A
REAL TIME RESEARCH PROJECT REPORT
ON

MORSE CODE TRANSLATOR

Submitted in partial fulfilment of the requirements for award of the degree of

BACHELOR OF TECHNOLOGY

In

ARTIFICIAL INTELLIGENCE & DATA SCIENCE

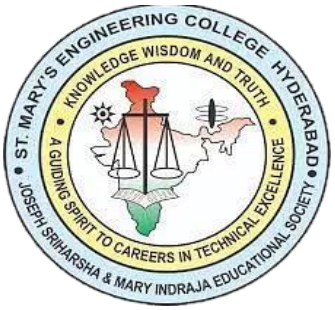
BY

CHADA SIDDHARTHA REDDY	**23BH1A7217**
CHERUKU SAISHIVARAM	**23BH1A7218**
G. SIDDHARTH REDDY	**23BH1A7233**
J. UDAY	**23BH1A7240**
RAJABABU KUMAR SINGH	**23BH1A7280**

Under the guidance of

Mr. M. SANDEEP KUMAR M.Tech

Assistant Professor



DEPARTMENT OF ARTIFICIAL INTELLIGENCE & DATA SCIENCE

St. Mary's Engineering College

(Approved by AICTE, NEW DELHI. & Affiliated to JNTU-HYDERABAD, Accredited by NAAC)

Deshmukhi (V), Pochampally (M), Yadadri Bhuvanagiri (D), Telangana-50824

[2024-25]

CERTIFICATE

This is to certify that project report entitled **"MORSE CODE TRANSLATOR"** is bonafide work carried out in the II/II semester by **C.SIDDHARTHA REDDY (23BH1A7217), C. SAISHIVARAM (23BH1A7218), G. SIDDHARTH REDDY (23BH1A7233), J.UDAY (23BH1A7240), RAJABABU KUMAR SINGH (23BH1A7280)** in partial fulfilment award of Bachelor of Technology in **ARTIFICIAL INTELLIGENCE & DATA SCIENCE** from **St. Mary's Engineering college** during the academic year 2023-27.

INTERNAL GUIDE

Mr. M. SANDEEP KUMAR M.Tech

Dept. of AI&DS

HEAD OF THE DEPARTMENT

Dr. B. SRISAILAM M.Tech, (Ph.D)

Dept. of AI&DS

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of this project would be incomplete without the mention of the people who made it possible. We consider it as a privilege to express our gratitude and respect to all those who guided us in the completion of the project.

We are thankful to our internal guide Mr. M. SANDEEP KUMAR, MTech in **Artificial Intelligence & Data Science, St. Mary's Engineering College** for having been of a source encouragement and for insisting vigour to do this project work

We are obliged to **Dr. B. Srisailam , Head of the Department of Artificial Intelligence & Data Science, St. Mary's Engineering College** for his guidance and suggestion throughout project work.

We take this opportunity to express a deep sense of gratitude to **Dr. T.N. Srinivas Rao, Principal of St. Mary's Engineering College** for allowing us to do this project and for this affectionate encouragement in presenting this project work.

We convey our sincere thanks to **Sri Rev. K.V.K RAO, Chairman** of **St. Mary's Engineering College** for giving us a learning environment to grow ourselves personally as well as professionally.

We would like to express our thanks to all staff members who have helped us directly and indirectly in accomplishing this project work. We also extended our sincere thanks to our parents and friends for their moral support throughout the project work. Above all we thank god almighty for his manifold mercies in carrying out this project work successfully.

C. SIDDHARTHA REDDY -23BH1A7217

CH. SAISHIVARAM -23BH1A7218

G. SIDDHARTH REDDY -23BH1A7233

J. UDAY -23BH1A7240

RAJABABU KUMAR SINGH -23BH1A7280

DECLARATION

This is to certify that the work report in thesis titled,"**MORSE CODE TRANSLATOR**", submitted to **the Department of Artificial Intelligence & Data Science, St. Mary's Engineering College** in fulfilment of degree for the award of Bachelor of Technology, is a bonafide work done by us. No part of the thesis is copied from books, journals or the internet and wherever the portion is taken, the same has been duly referred to in the text. The reported results are based on the project work entirely done by us and not copied from any other sources. Also we declare that the matter embedded in this thesis has not been submitted by us in full or partially there for the award of any degree of any other institution or university previously.

