WORD GUESSING GAME SYSTEM

A MINI PROJECT REPORT

Submitted by

MANIKANDAN K - 220701159
LITHAN G - 220701143
LOGESH D - 220701144

In partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE

RAJALAKSHMI ENGINEERING COLLEGE (AUTONOMOUS)

THANDALAM
CHENNAI-602105

2023-2024

BONAFIDE CERTIFICATE

Certified that this project report "**WORD GUESSING GAME**" is the bonafide work of " **LITHAN G(220701143), MANIKANDAN K (220701159), LOGESH D (220701144)"** who carried out the project work under my supervision. Submitted for the Practical Examination held on_____

SIGNATURE

Dr.R.SABITHA
Professor and II Year Academic Head
Computer Science and Engineering,
Rajalakshmi Engineering College
(Autonomous),
Thandalam, Chennai - 602 105

SIGNATURE

Ms.D.KALPANA
Assistant Professor (SG),
Computer Science and Engineering,
Rajalakshmi Engineering College
(Autonomous),
Thandalam, Chennai - 602 105

INTERNAL EXAMINER

EXTERNAL EXAMINER

ABSTRACT

This document describes the implementation of a word-guessing game using Python and the Tkinter library for the graphical user interface (GUI). The game, encapsulated in the `GuessingGame` class, offers an engaging and interactive experience where players attempt to guess a randomly selected word from a predefined list. The game runs in fullscreen mode, providing an immersive environment. The interface includes labels for instructions, masked words, number of attempts, and wrong guesses, as well as entry fields and buttons for submitting guesses, playing again, and exiting the game.

The game begins by randomly selecting and masking a word, and players guess letters to reveal it. Correct guesses update the masked word, while incorrect guesses decrease the attempts left. The game checks for win or loss conditions, displaying appropriate messages and restarting as needed. The exit functionality includes a confirmation prompt to prevent accidental exits. This game effectively combines classic word puzzle

elements with modern GUI features, demonstrating the power and flexibility of Tkinter for creating interactive Python applications.

TABLE OF CONTENTS

CHAPTER-1

INTRODUCTION

Welcome to the world of the Guessing Game! This interactive game is designed to entertain and challenge players as they embark on a journey to guess words and test their vocabulary skills. Whether you're a casual gamer looking for some fun or a word enthusiast seeking a mental challenge, this game offers an engaging experience for all.

1.1 Introduction

The Guessing Game is a classic yet captivating game where players attempt to uncover a hidden word by guessing individual letters. With each correct guess, letters are revealed, gradually unveiling the mystery word. However, incorrect guesses come with a penalty, reducing the number of attempts available to the player.

1.2 Objectives

The primary objective of the Guessing Game is to provide an enjoyable and stimulating experience for players of all ages. Key objectives include:

1. Entertainment: The game aims to entertain players by providing an interactive and engaging gameplay experience.
2. Educational Value: While entertaining players, the game

also serves an educational purpose by encouraging vocabulary expansion and cognitive skills development.

3. Challenge: The game offers a level of challenge that is both rewarding and motivating, encouraging players to improve their word-guessing abilities over time.
4. Scoring System: Through a scoring system, the game incentivizes players to make correct guesses while penalizing incorrect ones, adding an element of strategy to the gameplay.

1.3 Modules

The Guessing Game comprises several modules that work together seamlessly to deliver a smooth and immersive gaming experience. These modules include:

1. User Interface (UI): The UI module handles the graphical user interface elements, such as text labels, input fields, and buttons, providing an intuitive and user-friendly interface for players.
2. Word Database Integration: This module integrates with a MySQL database to fetch a pool of words for players to guess. By retrieving words from a database, the game ensures a diverse and ever-expanding selection of word options.
3. Game Logic: The core game logic module manages the

gameplay mechanics, including word selection, letter guessing, scoring, and win/lose conditions. It governs the flow of the game and ensures a fair and enjoyable experience for players.

4. Scorekeeping: The scorekeeping module tracks players' scores throughout the game, updating them based on correct and incorrect guesses. This module adds a competitive element to the game, motivating players to strive for higher scores.
5. Exit Handling: The exit handling module manages the termination of the game session, providing players with the option to exit the game gracefully while preserving their progress and scores.

These modules work together harmoniously to deliver a cohesive and entertaining gaming experience for players, making the Guessing Game a compelling choice for leisure and entertainment.

In conclusion, the Guessing Game offers an immersive and intellectually stimulating experience that combines entertainment with education. With its diverse word database, engaging gameplay mechanics, and user-friendly interface, the game promises hours of fun for players of all ages. So, embark on this word-guessing adventure and see how many mystery words you can uncover!

CHAPTER-2
Survey of Technologies

2.1 Software Description:

The Guessing Game employs a combination of software technologies to deliver an engaging user experience. These technologies encompass database management and programming languages, each playing a crucial role in the game's functionality.

2.2 Languages:

2.2.1 MySQL:

MySQL serves as the backend database management system for the Guessing Game. It is an open-source relational database known for its reliability and performance. MySQL stores the pool of words used in the game, allowing for dynamic word selection during gameplay. Through SQL queries, the game accesses and retrieves words from the database, ensuring a diverse and extensive word bank for players to guess from. MySQL's robust features, including data integrity and transaction support, contribute to the smooth operation of the game's database.

2.2.2 Python:

Python is the primary programming language used to develop the Guessing Game's core logic and user interface. Python is renowned for its simplicity, readability, and extensive library support. Within the game, Python handles various functionalities, including word selection, letter guessing, scoring, and user input validation. Additionally, Python's Tkinter library is utilized for creating the graphical user interface (GUI) of the game. Python's versatility and rich library ecosystem streamline development and enhance the overall performance and usability of the Guessing Game.

By leveraging MySQL for database management and Python for application development, the Guessing Game offers a seamless and enjoyable gaming experience, combining the strengths of both technologies to create an interactive and entertaining word-guessing adventure.

