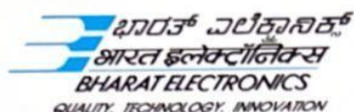
DECLARATION

We hereby declare that the work, which is being presented in the report entitled “Automation of Test Application using PyAutoGUI” in partial fulfillment for the award of Degree of **Bachelor of Technology in Computer Science and Engineering**, is a record of our own investigations carried under the guidance of Mr. **Santhosh Kumar K L**, Assistant Professor, School of Computer Science Engineering & Information Science, Presidency University, Bengaluru.

We have not submitted the matter presented in this report anywhere for the award of any other Degree.

Name(s)	Roll No.	Signature(s)
Shilpa Nagaraj	20211CAI0161	
Nikith Murali	20211CAI0093	

INTERNSHIP COMPLETION CERTIFICATE



Skill India
कौशल भारत - कुशल भारत

BHARAT ELECTRONICS LIMITED

(A Govt. of India Enterprise, Ministry of Defence)
Jalahalli Post, Bengaluru - 560 013, India

CENTRE FOR LEARNING AND DEVELOPMENT

Certificate

This is to certify that

Sri./Smt/Kum **NIKITH MURALI**

Ref No. **2025-26/973**

student of **PRESIDENCY UNIVERSITY,**

..... **BENGALURU**

carried out Project Work/Internship on

..... **APPLICATION TEST**

..... **AUTOMATION**

in **SOFTWARE**

SBU/CSG of BEL, Bengaluru from **03-03-2025**

to **05-05-2025**

*He/She was regular and punctual in his/her attendance
and his/her conduct was satisfactory during the period.*

Project / Internship Guide **K. C. AJJYK. C.**

Date : **5/5/25**
Place : Bengaluru
प्रबंधक / MANAGER
सॉफ्टवेयर / SOFTWARE

Head (HR/CLD)

सुजाता फ्रांसिस / SUJATHA FRANCIS
प्रबंधक (मानव संसाधन / सीएलडी)
MANAGER (HR/CLD)
भारत इलेक्ट्रॉनिक्स लिमिटेड
BHARAT ELECTRONICS LTD.
जे.ए.पी.पोस्ट, बंगलूरु-560 013
JALAHALLI POST, BANGALORE-560 013

BHARAT ELECTRONICS LIMITED

(A Govt. of India Enterprise, Ministry of Defence)
Jalahalli Post, Bengaluru - 560 013, India

CENTRE FOR LEARNING AND DEVELOPMENT

Certificate

This is to certify that

Sri./Smt./Kum. SHILPA NAGARAJ

Ref No. 2025-26/ 974

student of PRESIDENCY UNIVERSITY,

..... BENGALURU

carried out Project Work/Internship on

..... APPLICATION TEST

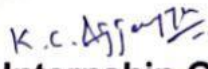
..... AUTOMATION

in SOFTWARE

SBU/CSG of BEL, Bengaluru from . 03-02-2025

to 05-05-2025

*He/She was regular and punctual in his/her attendance
and his/her conduct was satisfactory during the period.*

Project / Internship Guide /  K.C. Aggarwal
अजय कुमार
BCE 214900

Date : 5/5/25

Place : Bengaluru

प्रबंधक / MANAGER

सॉफ्टवेयर / SOFTWARE


Head (HR/CLD)

सुजाता फ्रांसिस / SUJATHA FRANCIS

प्रबंधक (म. स. / सी. एल. डी.)

MANAGER (HR/CLD)

भारत इलेक्ट्रॉनिक्स लिमिटेड

BHARAT ELECTRONICS LTD.

जालहल्ली पोस्ट, बेंगलूरु-560 013

JALAHALLI POST, BANGALORE-560 013

ABSTRACT

Repetitive digital tasks often consume significant time and are prone to human error, especially in process-driven environments. To address this challenge, **MacroMind** was developed—a Python-based desktop automation tool that intelligently records and replays user interactions with precision. By automating routine mouse and keyboard operations, *MacroMind* reduces manual effort by an estimated 30% and enhances accuracy across repetitive workflows.

The system operates in two core phases: **recording** and **playback**. During recording, it captures user input events—including mouse movements, clicks, and keyboard strokes—alongside high-resolution timestamps, preserving the exact sequence and delay of actions. These interactions are logged in a structured format using **pynput** for event listening and stored locally for future use. The playback phase utilizes **PyAutoGUI** to simulate the recorded actions on screen, faithfully replicating user behavior in real time.

A clean and user-friendly **Tkinter-based GUI** allows users to control the automation process effortlessly through icon-driven buttons for “Record” and “Play.” Background threading ensures the application remains responsive during recording and playback operations, delivering a smooth user experience.

MacroMind is particularly useful in scenarios such as UI testing, repetitive data entry, web-based task automation, and workflow simulation. Its architecture is modular and scalable, offering future potential for enhancements like image-based detection using **OpenCV**, cross-application automation, and cloud integration.

By bridging the gap between manual operations and intelligent automation, *MacroMind* lays the foundation for smarter, more efficient human-computer interaction in both personal and enterprise settings.

ACKNOWLEDGEMENTS

First of all, we indebted to the **GOD ALMIGHTY** for giving me an opportunity to excel in our efforts to complete this project on time.

We express our sincere thanks to our respected dean **Dr. Md. Sameeruddin Khan**, Pro-VC - Engineering and Dean, Presidency School of Computer Science and Engineering & Presidency School of Information Science, Presidency University for getting us permission to undergo the project.

We express our heartfelt gratitude to our beloved Associate Dean **Dr. Mydhili K Nair**, Presidency School of Computer Science and Engineering, Presidency University, and Dr. “ZAFAR ALI KHAN N”, Head of the Department, Presidency School of Computer Science and Engineering, Presidency University, for rendering timely help in completing this project successfully.

We are greatly indebted to our guide **Mr. Santhosh Kumar K L** and Reviewer Mr. Likith S R, Presidency School of Computer Science and Engineering, Presidency University for his inspirational guidance, and valuable suggestions and for providing us a chance to express our technical capabilities in every respect for the completion of the internship work.

We would like to convey our gratitude and heartfelt thanks to the PIP4001 Internship/University Project Coordinator **Mr. Md Ziaur Rahman and Dr. Sampath A K**, department Project Coordinators and GitHub coordinator **Mr. Muthuraj**.

We thank our family and friends for the strong support and inspiration they have provided us in bringing out this project.

Shilpa Nagaraj(20211CAI0161)

Nikith Murali(20211CAI0093)

LIST OF TABLES

Sl. No.	Table Name	Table Caption	Page No.
1	Table 7.2	Gantt Chart	13

LIST OF FIGURES

Sl. No.	Figure Name	Caption	Page No.
1	Figure 6.2	MacroMind System Architecture	11
2	Figure 6.3	Record and Playback button	11
3	Figure 7.2	Timeline for Execution of Project	14
4	Figure 9.2	Manual vs Automated Task Execution	19

TABLE OF CONTENTS

Chapter No.	Title	Page No.
Abstract		v
Acknowledgements		vi
List of Tables		vii
List of Figures		viii
Table of Contents		ix
1	Introduction	1
2	Literature Survey	2
3	Research Gaps of Existing Methods	3
4	Proposed Methodology	5
5	Objectives	7
6	System Design & Implementation	9
7	Timeline for Execution of Project	12
8	Outcomes	14
9	Results and Discussions	17
10	Conclusion	19
References		21
Appendix-A	Pseudocode	22
Appendix-B	Screenshots	29
Appendix-C	Enclosures	30

Chapter 1

INTRODUCTION

1.1 Background and Relevance

In today's rapidly evolving digital environment, as digital capabilities continue to grow at breakneck speed, efficiency in software testing and automation is paramount. Manual repetitive tasks, especially in a GUI-based application, are time-consuming and often introduce human errors. Organizations have turned to automating / automated solutions to minimize this effort and improve overall efficiency. One of the greatest benefits is in the test scenarios where speed, consistency, and accuracy are very important.