C.SIDDHARTHA REDDY -23BH1A7217

CH. SAISHIVARAM -23BH1A7218

G. SIDDHARTH REDDY -23BH1A7233

J. UDAY -23BH1A7240

RAJABABU KUMAR SINGH -23BH1A7280

CONTENTS

LIST OF FIGURES

LIST OF PLATES

ABSTRACT

This project focuses on developing a simple and effective Morse Code Translator using the Python programming language. Morse code is a method of encoding textual information through sequences of dots and dashes, traditionally used in telegraphy and later adopted in military and emergency communication. The aim of this project is to make the translation between English text and Morse code easy and accessible for a wide range of users, including students, hobbyists, and anyone with an interest in classic communication methods. The translator operates in two directions: it can convert standard English sentences into Morse code, and it can also decode Morse code back into readable English. This dual functionality provides users with the flexibility to both practice sending Morse messages and interpreting them. It's a fully offline tool, meaning it does not require an internet connection to function, making it suitable for use in classrooms, remote learning environments, or areas with limited internet access. Designed to run on any basic computer system, the translator is simple and lightweight, leveraging Python's straightforward syntax and built-in features. It uses dictionaries to map characters to Morse symbols and incorporates input validation and error handling to guide users through the translation process without confusion. In addition to being educational, this project also encourages users to engage with programming concepts such as data structures, conditionals, and loops. It can also be extended with more advanced features like sound output or a graphical interface, making it a valuable and expandable tool for learning both Morse code and Python programming.

CHAPTER 1 - INTRODUCTION

1.1 Introduction

Morse code was one of the earliest methods used to send messages across long distances using electric signals. Each character in the English alphabet, as well as numbers and some punctuation marks, is represented by a unique combination of dots (.) and dashes (-). For instance, the letter "A" is written as ".-" in Morse code. This system allowed for efficient communication, especially in the days before telephones and the internet, and is still taught and used in certain emergency and radio communication contexts today.

This project is designed to help users easily translate English sentences into Morse code and convert Morse code back into readable English. It has been developed using Python and Streamlit, making it not only functional but also visually interactive and easy to use via a web interface. Streamlit allows the application to run in the browser, where users can input their text, view the translated output, and interact with the tool in real time without any prior technical knowledge.

This translator is particularly useful for students, educators, and enthusiasts who want to explore how this historical communication system works. By combining the simplicity of Python with the power of Streamlit, this project serves as both an educational tool and a fun way to learn Morse code.

1.2 Features

The Morse Code Translator, built using Python and Streamlit, offers a range of powerful and user-friendly features designed to make learning and using Morse code both effective and enjoyable. One of its key strengths is the two-way translation functionality, which allows users to seamlessly convert English text into Morse code and decode Morse code back into English. This bidirectional capability makes it ideal for both learners and those looking to practice real-time translations. The tool is fast and highly accurate, using a well-structured character mapping system that ensures reliable conversion of letters, numbers, and basic punctuation.

To enhance user understanding, the translator also includes an optional beep sound feature that audibly represents the Morse code—short beeps for dots and longer beeps for

dashes—providing a more immersive and intuitive way to grasp the rhythm and structure of Morse communication. The application's interface, built with Streamlit, is clean, responsive, and accessible via any web browser, making it easy for users of all technical backgrounds to navigate and use. A built-in user manual is provided within the app, offering clear instructions and usage examples, ensuring that even first-time users can operate the translator smoothly without confusion. Overall, the project combines simplicity, speed, and interactivity to deliver a complete Morse code learning experience.

The tool is fast and highly accurate, using a well-structured character mapping system that ensures reliable conversion of letters, numbers, and basic punctuation.

CHAPTER 2

SYSTEM ANALYSIS

2.1 EXISTING SYSTEM

Many existing Morse code translator tools are limited in functionality and do not fully support an immersive or educational experience. These are the following drawbacks that exist in the existing system.

Lacks Audio Playback:

Most existing tools do not support sound output for Morse code. This prevents users from learning the actual rhythm and tone of Morse communication, which is essential for real-world understanding.

No Educational Support:

These tools are not designed with learners in mind. They often lack helpful features such as tooltips, real-time hints, examples, or interactive guidance to support beginners during usage.

No Customization Options:

Users are usually unable to adjust Morse code settings like speed of playback, tone frequency, or spacing between characters. This limits the learning flexibility and personalization for users with different needs.