CHAPTER-3

REQUIREMENT AND ANALYSIS

3.1 REQUIREMENT SPECIFICATION:

The Guessing Game is a word-guessing game developed using Python's Tkinter library for the GUI and MySQL for word retrieval. The game runs in full-screen mode, featuring various labels to guide the player, an entry widget for letter input, and buttons for submitting guesses, starting a new game, and exiting the application. Players are tasked with guessing letters of a randomly chosen word from a MySQL database, with each correct guess revealing parts of the word and each incorrect guess reducing the remaining attempts and score. The game starts with 100 points, losing 10 points for each incorrect guess.

Word selection is random from the `word_s` table in the `hello` database, with default words available if the database connection fails. The game logic includes validating guesses, updating the displayed word, and notifying the player of wins or losses. The application handles incorrect inputs gracefully and confirms with the user before exiting. The game aims to be user-friendly, responsive, reliable, maintainable, and portable, requiring Python, Tkinter, and MySQL. The code is modular and well-documented to facilitate easy updates and maintenance, ensuring a

smooth and engaging user experience.

3.2 HARDWARE AND SOFTWARE REQUIREMENTS:

The Guessing Game application requires the following hardware and software components to function effectively:

Hardware Requirements:

1. Processor:
   - Minimum: Dual-core processor
   - Recommended: Quad-core processor for enhanced performance

2. Memory (RAM):
   - Minimum: 2 GB
   - Recommended: 4 GB or more to ensure smooth operation

3. Storage:
   - Minimum: 200 MB of free disk space
   - Recommended: 500 MB to accommodate Python, necessary libraries, and MySQL

4. Display:
   - Minimum screen resolution: 1024x768 pixels
   - Should support full-screen mode

5. Input Devices:

- Keyboard

- Mouse

Software Requirements:

1. Operating System:

- Windows 7 or later

- macOS 10.13 (High Sierra) or later

- Linux (any modern distribution)

2. Python:

- Python 3.6 or later

3. Python Libraries:

- Tkinter (usually included with Python)

- mysql-connector-python (for MySQL database connectivity)

4. Database:

- MySQL Server 5.7 or later

- MySQL Workbench (optional, for database management)

5.Additional Software:

- A text editor or IDE (e.g., Visual Studio Code, PyCharm, or Sublime Text) for code editing

Installation and Setup Instructions:

1. Install Python:

- Download and install Python 3.6 or later from the [official Python website](https://www.python.org/).
- Ensure that the installation adds Python to your system PATH.

2. Install MySQL:

- Download and install MySQL Server from the [official MySQL website](https://dev.mysql.com/downloads/mysql/).
- Optionally, install MySQL Workbench for database management.

3. Install Python Libraries:

- Open a command prompt or terminal.
- Run the following command to install `mysql-connector-python`:

```sh
pip install mysql-connector-python
```

4. Setup MySQL Database:
   - Start MySQL Server.
   - Create the `hello` database and the `word_s` table:

     ```sql
     CREATE DATABASE hello;
     USE hello;
     CREATE TABLE word_s (words VARCHAR(255));
     INSERT INTO word_s (words) VALUES ('example'), ('words'), ('for'), ('the'), ('game');
     ```

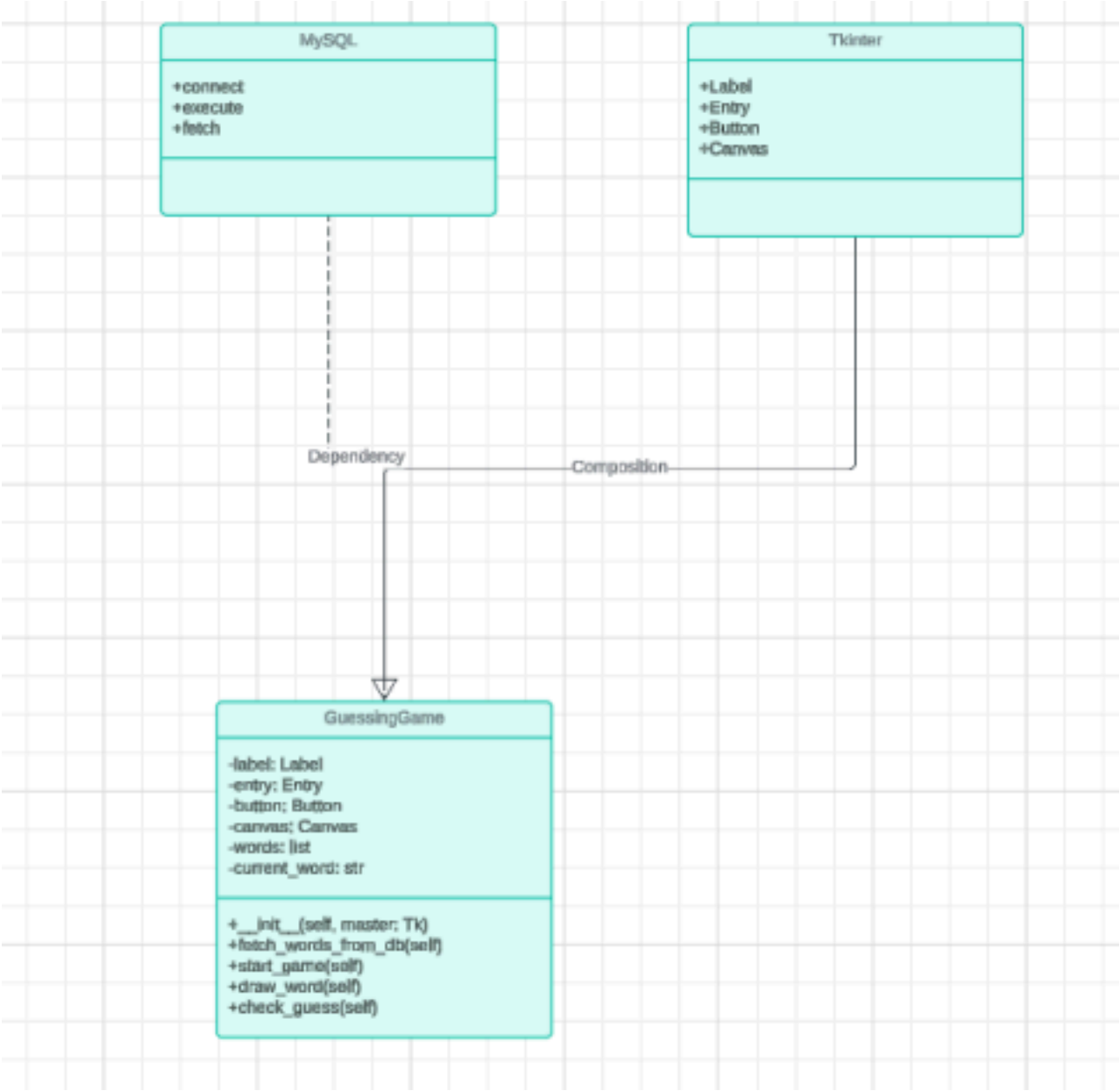
5. Run the Application:
   - Open the text editor or IDE.
   - Copy the provided code into a new Python file (e.g., `guessing_game.py`).
   - Run the script using the command:

     ```sh
     python guessing_game.py
     ```

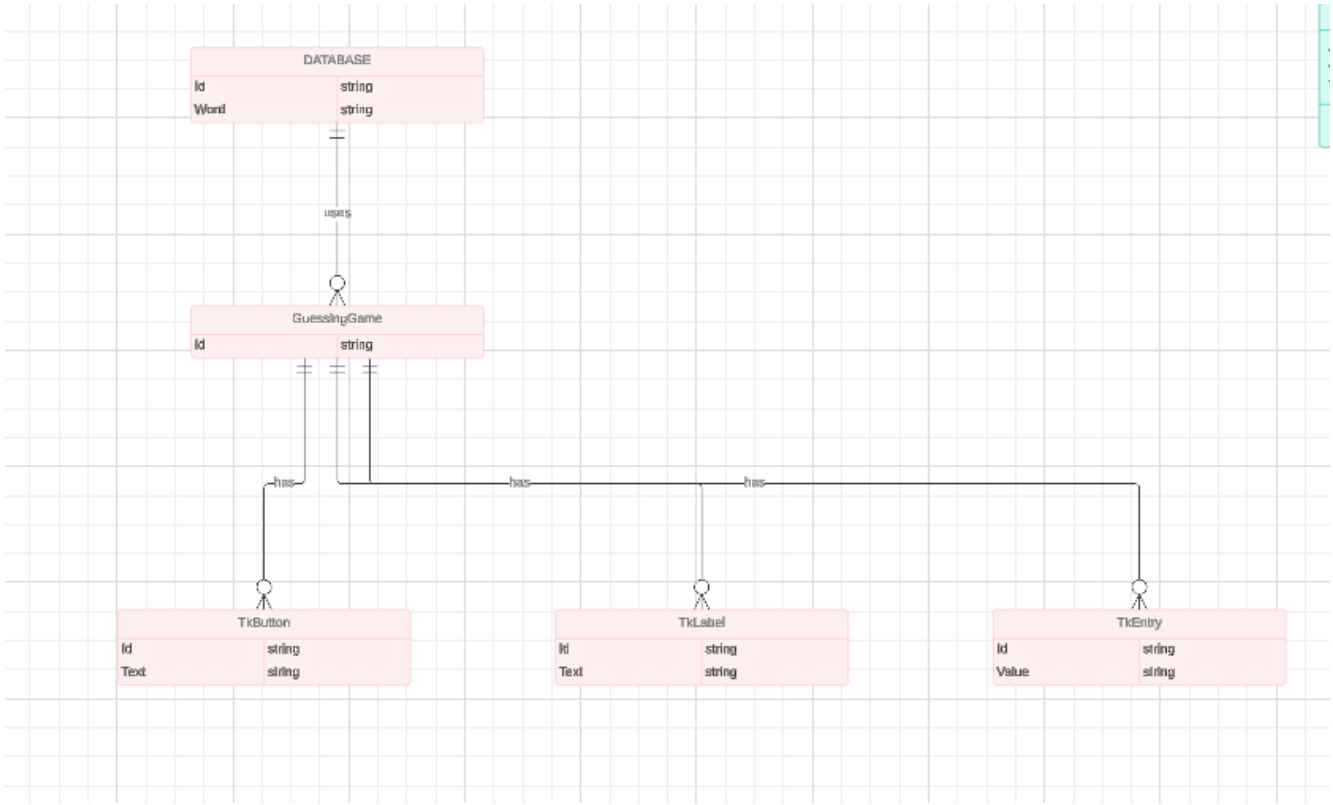
By adhering to these hardware and software requirements, the Guessing Game application should run efficiently, providing a smooth and

enjoyable user experience.

3.3 ARCHITECTURE DIAGRAM:



3.4 ER DIAGRAM:


DATABASE
Id
string
Word
string
uses
GuessingGame
Id
string
has
has
has
TkButton
Id
string
Text
string
TkLabel
Id
string
Text
string
TkEntry
Id
string
Value
string