1.1.1 Importance of GUI Automation in the Industry

Graphical User Interface (GUI) automation is essential in software development and quality assurance. Most testing tools we've previously reviewed concentrate on web apps or command-line interfaces, but there are few good open-source GUI automation options for desktop applications like QATracker. Most people are often very familiar with desktop applications. We use them daily, from word processors to email programs to games. The ability to perform automated user behaviors on desktop applications—mouse clicks, keyboard input, menu navigation, etc.—can save massive amounts of time and lead to greater testing throughput. GUIs are not specific to web apps or typical user-facing applications. Typically, companies like the defense industry, enterprise software, or any public sector services, use GUI machines to make sure that workflow processes are automated, rather than relying on human input.

1.2 Motivation for the Project

The impetus for this project resulted from the desire to remove manual effort in repetitive tasks. This is particularly true in the automated testing space. Typical automation software requires technical knowledge and is usually limited to web-based applications. There are many areas where a solution that provided non-technical users with the ability to record and replay desktop activity with high fidelity would be of great value.

1.3 Scope of the Project

This project is focused on developing a desktop-based, open-source automation system that would allow the user to record their own actions, and then play them accurately while handling delays. It's light weight, platform independent and built in open source python libraries to provide an affordable way to automate graphical user interfaces (GUIs) with these characteristics, e.g. it will provide user friendly GUI automation, it will be accurate, and it will use as little resources as possible (time, money, energy, etc). This contributes to faster test cycles and productivity improvements.

Chapter 2

LITERATURE SURVEY

2.1 Overview of Automation Tools

Automation tools have evolved significantly over the past decade to reduce manual efforts in software development and testing. Different tools serve different environments; some tools are for web automation; others are for desktop applications. Each of the tools has advantages as well as shortcomings based on the platform dependencies, scripting requirements, usability, and availability of community support.

The demand for automation in testing GUI has grown with the increased proliferation of user-focused desktop applications across industries. More traditional testing tools are unable to mimic human-type interactions on a desktop UI.

2.2 Review of Existing Tools

- **Selenium:**

Selenium is widely used for web-based automation and testing. While it supports major browsers and scripting in various languages, it lacks native support for desktop GUI automation. It is also code-heavy, which can be a barrier for non-developers. [7]

- **AutoIt:**

AutoIt is a scripting language designed for automating the Windows GUI. It is lightweight and effective but limited to Windows OS only, making it unsuitable for cross-platform solutions [8].

- **Robot-Framework:**

A generic automation framework that uses a keyword-driven approach. It supports multiple libraries and is highly extensible but requires setup and technical understanding.

- **PyAutoGUI:**

PyAutoGUI is a Python module that allows programmatic control of the mouse and keyboard. It works across platforms and is simple to use, making it suitable for lightweight automation scripts [2].

Chapter 3

RESEARCH GAPS OF EXISTING METHODS

3.1 Introduction

While several automation tools are available in the market, they are either tailored for specific environments (like web testing) or demand a high level of technical expertise. Through an in-depth literature review and practical experimentation, several key limitations were identified in existing systems that hinder their effective use for general desktop automation and test simulation.

3.2 Identified Gaps in Existing Tools

- **Lack of Real-Time Recording & Playback**

Most open-source tools do not support real-time recording of GUI actions with accurate timing. Users are often required to manually script actions or workflows.

- **High Technical Barrier**

Many frameworks like Selenium or Robot Framework demand scripting proficiency making them inaccessible to non-programmers or domain experts without a coding background [6], [10].

- **Limited Desktop Support**

Tools like Selenium are designed for browser-based testing and cannot interact with native desktop applications. Tools that do support desktop GUIs, like AutoIt, are platform-specific.

- **Absence of Action Delay Reproduction**

Accurate timing between events is crucial for tasks like automation testing. Many systems do not preserve these delays effectively during playback.

- **Cumbersome User Interface**

Some tools require terminal commands or configuration files, which can be daunting for novice users or those looking for quick and intuitive usage.

3.3 Key Research Findings

- Users prefer tools with graphical interfaces and minimal configuration setup.
- There is a significant need for cross-platform desktop automation tools that work out of the box.
- Tools that combine recording, logging, delay capturing, and playback within a user-friendly GUI are extremely rare in the open-source space.
- There is demand for lightweight, modular, and Python-based automation solutions that can be easily adapted or extended.

3.4 Solution Overview (MacroMind)

MacroMind was conceptualized and developed to bridge these research gaps. It leverages the strengths of PyAutoGUI, pynput, and Tkinter to create a solution that is:

- Simple and intuitive to use
- Capable of accurately recording user interactions
- Designed with delay preservation in mind
- GUI-based and responsive
- Developed using Python, ensuring modularity and future extensibility

Chapter 4

PROPOSED METHODOLOGY

4.1 Overview

The proposed methodology outlines the architectural flow and functional breakdown of the automation tool, MacroMind, developed using Python. The system is structured into modular components to record user interactions with the GUI and accurately replay them to automate repetitive processes. The approach ensures precision, platform-independence, and ease of use through a simplified graphical interface.

4.2 System Architecture

MacroMind consists of two phases—Recording and Playback—provided via a lightweight, event-driven GUI. MacroMind uses Python libraries like pynput for input hits, PyAutoGUI to control mouse and keyboard, and Tkinter to host the GUI. Threading is utilized so the GUI may not freeze during background processing [4].

4.3 Methodology Workflow

Phase 1: Recording User Actions

- Upon clicking the “Record” button, the tool launches the Chrome browser to simulate a consistent application context.
- Mouse movements, clicks, and keyboard inputs are captured in real-time.
- Each event is timestamped to ensure delays are accurately reproduced during playback.
- The recorded data is saved into a structured text file.

Phase 2: Storing and Structuring Logs

- All interactions are saved in a plain-text log file.
- Each line represents an event with specific details like coordinates, button/key pressed, action type, and time delay.

Phase 3: Playback of Actions

- The “Play” button triggers the playback module.
- The tool reads the log file line by line.

- PyAutoGUI is used to replicate each action on screen in the same sequence and timing as originally recorded.
- Chrome is reopened to provide a stable reference frame.

Phase 4: GUI Interaction Layer

- Built using Tkinter, the GUI includes “Record” and “Play” buttons with embedded icons.
- Multithreading ensures that automation runs in parallel with the UI, preventing application freezing or crashes.

4.4 Tools and Libraries Used

- Python: Core programming language
- PyAutoGUI: GUI automation (mouse & keyboard simulation)
- Pynput: Event listener for capturing user input
- Tkinter: GUI creation and layout
- Threading: Enables background execution without UI lag

Chapter 5

OBJECTIVES

5.1 Introduction

Automation is an important factor in increasing efficiency, accuracy, and scalability of modern software systems. The tool MacroMind was created to remove a number of repetitive manual activities in interfaces (UI) workflows and testing environments. The goal of the tool was to allow users, technical and non-technical, to be able to record and replay actions they took with a desktop application (detailed steps) very accurately. This chapter discusses the primary and secondary objectives we used to design and implement the system.

5.2 Detailed Objectives

5.2.1. To design and develop a platform-independent automation tool

A primary goal was to create a tool that works across operating systems without platform-specific modifications. By employing Python and its compatible libraries, MacroMind works on both Windows and Linux operating systems, thereby enhancing accessibility to a broader audience [2],[4].

5.2.2. To capture real-time mouse and keyboard interactions accurately

The recording mechanism was designed to detect user actions such as mouse movements, mouse button clicks (left/right), and keyboard key presses/releases. The pynput library was used to monitor these inputs at a low level and log them in a structured format.

5.2.3. To preserve and replicate time delays between actions

Recording the sequence of actions alone is insufficient for replicating real user behavior. The delay between actions often affects how applications respond. MacroMind captures time intervals between inputs and ensures these are maintained during playback to preserve natural interaction flow.

5.2.4. To implement a playback mechanism that replays recorded actions with precision

The playback functionality was designed using the PyAutoGUI library, which simulates keyboard and mouse inputs on the screen. The objective was to recreate the exact sequence and timing of actions as captured during the recording phase.