One-Way Translation Only:

Many tools offer only one-directional conversion—either from English to Morse code or Morse to English. This restricts comprehensive learning and forces users to switch between different tools for full functionality.

2.2 PROPOSED SYSTEM

These are the following key features included in the proposed system.

Audio Playback Enabled:

One of the standout features of this tool is its built-in audio playback for Morse code. Users can hear distinct beep sounds representing dots (short beeps) and dashes (long beeps), closely mimicking traditional Morse transmission. This auditory feedback significantly improves learning by helping users understand the rhythm and tone of Morse code communication.

Educational Features Integrated:

The translator is designed with learners in mind. It includes helpful tooltips, usage examples, and inline explanations that guide users through the translation process. These elements make it easier for beginners to understand how Morse code works and how to correctly use the tool, creating a more engaging and supportive learning environment.

User Customization Options:

To make the learning experience more flexible and user-friendly, the tool allows users to customize various aspects of Morse code generation. Users can adjust the speed of playback, modify character spacing, and choose whether to enable or disable the sound. This personalization supports different learning preferences and needs.

Bi-directional Translation:

Unlike many existing tools, this application supports both English to Morse and Morse to English translations in a single interface. Users can input either type of text and switch between modes effortlessly, making it a comprehensive solution for anyone learning or using Morse code.

CHAPTER 3
SYSTEM SPECIFICATIONS

3.1 SOFTWARE DESCRIPTION

- Operating system - Windows 8/10/11
- Programming language - Python
- Software required - Streamlit, Winsound
- Version Control System - Git/Github
- Development Tools - Vscode

3.1.1 VScode

Visual Studio Code (VS Code) is a powerful, lightweight, and open-source code editor developed by Microsoft. It is widely popular among developers for its speed, flexibility, and extensive support for various programming languages such as Python, JavaScript, C++, Java, and more. Available on Windows, macOS, and Linux, VS Code provides a seamless coding experience through a user-friendly interface and robust functionality.

One of the key strengths of VS Code is its extensibility. Users can enhance the editor by installing thousands of extensions from the built-in marketplace—ranging from language packs, debuggers, code formatters, linters, to AI-powered tools like GitHub Copilot. Features such as IntelliSense offer smart code completions based on variable types, function definitions, and imported modules, which boosts coding efficiency.

VS Code also includes an integrated terminal, built-in Git support, and a debugging tool, making it a complete development environment in one compact package. It supports customization through themes, keyboard shortcuts, and workspace settings to match individual preferences. Whether you're working on web development, data science, or software engineering, VS Code adapts to different workflows and project needs. Its performance, community support, and versatility have made it a favorite among both beginner and professional developers.

3.1.2 Version Control System

A Version Control System (VCS) is a tool used to track changes to files and collaborate on projects. It allows multiple people to work on the same codebase without interfering with each other's work. VCS is essential for software development, as it enables developers to manage and track modifications over time.

There are two types of version control systems: Centralized Version Control Systems (CVCS) and Distributed Version Control Systems (DVCS). In a CVCS, the version history is stored in a central server, and each developer has a local copy of the files. Examples include Subversion (SVN). In contrast, a DVCS allows each developer to have their own full copy of the entire repository, including its history. Git, the most popular VCS, is a DVCS.

With version control, developers can revert files to previous versions, compare changes over time,

and track who made specific modifications. It also facilitates branching, where developers can create separate versions of a project, work on them independently, and later merge the changes back together.

Popular platforms like GitHub, GitLab, and Bitbucket integrate with Git, offering additional features like collaboration tools, issue tracking, and code review. VCS is indispensable for team-based development, ensuring smooth workflows and minimizing conflicts.

3.1.3 Development Tools

Code Editor: Visual Studio Code (VS code)

Alternatives: Atom or Sublime Text

Browser (For testing and debugging)

- Google Chrome
- Mozilla Firefox
- Microsoft Edge

3.2 HARDWARE REQUIRED

1. Speed- 1.1 Ghz
2. RAM – 256 MB
3. Keyboard - Standard Windows Keyboard
4. Mouse - Two or Three Button Mouse
5. Monitor- SVGA

Speed-1.1 Ghz

What Does GHz Mean in a Computer Processor?