CHAPTER-4

PROGRAM CODE

```
import random

import tkinter as tk

from tkinter import messagebox

import mysql.connector

class GuessingGame:
    def __init__(self, master):

        self.master = master

        self.master.title("Guessing Game")

        self.master.attributes('-fullscreen',

        True)
```

```
        self.words = self.fetch_words_from_db()

        self.current_word = ""

        self.acopy = ""

        self.masked_word = ""

        self.attempts = 0

        self.success = 0

        self.wrong_guesses = ""
        self.score = 0

        self.point_value = 100

        self.instructions = tk.Label(master, text="Guess the word! Enter a
letter and press 'Submit'.")
        self.instructions.pack(pady=20)

        self.canvas = tk.Canvas(master, width=400, height=50)

        self.canvas.pack()
```

```
        self.guess_label = tk.Label(master, text="Enter a letter:")

        self.guess_label.pack(pady=10)

        self.guess_entry = tk.Entry(master, width=3, font=('Helvetica', 18))

        self.guess_entry.pack(pady=10)

        self.submit_button = tk.Button(master, text="Submit",
command=self.check_guess, font=('Helvetica', 14))

        self.submit_button.pack(pady=10)
        self.attempts_label = tk.Label(master, text="", font=('Helvetica',
14))
        self.attempts_label.pack(pady=10)

        self.score_label = tk.Label(master, text="", font=('Helvetica', 14))

        self.score_label.pack(pady=10)

        self.wrong_label = tk.Label(master, text="", font=('Helvetica', 14))

        self.wrong_label.pack(pady=10)
```

```
        self.play_again_button = tk.Button(master, text="Play Again",
command=self.start_game, font=('Helvetica', 14))

        self.play_again_button.pack(pady=10)

                                self.exit_button = tk.Button(master, text="Exit",
command=self.exit_program, font=('Helvetica', 14))

        self.exit_button.pack(pady=10)

        self.start_game()
    def fetch_words_from_db(self):
        try:

            conn = mysql.connector.connect(

                host='localhost',

                user='root',

                password='Travler',

                database='hello'

            )
```

```
            cursor = conn.cursor()

            cursor.execute("SELECT words FROM word_s")

            words = [row[0] for row in cursor.fetchall()]

            conn.close()

            return words
        except mysql.connector.Error as err:

            messagebox.showerror("Database Error", f"Error: {err}")

            return ["default", "words", "if", "db", "fails"]


    def start_game(self):

        self.current_word = random.choice(self.words)

        self.acopy = self.current_word
```

```
        self.masked_word = ["_" for _ in self.current_word]

        self.attempts = len(self.current_word)

        self.success = 0
        self.wrong_guesses = ""

        self.score = self.point_value

        self.draw_word()
        self.attempts_label.config(text=f"Attempts left: {self.attempts}")

        self.score_label.config(text=f"Score: {self.score}")

        self.wrong_label.config(text="Previous wrong guesses: None")

        self.guess_entry.delete(0, tk.END)

        self.instructions.config(text="Guess the word! Enter a letter and
press 'Submit'.")

    def draw_word(self):

        self.canvas.delete("all")
```

```
        for i, char in enumerate(self.masked_word):

            self.canvas.create_rectangle(50 * i, 0, 50 * (i + 1), 50,
outline="black", fill="white")

            self.canvas.create_text(50 * i + 25, 25, text=char,
font=('Helvetica', 20))

    def check_guess(self):
        ch = self.guess_entry.get().strip().lower()

        if not ch:

            messagebox.showinfo("Input Error", "Please enter a valid
letter.")

            return

        if len(ch) != 1:
            messagebox.showinfo("Input Error", "Please enter only one
letter.")

            return

        if ch in self.wrong_guesses or ch in self.masked_word:
```

```
            messagebox.showinfo("Input Error", "You guessed it
already! It's wrong.")

            return

        if ch in self.current_word:

            while ch in self.current_word:
                index = self.current_word.index(ch)

                self.masked_word[index] = ch

                self.current_word = self.current_word[:index] + '*' +
self.current_word[index + 1:]

                self.success += 1

            self.draw_word()

        else:

            self.wrong_guesses += ch

            self.attempts -= 1
```

```
            self.score -= 10

            self.wrong_label.config(text=f"Previous wrong guesses: {self.wrong_guesses}")

        self.attempts_label.config(text=f"Attempts left: {self.attempts}")
        self.score_label.config(text=f"Score: {self.score}")

        self.guess_entry.delete(0, tk.END)
        if self.success == len(self.acopy):

            messagebox.showinfo("Congratulations", "Congrats! The word is " + self.acopy)

            self.start_game()

        elif self.attempts == 0:

            messagebox.showinfo("Game Over", "Better luck next time. The word was " + self.acopy)

            self.start_game()

    def exit_program(self):
```

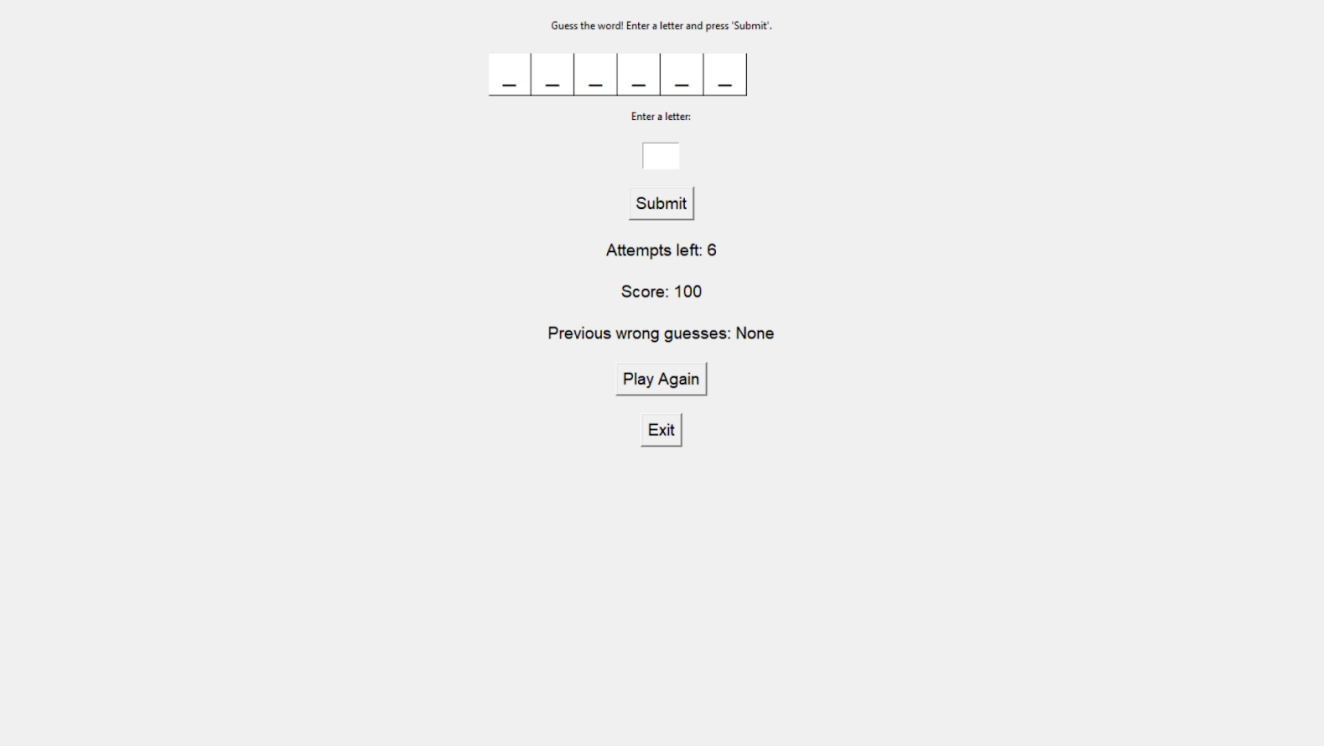
```
        confirm_exit = messagebox.askquestion("Exit", "Are you sure you
want to exit?")


        if confirm_exit == "yes":
            self.master.quit()


            self.master.destroy()
root = tk.Tk()


game = GuessingGame(root)


root.mainloop()
```