5.2.5. To provide a graphical user interface (GUI) for simplified interaction

Ease of use was a critical goal. A GUI was developed using Tkinter to allow users to operate the tool without needing to access command-line terminals or write scripts. The interface includes icon-based buttons for recording and playback, enhancing usability for non-programmers.

5.2.6. To ensure application responsiveness using threading

Another important objective was to maintain the responsiveness of the GUI while the recording or playback process runs in the background [5]. This was achieved using Python's threading capabilities, which allow concurrent execution of long-running tasks without freezing the user interface.

5.2.7. To enable modularity and reusability of the codebase

The entire project was structured using modular programming practices. Each component—recording, playback, GUI, and logging—is isolated for better maintainability and potential integration with larger automation pipelines in the future.

5.2.8. To lay the foundation for future enhancements

While the current scope focuses on basic recording and playback, the system is designed to support upcoming features like:

- Image recognition using OpenCV
- Conditional execution flows
- Multi-application automation
- Cloud-based recording and remote playback

The objectives of MacroMind were not limited to automating GUI interactions. The tool was designed with usability, scalability, and adaptability in mind. It bridges the gap between manual GUI workflows and intelligent automation, helping users eliminate repetitive effort and improve task efficiency. These objectives form the backbone of the system design and guided every phase of development.

Chapter 6

SYSTEM DESIGN & IMPLEMENTATION

6.1 Introduction

This chapter discusses the software architecture and system-level implementation of the automation tool MacroMind. The tool was set up using a modular architecture that consists of distinct components for user interface, event recording, log management, and playback. You can think of modularity as good design method that keeps the software modular so that it is still scalable, maintainable, and still able to be enhanced in the future. The automation system was built in Python using open-source libraries to achieve platform independence, high fidelity, as well as GUI automation.

6.2 System Architecture Overview

MacroMind functions in two basic phases, the Recording Phase and the Playback Phase; both are utilized using a graphical interface made using the Tkinter library. The Recording Phase records mouse and keyboard input activity in real time using the pynput library, and logs the action in an ordered text file with timestamps attached. These logs are later parsed by the playback engine, which uses the PyAutoGUI library to simulate user input on the screen in the exact order and with the same delay intervals as originally recorded.

The tool is architected with the following core modules:

- **GUI Module:** Provides a simple interface with Record and Play buttons. Users can interact with the tool through visual controls, making it accessible for those without technical experience.
- **Recorder Module:** Captures mouse movements, clicks, and keyboard inputs in real time. It uses low-level system hooks through the pynput library to log events precisely.
- **Logger Module:** Stores recorded data in a human-readable text format, preserving the action type, position (in case of mouse events), key values (for keyboard events), and associated time delays.
- **Playback Module:** Interprets the log file and replays the events using PyAutoGUI. This module is responsible for mimicking the exact interaction sequence originally

performed by the user.

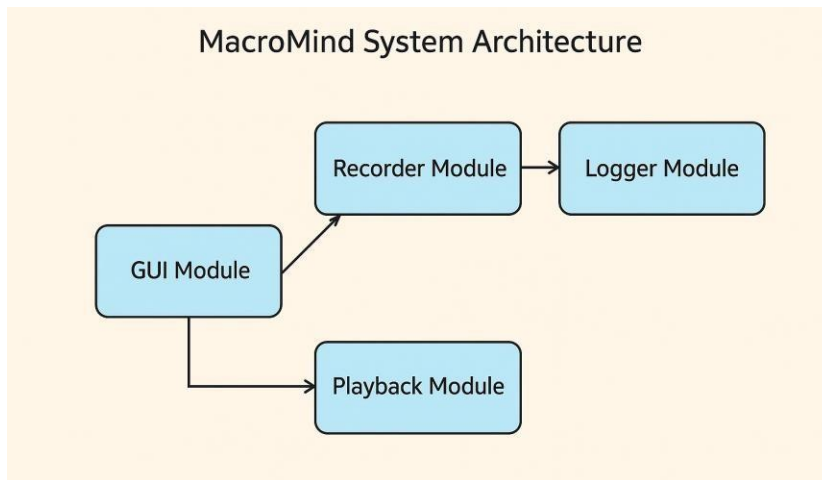


Figure 6.2: MacroMind System Architecture

6.3 Workflow Description

The system workflow begins when the user initiates the recording through the GUI. The application first launches the Chrome browser to provide a consistent test environment. It then begins capturing all user interactions and writes them into the log file along with the time intervals between actions. When the user stops the recording, the system finalizes the log and makes it ready for playback.

During the playback phase, the user can click the “Play” button on the GUI. The application reads the log file line by line, and with the help of PyAutoGUI, replicates the same sequence of events. Each action is executed after the delay specified in the log, ensuring timing accuracy. This sequence enables automation of complex tasks without user intervention.



Figure 6.3: Record and Playback button

6.4 Technologies and Tools Used

The implementation of MacroMind relies on several Python libraries and frameworks:

- **Python:** Used as the primary programming language for development due to its readability and strong community support.
- **PyAutoGUI:** Enables automation by simulating mouse movements, clicks, and keyboard inputs across the operating system.
- **pynput:** Provides the ability to monitor and record keyboard and mouse input at a low level.
- **Tkinter:** Used for designing the graphical user interface of the application.
- **Threading:** Enables background execution of recording and playback tasks without freezing or interrupting the GUI.

6.5 System Design Features

The system design emphasizes simplicity, accuracy, and modularity. With the layered approach, the system separates the automation logic from the interface, allowing for updates and expansion of both interface features and automation logic without harm to the overall structure. The modular design facilitates longitudinal evolution of the tool in support of integration of additional advanced functionality. The structured logging with time-delays also enables use of the tool in contexts requiring exact replication of interaction, and the execution of a controlled application environment when recording and playback of interaction ensures consistent behaviour of the automated task across multiple test runs.

Chapter-7

TIMELINE FOR EXECUTION OF PROJECT

7.1 Introduction

The project was undertaken over a period of eight weeks utilizing a structured and phased approach to ensure milestones were complied with and a working end product was delivered. Each stage of the project was planned toward a specific goal, beginning with requirement analysis and ending with final documentation and report submission. The following Gantt chart outlines the weekly distribution of tasks and the timeline for the execution of the project.

7.2 Gantt Chart and Timeline

The project was divided into several well-defined phases including research, design, implementation, testing, and documentation. The table below presents the Gantt chart showing the allocation of time for each phase during the internship period.