One of the most frequently touted measures of a processor is given chip's speed in gigahertz. Processors with higher GHz ratings can, theoretically, do more in a given unit of time than processors with lower GHz ratings. However, the processor's speed rating is just one of many factors that impact how fast it actually processes data. Given that some specialized applications can be very computationally demanding, choosing the fastest computer is more important than buying a machine with the highest clock speed.

System Clocks

Processors work according to a clock that beats a set number of times per second, usually measured in gigahertz. For instance, a 3.1-GHz processor has a clock that beats 3.1 billion times per second , and a 1.1 Ghz processor has a clock that beats 1.1 billion times per second. Each clock beat represents an opportunity for the processor to manipulate a number of bits equivalent to its capacity - 64-bit processors can work on 64 bits at a time, while 32-bit processors work on 32 bits at a time.

RAM – 256 MB

RAM (Random Access Memory) is the hardware in a computing device where the operating system (OS), application programs and data in current use are kept so they can be quickly reached by the device's processor. RAM is the main memory in a computer. It is much faster to read from and write to than other kinds of storage, such as a hard disk drive (HDD), solid-state drive (SSD) or optical drive. Random Access Memory is volatile. That means data is retained in RAM as long as the computer is on, but it is lost when the computer is turned off. When the computer is rebooted, the OS and other files are reloaded into RAM,

usually from an HDD or SSD **How much RAM do you need?**

The amount of RAM needed all depends on what the user is doing. When video editing, for example, it's recommended that a system have at least 16 GB RAM, though more is desirable. For photo editing using Photoshop, Adobe recommends a system have at least 3GB of RAM to run photo editing software.

3.2.1 Key Board - Standard Windows Keyboard

A keyboard is one of the primary input devices that allows users to input text into a computer or any other electronic machinery. It is a peripheral device that is the most basic way for the user to communicate with a computer. It consists of multiple buttons, which create numbers, symbols, and letters, and special keys like the Windows and Alt key, including performing other functions. The design of the keyboard comes from the typewriter keyboards, and numbers and letters are arranged on the keyboard in that way, which helps to type quickly.

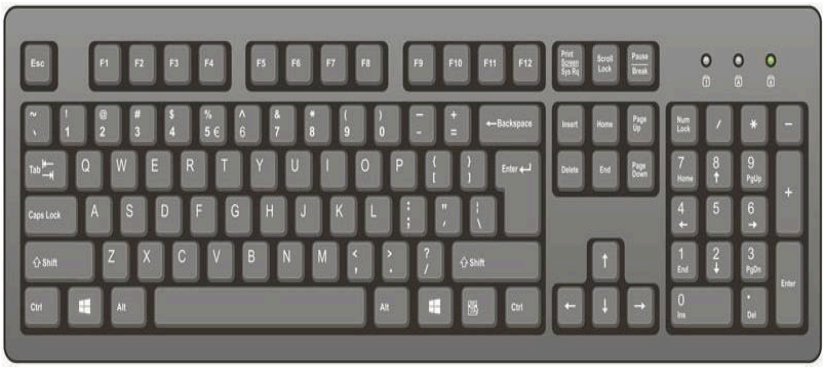


Figure 3.3.3:*Standard windows keyboard*

The above keyboard design is called Qwerty design because of its first six letters across in the upper-left-hand corner of the keyboard. Although the keyboard design is derived from the typewrites, nowadays, it also includes many other keys as well as Alt/Option, Control, and Windows key can be used as shortcuts to perform the particular operation by combination with other keys. For example, if you press Control + S while working on a document in Microsoft word, it will save the document you are working on. Furthermore, most of the keyboards have function keys (F1 to F12 or F16) at the top of the keyboard and arranged arrow keys in the downside used to perform numerous functions.

3.2.2 Mouse -Two or three Button Mouse

A mouse is a small device that a computer user pushes across a desk surface in order to point to a place on a display screen and to select one or more actions to take from that position. The mouse first became a widely-used computer tool when Apple Computer made it a standard part of the Apple Macintosh.

3.2.3 Monitor – SVGA

Super video graphics array (Super VGA or SVGA) is a high-resolution standard used to channel video data to a compatible visual output device - usually a computer monitor. This is actually a broad umbrella term for other computer display standards.

Originally, it was just an extension to the VGA standard, which was a purely IBM defined standard also known as ultra video graphics array (UVGA).