CHAPTER-5

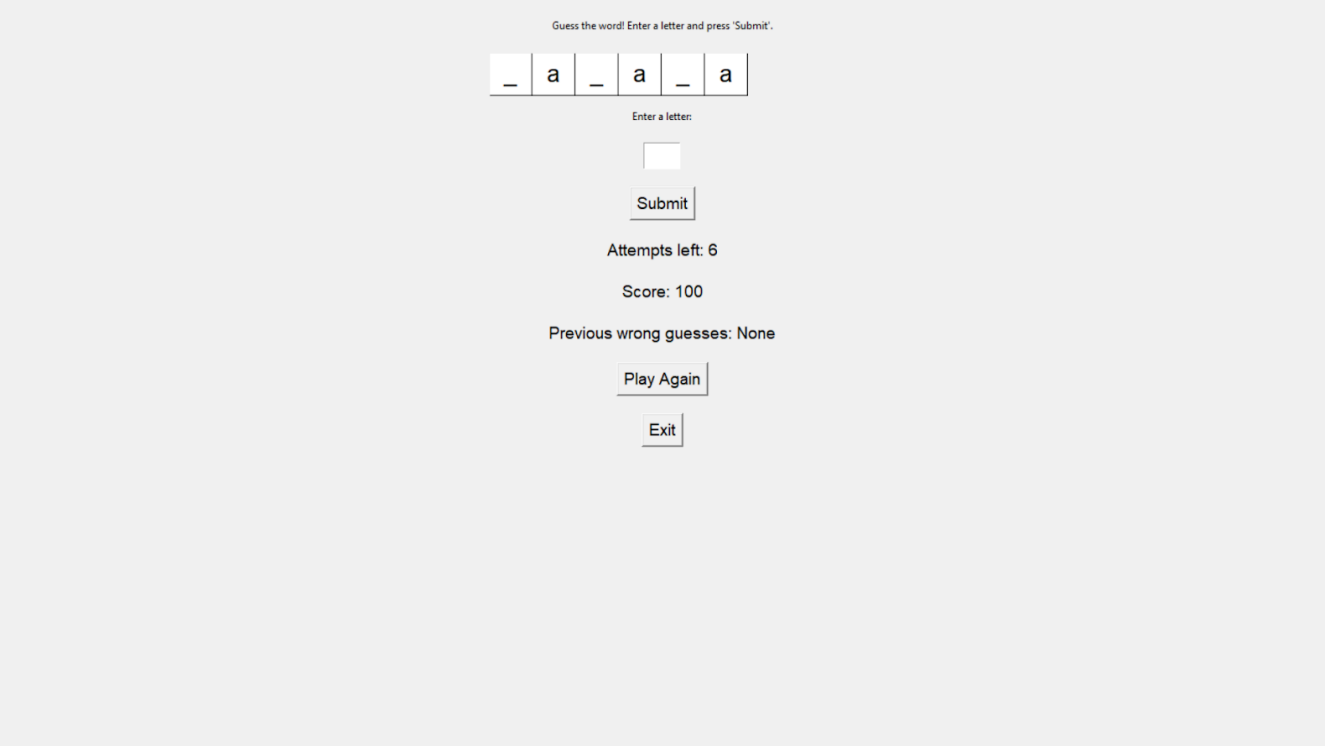
RESULTS AND DISCUSSION

5.1 USER DOCUMENTATION:

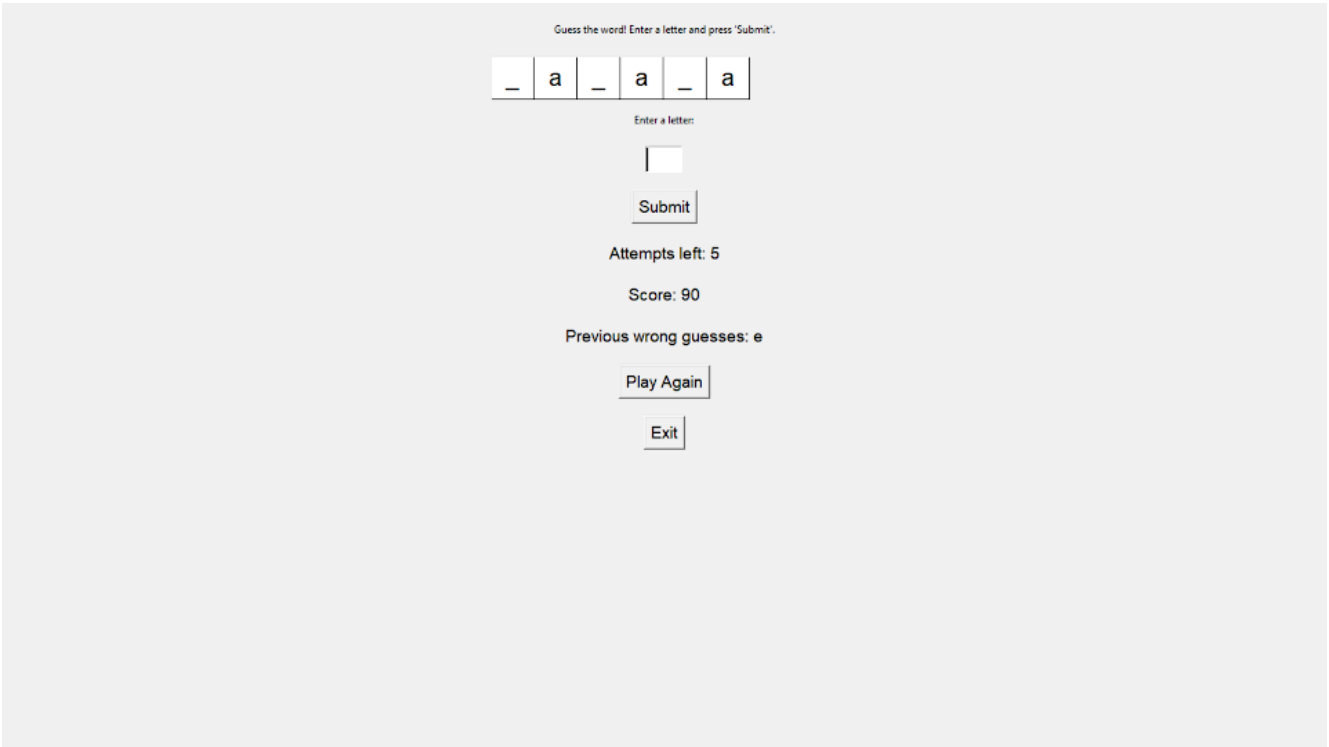
GAME MODULE:



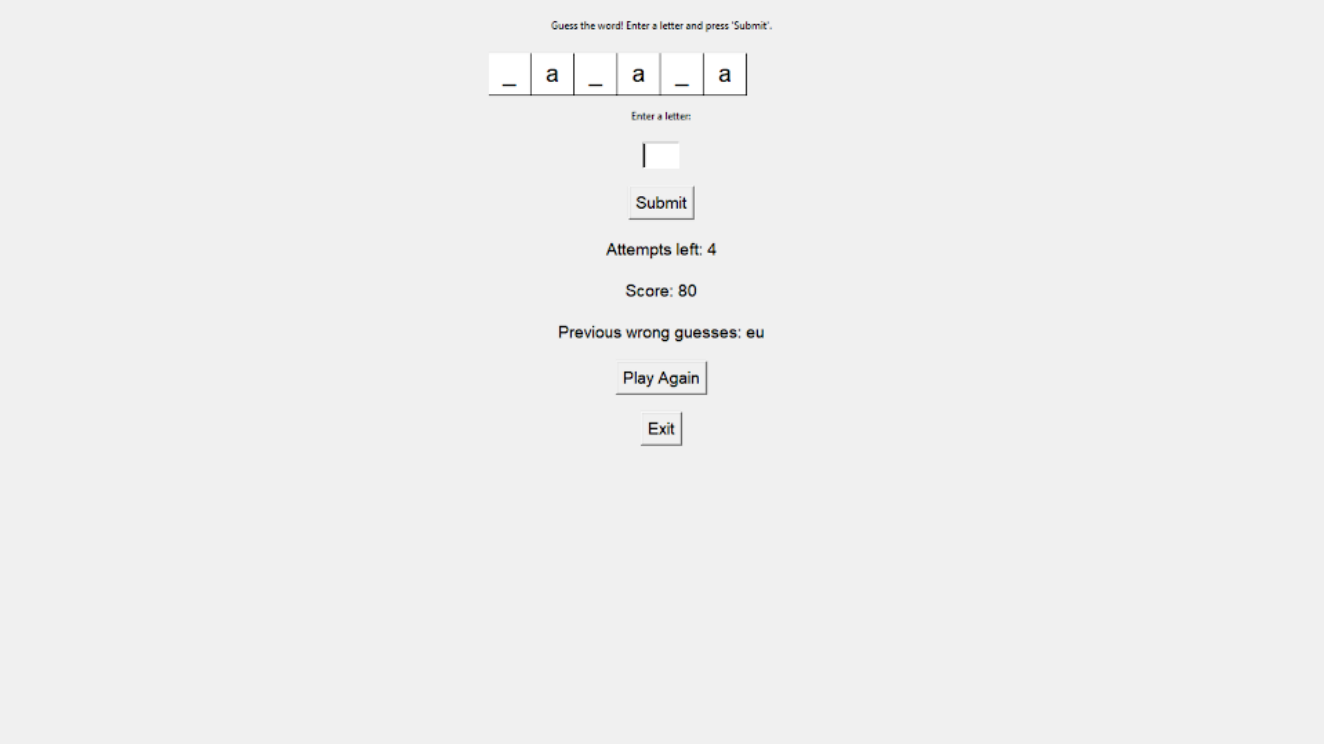
GUESSING WORD MODULE:



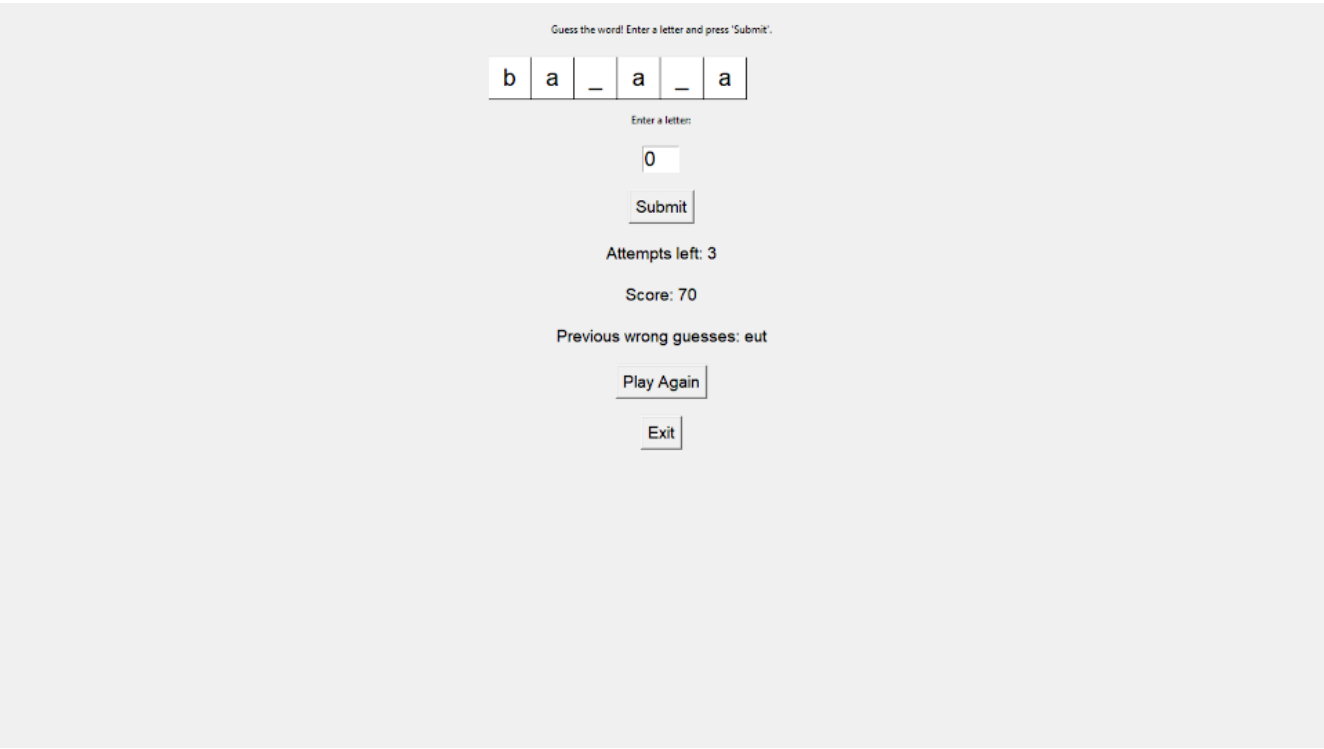
GUESSING WORD MODULE:



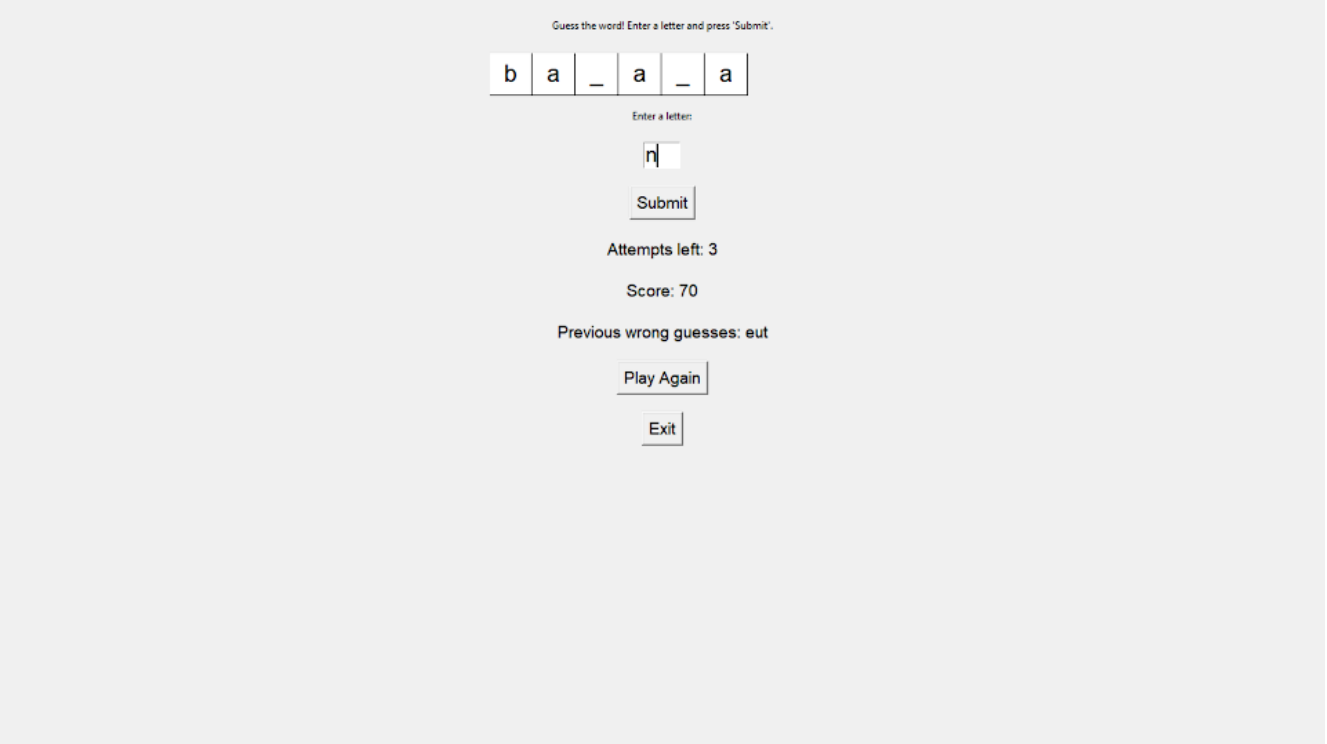
GUESSING WORD MODULE:



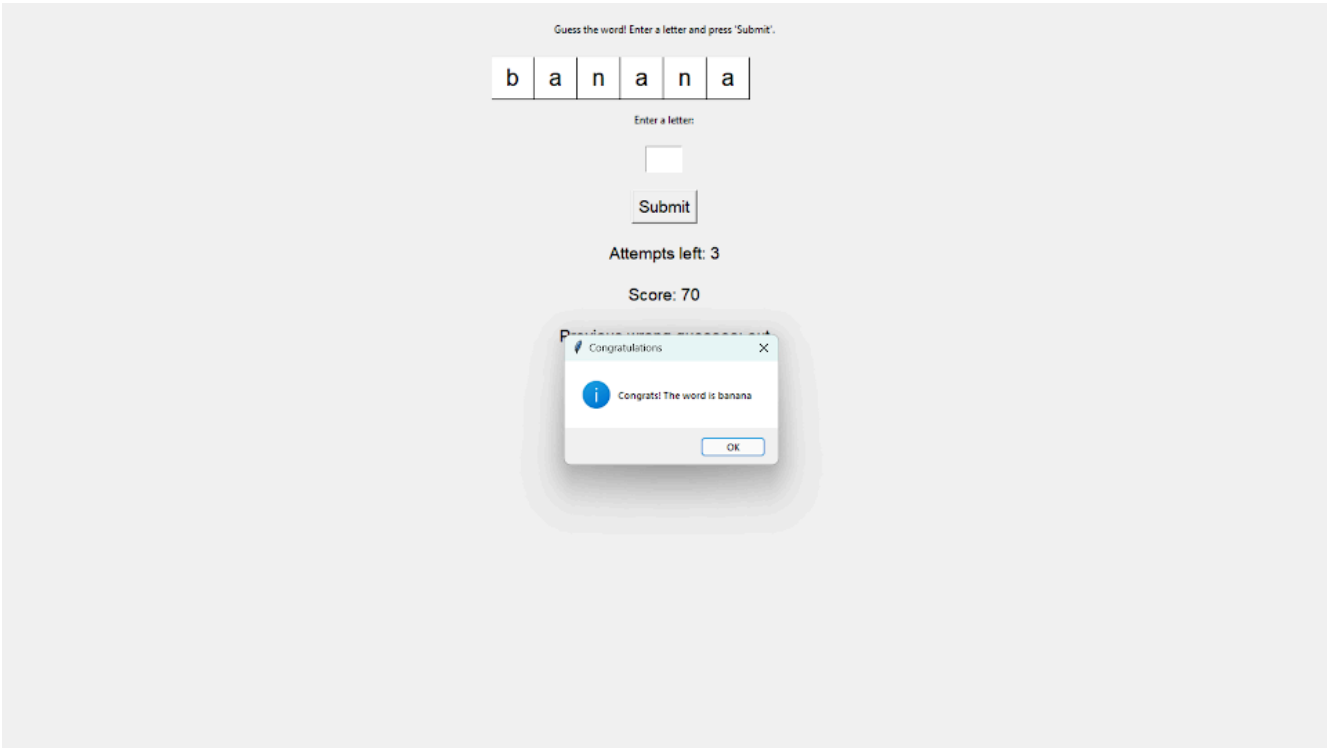
GUESSING WORD MODULE:



GUESSING WORD MODULE:



FINAL STATUS MODULE:



CHAPTER-6

6.1 CONCLUSION:

The Guessing Game application offers an engaging and educational experience through its word-guessing gameplay, seamlessly integrating Python's Tkinter library for the user interface and MySQL for backend word storage. The application demonstrates how a simple yet effective game can be created using these technologies, emphasizing user-friendly design and efficient game mechanics. By adhering to the specified hardware and software requirements, the game ensures smooth operation and responsiveness. The setup process, involving Python and MySQL installation, library management, and database configuration, is straightforward, enabling users to quickly get the game up and running. This project not only serves as an enjoyable pastime but also provides valuable insights into GUI development, database connectivity, and Python programming.

CHAPTER-7

7.1 REFERENCES:

References

1. Python Official Website:

- For downloading Python and accessing its documentation.
- [Python.org](https://www.python.org/)

2. Tkinter Documentation:

- For information on Python's standard GUI toolkit used in the application.

- [Tkinter Documentation](https://docs.python.org/3/library/tkinter.html)

3. MySQL Official Website:

- For downloading MySQL Server and MySQL Workbench.
- [MySQL.com](https://dev.mysql.com/)

4. MySQL Connector/Python:

- For connecting Python applications to MySQL databases.

- [MySQL Connector/Python Documentation](https://dev.mysql.com/doc/connector-python/en/)

5. Python Package Index (PyPI):

- For installing `mysql-connector-python` using pip.

- [PyPI:

mysql-connector-python](https://pypi.org/project/mysql-connector
python/)

6. W3Schools:

- For tutorials on Python, MySQL, and basic SQL queries.
- [W3Schools Python](https://www.w3schools.com/python/)
- [W3Schools SQL](https://www.w3schools.com/sql/)

7. GeeksforGeeks:

- For examples and explanations on using Tkinter and MySQL with Python.

- [GeeksforGeeks

Tkinter](https://www.geeksforgeeks.org/python-gui-tkinter/)

- [GeeksforGeeks MySQL with

Python](https://www.geeksforgeeks.org/mysql-connector-python/)

8. Stack Overflow:

- For troubleshooting and community support on specific coding issues.
- [Stack Overflow](https://stackoverflow.com/)

These references provide the necessary resources for understanding and extending the functionality of the Guessing Game application, ensuring a comprehensive grasp of the tools and technologies involved.