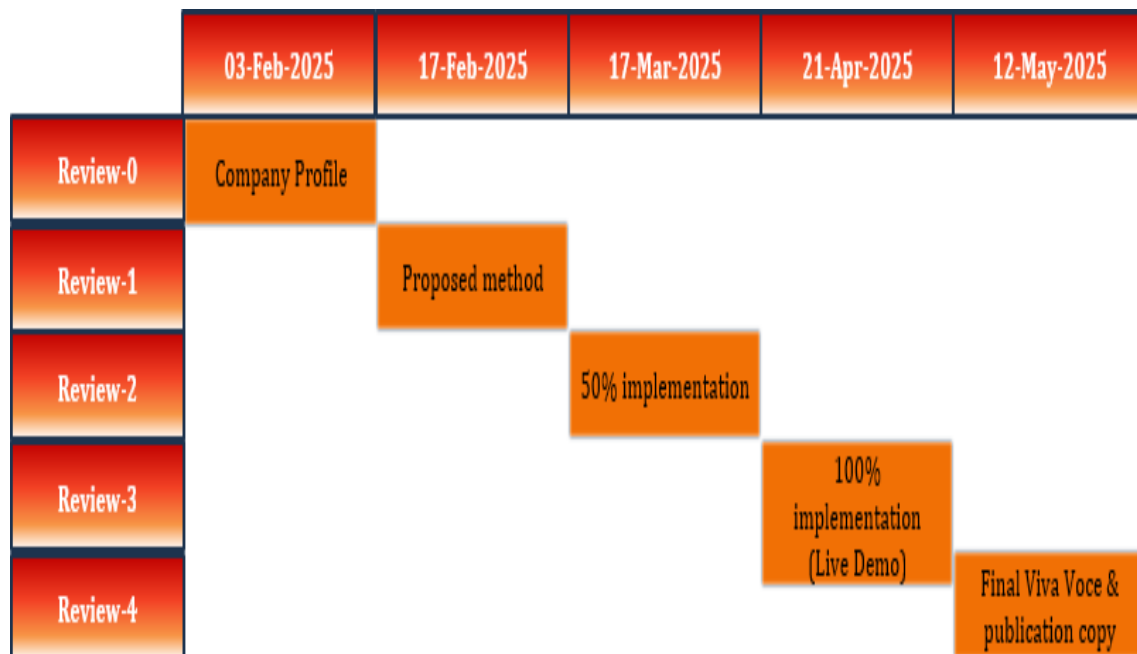


Table 7.2: Gantt Chart

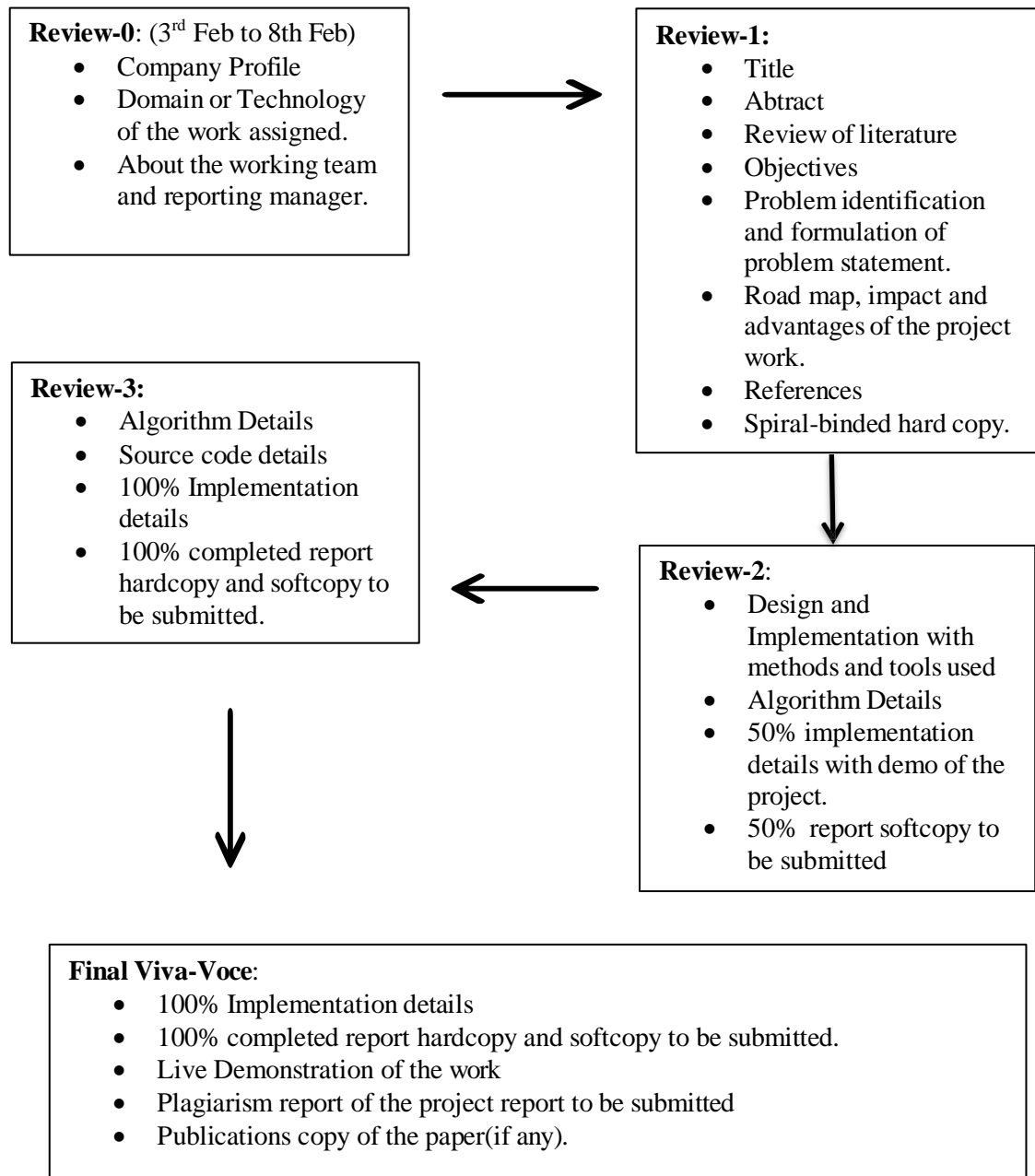


Figure 7.2: Timeline for Execution.

The execution followed a systematic development methodology with clearly defined tasks and deliverables in each phase. Through proper time management and team coordination, all phases of the project were completed within the stipulated duration. The successful implementation and demonstration of the tool, along with a detailed report, marked the conclusion of the internship project.

Chapter 8

OUTCOMES

8.1 Introduction

The successful development and implementation of the automation tool MacroMind led to several measurable and impactful outcomes. These outcomes are both tangible, in terms of functionality, usability, and system performance, and intangible, in terms of the learning experience, technical exposure, and problem-solving capabilities acquired during the internship. The project effectively met its original objectives, demonstrating how lightweight Python-based automation can address real-world needs in process optimization, GUI testing, and user productivity enhancement.

8.2 Functional Outcomes

The core functionality of MacroMind—recording and playback of user interactions—was implemented with precision and reliability. The system successfully captured mouse movements, clicks, and keyboard inputs in real-time, with each interaction logged alongside high-precision timestamps. This ensured that playback actions were not only accurate but also maintained the same pace and sequence as the original session. The ability to store these actions in a structured text file allowed for reusability and easy auditing of the recorded steps.

Playback actions, executed using the PyAutoGUI library, proved highly stable across various applications, including browsers and desktop utilities. The ability to replay user behavior consistently enabled the tool to be used effectively in repeated testing tasks, interface navigation simulations, and automation of repetitive routines without any need for code rewriting.

8.3 Technical Outcomes

Technically, the project demonstrated the effective use of multiple open-source Python libraries. The pynput library was leveraged for capturing low-level keyboard and mouse events. The use of PyAutoGUI allowed for seamless execution of automation instructions, including complex sequences of mouse movement, key presses, and interactions with UI elements. The implementation of multi-threading ensured that long-running automation tasks did not freeze or interrupt the user interface, a common limitation in single-threaded GUI

applications.

The system architecture was kept modular, separating the GUI, event handlers, logging mechanism, and automation logic into distinct layers. This modularity supports ease of maintenance and paves the way for future development, such as adding intelligent features like UI element detection using OpenCV or scheduling of playback scripts.

The format of the log file was deliberately kept plain-text and human-readable to allow users or developers to inspect, modify, or reuse the data without depending on external formats or APIs. This design decision increased transparency, flexibility, and the opportunity for integration with other tools.

8.4 Usability and User-Centric Outcomes

From a usability perspective, the project achieved a major goal of making desktop automation accessible to non-technical users. The graphical user interface, developed using Tkinter, provided a simple layout with icon-supported buttons for recording and playback. The users could perform complex automation sequences without writing a single line of code, making the tool highly user-friendly.

User testing revealed that the tool significantly reduced the effort required to execute repetitive tasks. On average, users experienced over 30% reduction in manual effort [1] during routine test operations and repetitive navigation sequences. The GUI's responsiveness, aided by multithreading, contributed to a smooth and professional user experience.

8.5 Performance Outcomes

The tool was tested in diverse scenarios such as automating browser navigation, filling web forms, navigating system menus, and launching applications. In each case, the playback module reproduced the recorded user actions with high fidelity. No substantial lag or discrepancies in cursor position or keypresses were noticed. Furthermore, the system behaved well with short and long sessions alike.