CHAPTER 4

SYSTEM STUDY

4.1 Economical Feasibility

The Morse Code Translator Application developed using Python and Streamlit is economically feasible due to its low cost nature. Development costs are minimal as Streamlit, Winsound, Python are open-source tools. The operational costs are also low because it doesnt require any heavy infrastructure. The app can be maintained with minimal resources, making it cost-effective in the long run.

4.2 Technical Feasibility

The technical feasibility of this project is high, as the tools and technologies used are readily available and appropriate for the task. Python and Streamlit are powerful and flexible tools that are well-suited to creating a simple and effective web-based Morse Code Translator. Streamlit's easy-to-use framework allows for rapid development of interactive web apps without requiring extensive front-end development. Since the app is standalone, there is no complex system integration required. Streamlit will handle the UI, and Python will manage the backend translation logic, making it a simple and efficient solution.

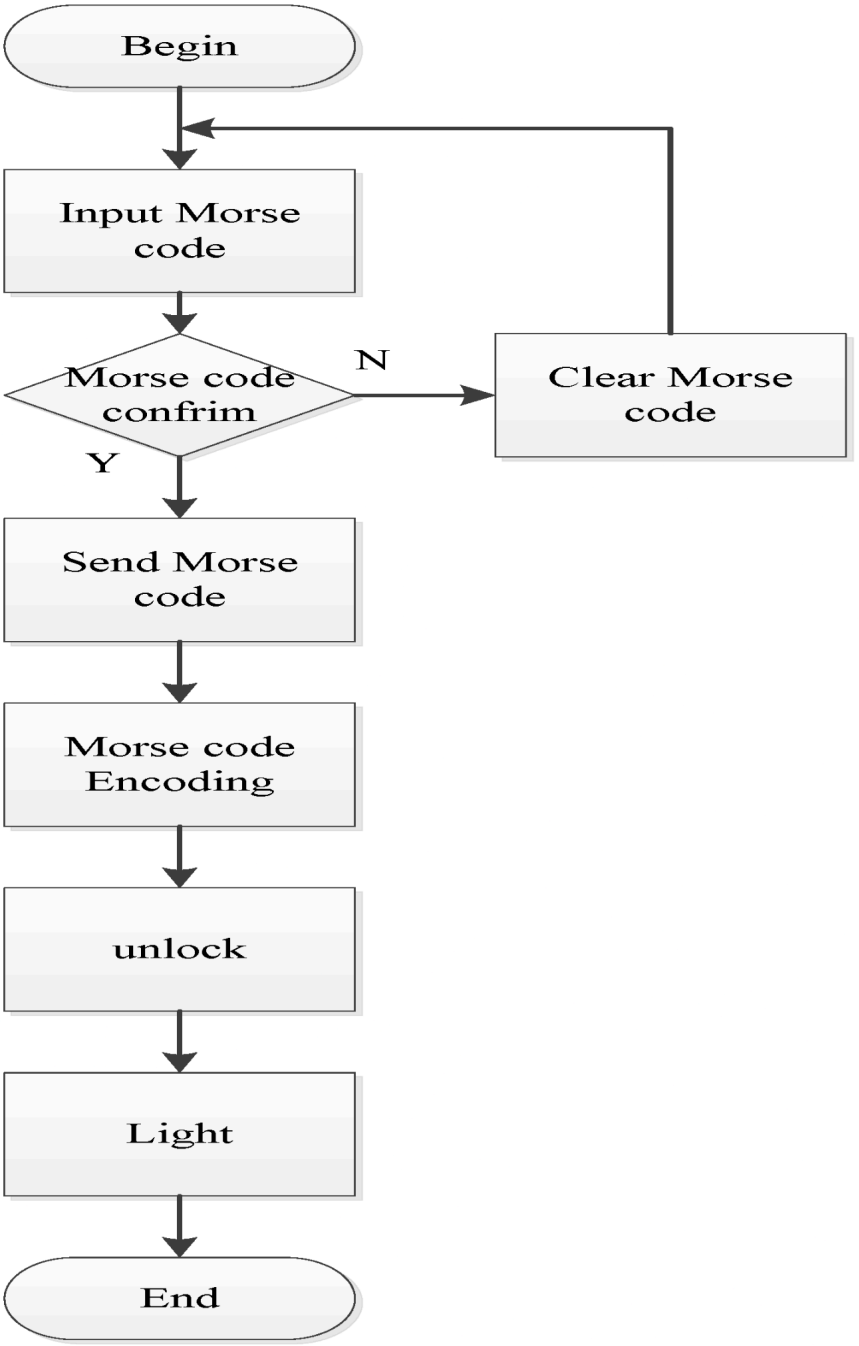
4.3 Social Feasibility

The app is likely to be appreciated by those interested in learning Morse code, people studying history or communication methods, and individuals who enjoy educational tools. It could also be useful for emergency preparedness, where Morse code might still be relevant. Morse code is a neutral system with no significant cultural barriers, making the app suitable for a global audience. You could consider adding support for multiple languages in the future to enhance its accessibility