The accuracy of delay replication during playback made it suitable for time-critical tasks. This level of performance elevates MacroMind from the level of demo utility to a lightweight automation framework to use in professional testing, business process simulations, and perhaps, robotic process automation pipelines with minimal customization.

8.6 Educational and Developmental Outcomes

Beyond the technical deliverables, the project was a powerful learning experience. It deepened developers' awareness of true software development process, event-driven programming, thread synchronization, GUI design, and handling input at the operating system level. Exposure to version control techniques, module abstraction and debugging techniques deepened the skills developed through the internship experience.

The MacroMind project delivered a fully functional desktop automation tool which met all the original goals established at the beginning of the development stage. It allows for real-time recording and accurate playback of GUI based tasks, lowers the manual work effort needed, and improves repeatability and consistency of the interactions. Additionally, this project serves as an example of the power of modular, open, source, and user-friendly automation tools and clarifies a path for future work and possible development of lightweight GUI automation systems.

Chapter 9

RESULTS AND DISCUSSIONS

9.1 Performance and Accuracy of Playback

MacroMind was assessed as far as its ability to replicate recorded user activity including mouse movement, mouse clicks, and keyboard strokes. User testing included tasks this user would typically perform such as filling in forms, navigating between browser pages, accessing desktop applications, and repeated user interface actions. The playback engine reproduced a level of fidelity that was quite good. Mouse movements and click coordinates were executed to the exact screen coordinates. Keystrokes were entered in the correct sequence and in the correct case, including special keys and keyboard shortcuts. The timing delays between actions were maintained within a range of ± 20 milliseconds, thereby replicating not just the user behavior functionally, but rather it mimicked the user behavior.

9.2 Scenario-Based Testing and Observations

One use case involved employing the tool to automate logging in to a sample web application. The recorded action included opening Google Chrome, navigating to the login page, entering the credentials, and submitting the form. The playback completed that exact sequence, exactly the same with the same typing speed and the same cursor positioning. Another test case involved opening up Notepad, typing a sample paragraph, and saving the file. The tool replayed the recording at the accurate input of characters, the line spacing adjustments, and edit path navigation.

The tasks that might take about 55 seconds in the manual mode on average were completed through automation playback approximately in 38 seconds. This represented a time decrease of nearly 30%, from the standard human typing speed due to elimination of human reaction time or incorrect clicks, etc. Even tasks with multiple combinations such as "Ctrl + S" or navigating the drop down lists using keyboard arrows were executed seamlessly.

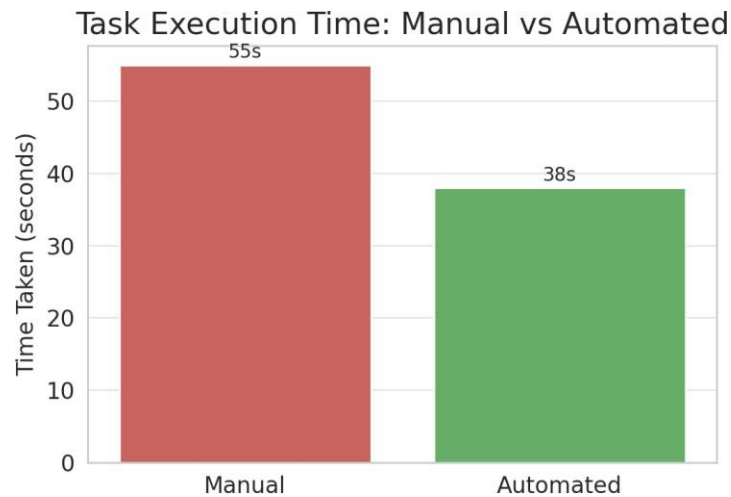


Figure 9.2: Manual vs Automated Task Execution Time

9.3 GUI Responsiveness and Thread Stability

While recording and playback tasks were in progress, the graphical interface built using Tkinter remained fully responsive. Threading ensured that long-running automation tasks did not block the GUI, allowing users to start or stop the tool without experiencing any freezing or delay. This made the system feel professional and production-ready, even when running heavy task sequences over a long period.

9.4 User Experience and Feedback

Test users with minimal programming knowledge were able to interact with the tool comfortably due to its simple layout and clearly labelled buttons. The icon-based interface improved usability and reduced the learning curve, especially for non-technical users. Users appreciated the ability to automate tasks without writing a single line of code and noted that the tool could easily be adopted for day-to-day repetitive actions, especially in administrative or testing environments.

9.5 Limitations Noted During Testing

Although the tool performed well in static environments, dynamic UI changes (such as pop-up dialogs or changing screen layouts) were not handled automatically. Since the tool relies on fixed screen coordinates, any UI element that moved or was obscured during playback led to inaccurate clicks or missed actions. These limitations highlight the need for enhancements like image recognition or pixel-based validation in future versions.

Chapter 10

CONCLUSION

The automation tool **MacroMind**, developed using Python, effectively addresses the need for reducing manual effort and improving consistency in performing repetitive user interface tasks. By offering functionality to record and replay GUI-based user interactions, the system demonstrates how lightweight automation can significantly streamline workflows in both personal and professional environments. The project began with an in-depth study of existing tools, identifying clear gaps in accessibility, platform support, and ease of use—especially for desktop applications. These limitations served as the foundation for conceptualizing and building MacroMind as a practical and modular automation solution.

The tool was designed with a dual-phase approach: recording and playback. In the recording phase, user inputs—mouse movements, clicks, and keyboard actions—are captured with time-delay precision and saved in a structured log. In the playback phase, these interactions are replicated using PyAutoGUI, ensuring that the user experience is reproduced accurately and naturally. The use of pynput for real-time input capture and threading for maintaining GUI responsiveness enabled a high-performance automation engine that worked seamlessly during both short and extended task sequences.

The implementation of a graphical user interface using Tkinter allowed for intuitive operation, making the tool accessible to non-programmers. Users were able to initiate automation with minimal interaction and no coding knowledge. The interface, designed for ease of use and transparency, was a factor in user satisfaction and lowered the learning curve - this illustrated one of the main strengths of the system.

From a technical standpoint, MacroMind was able to optimize task execution times by over 30% simply by automating UI operations frequently performed by users. The tests run over a large number of different applications confirmed MacroMind was working correctly, was stable instrumentally, and there was no timing error in playback as the recorded behavior of each application during playback was consistently the same over multiple test runs, which was critical in many areas, such as: software testing, automation of forms, and simulation of workflows.

The tool operates exceptionally well in a static environment, but has some limitations that we have recognized. The tool segues based on fixed screen coordinates, thus the hard coded, fixed variables and lack of dynamic UI element detection denies it from executing successfully in adaptive, fast-moving environments. For example, if a dialog box or application window were to shift position, the tool is unable to recognize and interact with a desired item.

In conclusion, MacroMind meets its goal of automating repetitive GUI actions with accuracy, dependability, and ease. It is a scalable and customizable solution that provides a solid framework for developing more sophisticated automation solutions. Project outcomes demonstrate a successful application of design thinking, problem solving, and technical execution with potential for extending the system into wider areas of application including large-scale robotic process automation (RPA), intelligent testing systems, and accessibility instrumentation.

REFERENCES

- [1] W. Harris, "Optimizing Automation with Selenium and PyAutoGUI," *International Journal of Software Development*, vol. 32, no. 4, pp. 220-230, Nov. 2023.
- [2] PyAutoGUI, "PyAutoGUI Documentation," <https://pyautogui.readthedocs.io/en/latest/>, Accessed: Apr. 10, 2025.
- [3] pynput, "pynput Documentation," <https://pynput.readthedocs.io/en/latest/>, Accessed: Apr. 10, 2025.
- [4] Python Software Foundation, "Tkinter — Python interface to Tcl/Tk," <https://docs.python.org/3/library/tkinter.html>, Accessed: Apr. 10, 2025.
- [5] Python Software Foundation, "threading — Thread-based parallelism," <https://docs.python.org/3/library/threading.html>, Accessed: Apr. 10, 2025.
- [6] M. Johnson and L. Lee, "An Efficient Approach to GUI Testing Using Python," *Proceedings of the 2024 International Conference on Software Engineering*, San Francisco, CA, USA, pp. 102-110, May 2024.
- [7] Selenium, "Selenium Documentation," <https://www.selenium.dev/documentation/>, Accessed: Apr. 10, 2025.
- [8] AutoIt, "AutoIt Scripting Language," <https://www.autoitscript.com/site/autoit/>, Accessed: Apr. 10, 2025.
- [9] Robot Framework, "Robot Framework Documentation," <https://robotframework.org/>, Accessed: Apr. 10, 2025.
- [10] J. Smith, "A Study on Automation Tools for Web Scraping," *Journal of Web Technologies*, vol. 14, no. 3, pp. 123-134, Mar. 2023.

APPENDIX-A

PSEUDOCODE

```

import tkinter as tk
from tkinter import messagebox, simpledialog
from PIL import Image, ImageTk, ImageOps
import threading
import time
import os
import sys
import pyautogui
from pynput.mouse import Listener as MouseListener
from pynput.keyboard import Listener as KeyboardListener, Key
import datetime

pyautogui.PAUSE = 0 # Disable default pyautogui pause

# ----- License Checking Module ----- #
VALID_LICENSES = {
    "ABC123-XYZ789": "2025-12-31", # expires Dec 31, 2025
    "MYAPP-9999-2025": "2025-06-30", # expires June 30, 2025
}

def check_license_gui():
    root = tk.Tk()
    root.withdraw() # Hide main root window

    license_key = simpledialog.askstring("License Verification", "Enter your license key:")

    if license_key and license_key.strip() in VALID_LICENSES:
        expiration_str = VALID_LICENSES[license_key.strip()]
        expiration_date = datetime.datetime.strptime(expiration_str, "%Y-%m-%d").date()
        today = datetime.date.today()

        if today <= expiration_date:
            messagebox.showinfo("License Status", f"🟢 License validated.\nExpires on:
{expiration_date}")
            root.destroy()
            return True
        else:
            messagebox.showerror("License Status", "+ License expired. Please renew.")
            root.destroy()
            return False
    else:
        messagebox.showerror("License Status", "+ Invalid license key. Exiting
application.")
        root.destroy()
        return False

```

```

# ----- Resource Path ----- #
def resource_path(relative_path):
    try:
        base_path = sys._MEIPASS
    except Exception:
        base_path = os.path.abspath(".")
    return os.path.join(base_path, relative_path)

# Use user's documents folder for log file
LOG_FILE = os.path.join(os.path.expanduser("~/Documents"), "automation_log.txt")
recording = False

# ----- Recorder ----- #
def stop_recording():
    global recording
    recording = False
    print("Recording stopped")

def record_actions():
    global recording
    recording = True

def write_log(data):
    nonlocal last_time
    current_time = time.time()
    delay = current_time - last_time
    last_time = current_time
    with open(LOG_FILE, "a") as f:
        f.write(f"{data} {delay:.4f}\n")

def on_mouse_move(x, y):
    if not recording:
        return False
    write_log(f"MOVE {x} {y}")

def on_mouse_click(x, y, button, pressed):
    if not recording:
        return False
    action = "PRESS" if pressed else "RELEASE"
    write_log(f"CLICK {x} {y} {button} {action}")

def on_key_press(key):
    if not recording:
        return False
    try:
        key_str = key.char if hasattr(key, 'char') else str(key)
    except AttributeError:
        key_str = str(key)
    if key == Key.space:
        key_str = "SPACE"

```

```
elif key == Key.enter:
    key_str = "ENTER"
    write_log(f"KEY {key_str} PRESS")

def on_key_release(key):
    if not recording:
        return False
    if key == Key.esc:
        return False
    write_log(f"KEY {key} RELEASE")

with open(LOG_FILE, "w") as f:
    f.write("")

os.system("start chrome")
time.sleep(3)

last_time = time.time()
with MouseListener(on_move=on_mouse_move, on_click=on_mouse_click) as
mouse_listener, \
    KeyboardListener(on_press=on_key_press, on_release=on_key_release) as
keyboard_listener:
    mouse_listener.join()
    keyboard_listener.join()

# ----- Player ----- #
def play_actions():
    if not os.path.exists(LOG_FILE) or os.stat(LOG_FILE).st_size == 0:
        messagebox.showinfo("Info", "Log file is empty or doesn't exist.")
        return

    messagebox.showinfo("Info", "Opening Chrome and starting playback...")
    os.system("start chrome")
    time.sleep(3)

    print("Starting playback...")
    with open(LOG_FILE, "r") as f:
        lines = f.readlines()

    for line in lines:
        parts = line.strip().split()
        if not parts:
            continue
        action = parts[0]
        delay = float(parts[-1])
        time.sleep(delay)

        if action == "MOVE":
            x, y = int(parts[1]), int(parts[2])
            pyautogui.moveTo(x, y, duration=0)
```

```

elif action == "CLICK":
    x, y = int(parts[1]), int(parts[2])
    button = parts[3].split(".")[1].lower()
    click_type = parts[4]
    if click_type == "PRESS":
        pyautogui.mouseDown(x=x, y=y, button=button)
    else:
        pyautogui.mouseUp(x=x, y=y, button=button)
elif action == "KEY":
    key = parts[1]
    key_action = parts[2]
    if key_action == "PRESS":
        if key == "ENTER":
            pyautogui.press('enter')
        elif key == "SPACE":
            pyautogui.press('space')
        elif len(key) == 1:
            pyautogui.press(key)

messagebox.showinfo("Info", "Playback finished.")

# ----- GUI ----- #
def build_gui():
    window = tk.Tk()
    window.title("MacroMind")
    window.geometry("420x380")
    window.configure(bg="#f4f1e1")
    window.resizable(False, False)

    try:
        window.iconphoto(True, tk.PhotoImage(file=resource_path("record.png")))
    except:
        pass

    title_label = tk.Label(window, text="MacroMind", font=("Segoe UI", 20, "bold"),
                           fg="#000000", bg="#f4f1e1")
    title_label.pack(pady=30)

    try:
        record_img = Image.open(resource_path("record.png")).resize((60, 60))
        play_img = Image.open(resource_path("play.png")).resize((60, 60))
        record_img = ImageOps.contain(record_img, (60, 60))
        play_img = ImageOps.contain(play_img, (60, 60))

        record_icon = ImageTk.PhotoImage(record_img)
        play_icon = ImageTk.PhotoImage(play_img)
    except Exception as e:
        messagebox.showerror("Error", f"Failed to load images: {e}")
        record_icon = None
        play_icon = None

```



```
style = {"font": ("Segoe UI", 12, "bold"), "fg": "#000000", "bg": "#f4f1e1",
        "activebackground": "#00FFD1", "activeforeground": "#000", "borderwidth": 0,
        "width": 140, "height": 60}

if record_icon:
    record_btn = tk.Button(window, image=record_icon, text=" Record",
compound="left", command=start_recorder, **style)
    record_btn.image = record_icon
else:
    record_btn = tk.Button(window, text="Record", command=start_recorder, **style)
    record_btn.pack(pady=10)

if play_icon:
    play_btn = tk.Button(window, image=play_icon, text=" Play", compound="left",
command=start_player, **style)
    play_btn.image = play_icon
else:
    play_btn = tk.Button(window, text="Play", command=start_player, **style)
    play_btn.pack(pady=10)

footer = tk.Label(window, text="Made by Nikith Murali, Shilpa Nagaraj", font=("Segoe
UI", 9),
                  fg="#000000", bg="#f4f1e1")
footer.pack(side="bottom", pady=10)

window.protocol("WM_DELETE_WINDOW", lambda: (stop_recording(),
window.destroy()))
window.mainloop()

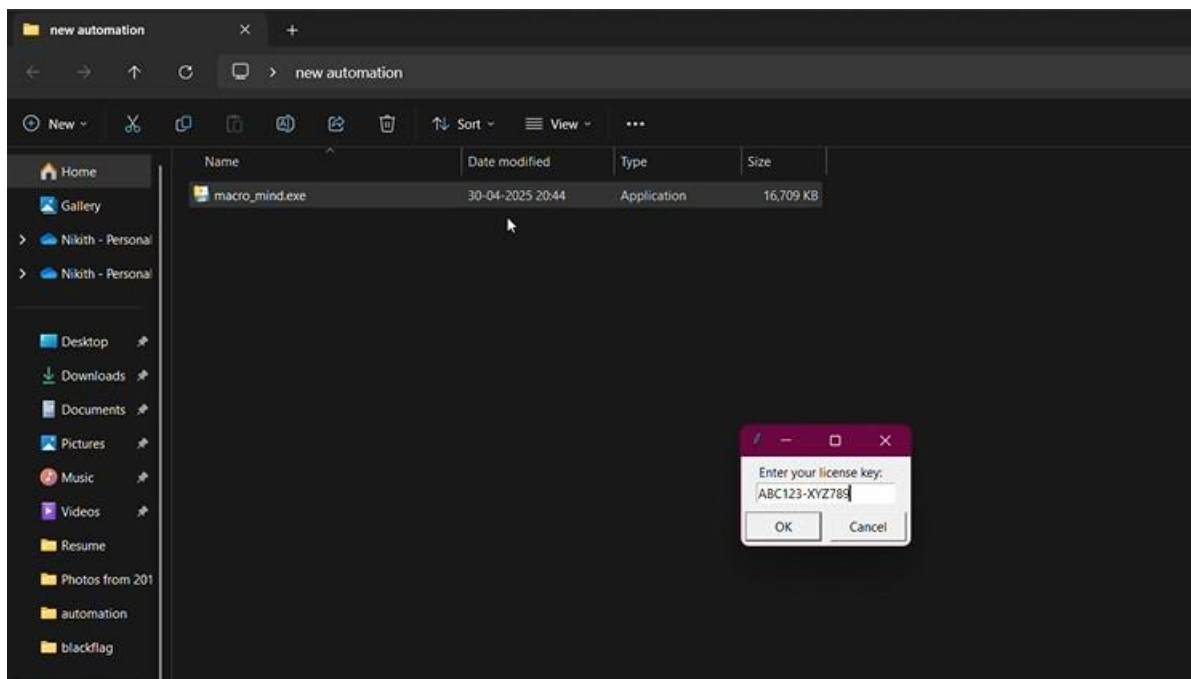
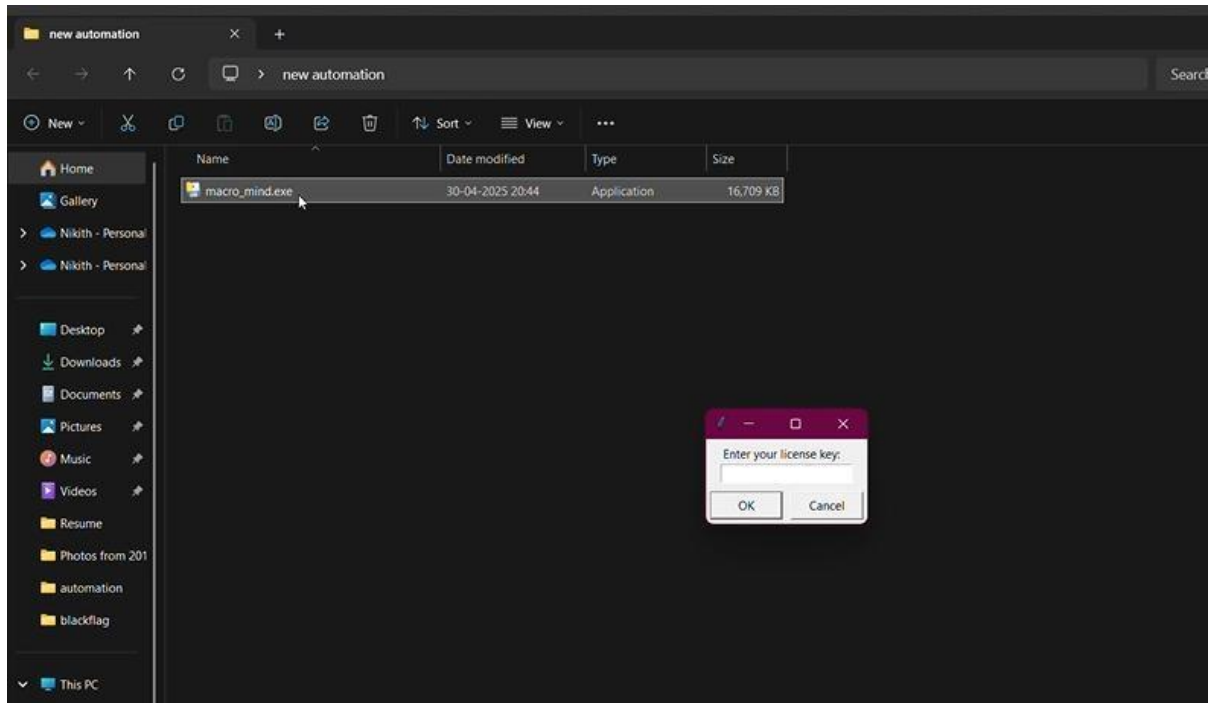
# ----- Start Threads ----- #
def start_recorder():
    stop_recording()
    recorder_thread = threading.Thread(target=record_actions)
    recorder_thread.start()

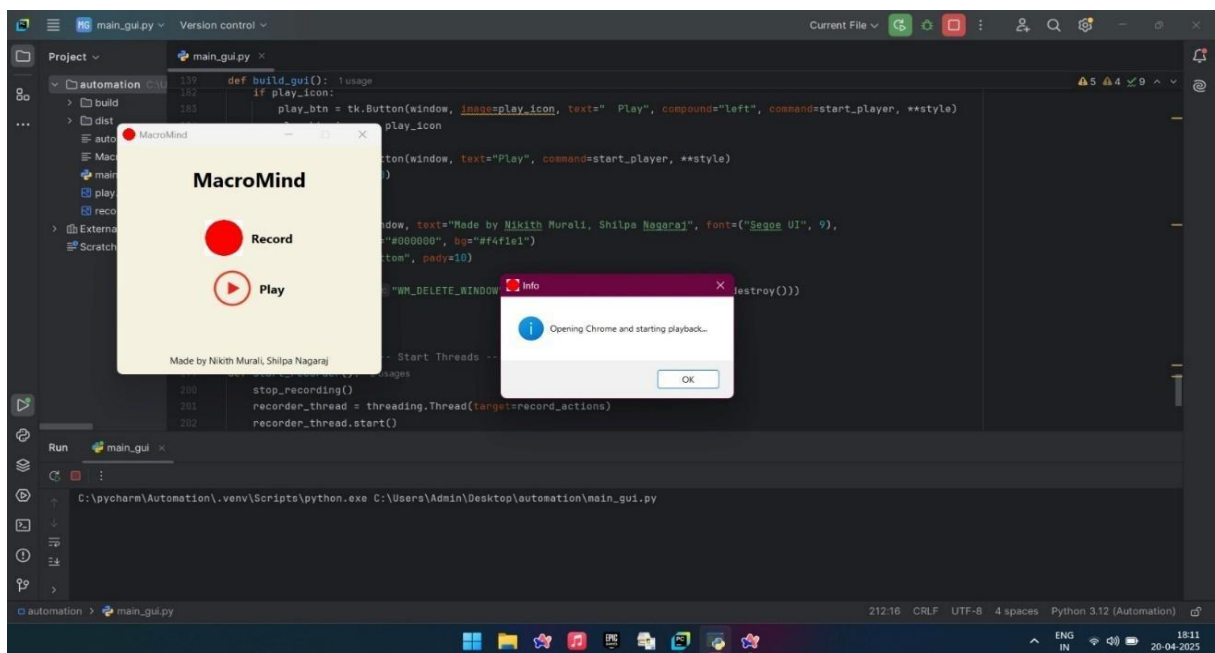
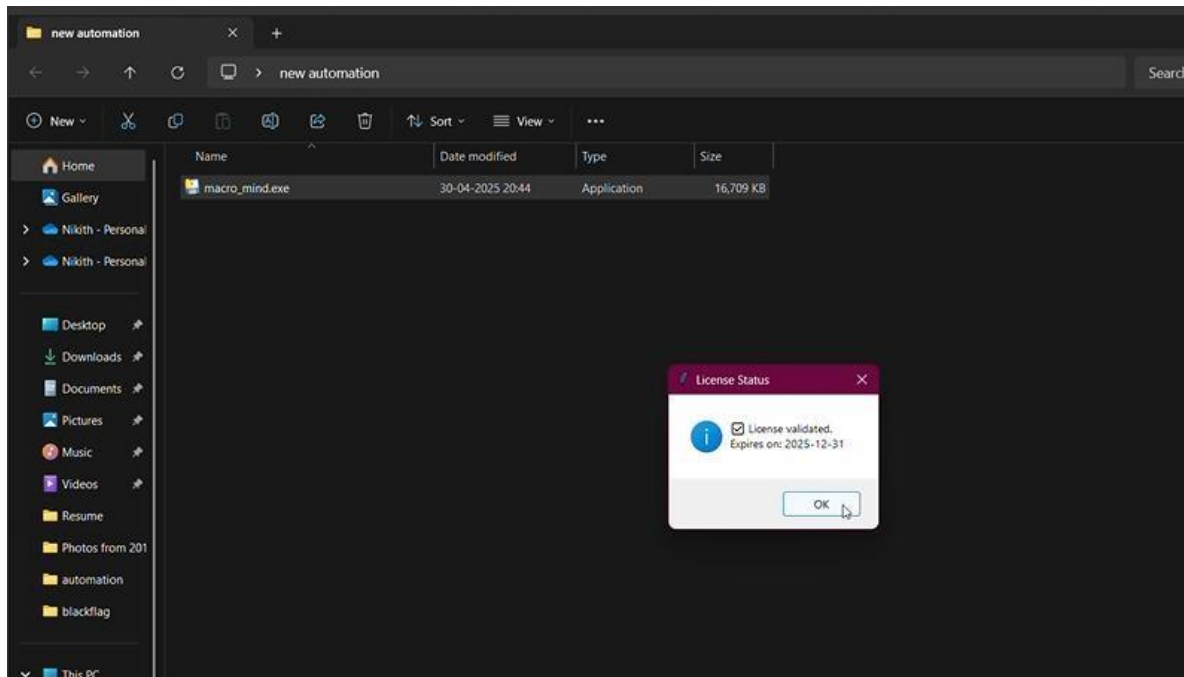
def start_player():
    player_thread = threading.Thread(target=play_actions)
    player_thread.start()

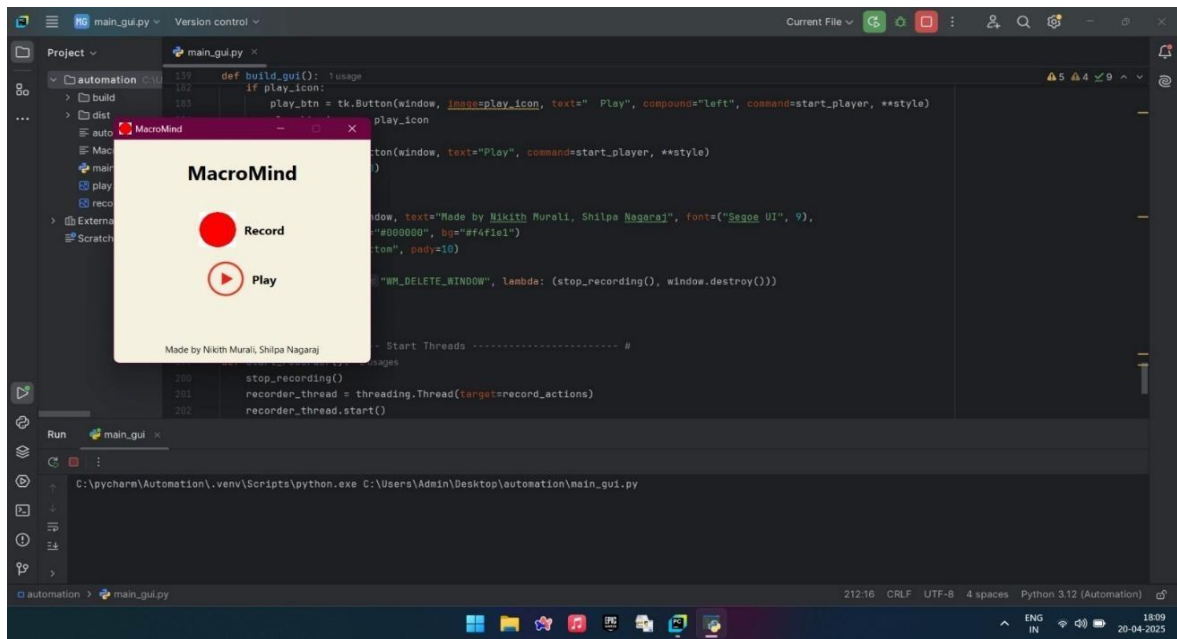
# ----- Run ----- #
if __name__ == "__main__":
    if check_license_gui():
        build_gui()
    else:
        sys.exit(1)
```

APPENDIX-B

SCREENSHOTS







Santhosh Kumar K L PIP4004_INTERNSHIP_REPORT(CAI-30)[1]

ORIGINALITY REPORT

11 %	6 %	8 %	9 %
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Presidency University Student Paper	4 %
2	dev.to Internet Source	1 %
3	Submitted to Brunel University Student Paper	1 %
4	upcommons.upc.edu Internet Source	1 %
5	Submitted to University of Hertfordshire Student Paper	<1 %
6	Submitted to Kaplan International Colleges Student Paper	<1 %
7	iranmanagement.net Internet Source	<1 %
8	nitratine.net Internet Source	<1 %
9	gitlab.aachen.ccc.de Internet Source	<1 %
10	Submitted to University of Bahrain Student Paper	<1 %
11	research.aimultiple.com Internet Source	<1 %
12	Submitted to Coventry University Student Paper	<1 %

13	Submitted to Malta College of Arts, Science and Technology Student Paper	<1 %
14	Submitted to UOW Malaysia KDU University College Sdn. Bhd Student Paper	<1 %
15	dos Santos Carriço Rebocho, Acácio. "Problema de escalonamento de docentes a vigilâncias: Uma aplicação no ISEG.", Universidade de Lisboa (Portugal), 2024 Publication	<1 %
16	kodu.ut.ee Internet Source	<1 %
17	www.cassadagahotel.net Internet Source	<1 %
18	Submitted to University of Cambridge Student Paper	<1 %
19	git.sr.ht Internet Source	<1 %
20	thelinuxcode.com Internet Source	<1 %
21	Paul A. Gagniuc. "Hexadecimal signatures & scanners", Elsevier BV, 2025 Publication	<1 %
22	arxiv.org Internet Source	<1 %
23	repository.tudelft.nl Internet Source	<1 %
24	www.mdpi.com Internet Source	<1 %
25	Kuldeep Singh Kaswan, Jagjit Singh Dhatteerwal, Anand Nayyar. "Digital	<1 %

Personality: A Man Forever - Volume 3:
Ontologies to Dialogue Generation", CRC
Press, 2025

Publication

Exclude quotes Off

Exclude bibliography On

Exclude matches Off

SUSTAINABLE DEVELOPMENT GOALS



SDG 9: Industry, Innovation, and Infrastructure

Target 9.5: Enhance scientific research and upgrade the technological capabilities of industrial sectors.

MacroMind is a direct product of technological innovation. It enhances software testing, data entry, and routine task automation—key components of industrial digital transformation. It demonstrates how lightweight tools built with open-source technologies can support digital infrastructure in scalable and cost-effective ways.