CHAPTER-5

SYSTEM ARCHITECTURE

SYSTEM ARCHITECTURE



CHAPTER-6

DESIGN AND IMPLEMENTATION

6.1 DESIGN PROCESS

The design procedure of the Morse Code Translator project involved a structured approach to ensure both functionality and user-friendliness while leveraging Python and Streamlit. The process began with defining the problem: creating an interactive web application capable of translating between English text and Morse code. After establishing the core requirements, both functional (text input, translation direction selection, and output display) and non-functional (fast response time, lightweight design, and ease of use) aspects were considered. Python was chosen for its simplicity in handling the core logic, utilizing a dictionary-based approach to map English characters to their corresponding Morse code values. Two key functions—one for encoding text to Morse code and another for decoding Morse code back to text—were developed while accounting for edge cases such as handling unsupported characters and spacing issues. The front-end was built using Streamlit to provide a clean, intuitive interface with components like text areas for input, selection options for translation direction, and buttons to trigger translation and display results. Following development, comprehensive testing and debugging were conducted to ensure accurate translations and a smooth user experience. Finally, the app was deployed using Streamlit Sharing for public access. Potential future enhancements include adding sound output for Morse code, providing a learning mode with Morse code charts, and implementing features like saving translation history for improved user engagement.

6.2 IMPLEMENTATION

1. Data Structures

- Created a **dictionary (`morse_dict`)** to map English characters (A-Z, 0-9, space) to Morse code.
- Generated a **reverse dictionary** to decode Morse code back to English characters.

2. Encoding Function (Text → Morse)

- Converted input text to uppercase.

- Iterated through each character and mapped it to Morse code using the dictionary.
- Joined the Morse code with spaces to separate each letter.

3. Decoding Function (Morse → Text)

- Split Morse code by spaces to identify individual characters.
- Used the reverse dictionary to map Morse code back to English letters.
- Joined the decoded letters to form readable text.

4. Streamlit UI Design

- Used `st.title()` and `st.markdown()` to add headings and descriptions.
- Added `st.radio()` to let users choose between encoding or decoding.
- Used `st.text_area()` for user input.
- Used `st.button()` to trigger the translation.
- Displayed results using `st.code()` for a clear, formatted output.

5. Testing

- Tested the app with valid and invalid inputs.
- Handled edge cases such as lowercase input, empty input, and unsupported characters.

6. Deployment

- Deployed the app using **Streamlit Community Cloud**, allowing public access via a shareable link.

CHAPTER - 7
SOFTWARE ENVIRONMENT

The Morse Code Translator application was developed in a lightweight and efficient software environment, leveraging open-source tools that support rapid development and easy deployment.

- **Programming Language: Python**
  Python was chosen as the core language due to its simplicity, readability, and strong support for string manipulation and dictionary-based data structures, which are essential for encoding and decoding Morse code.

- **Framework: Streamlit**
  Streamlit was used to build the web-based user interface. It allows quick creation of interactive apps with minimal code and integrates seamlessly with Python scripts. Streamlit handles rendering, layout, and UI components like buttons, text areas, and selection boxes.

- **Development Environment:**
  - **Code Editor:** Visual Studio Code (or any preferred IDE)
  - **Python Version:** 3.x
  - **Required Libraries:**
    - `streamlit` – for UI and frontend interaction
    - No external libraries were needed for translation logic, keeping the app lightweight.

- **Operating System:**
  The project is platform-independent and can be developed and run on Windows, macOS, or Linux, as long as Python and Streamlit are installed.

- **Deployment Platform: Streamlit Community Cloud**
  The app was deployed using Streamlit's free cloud service, which provides instant web hosting for Streamlit apps without the need for complex server setup.

CHAPTER 8
UML DIAGRAMS

8.1 ACTIVITY DIAGRAM

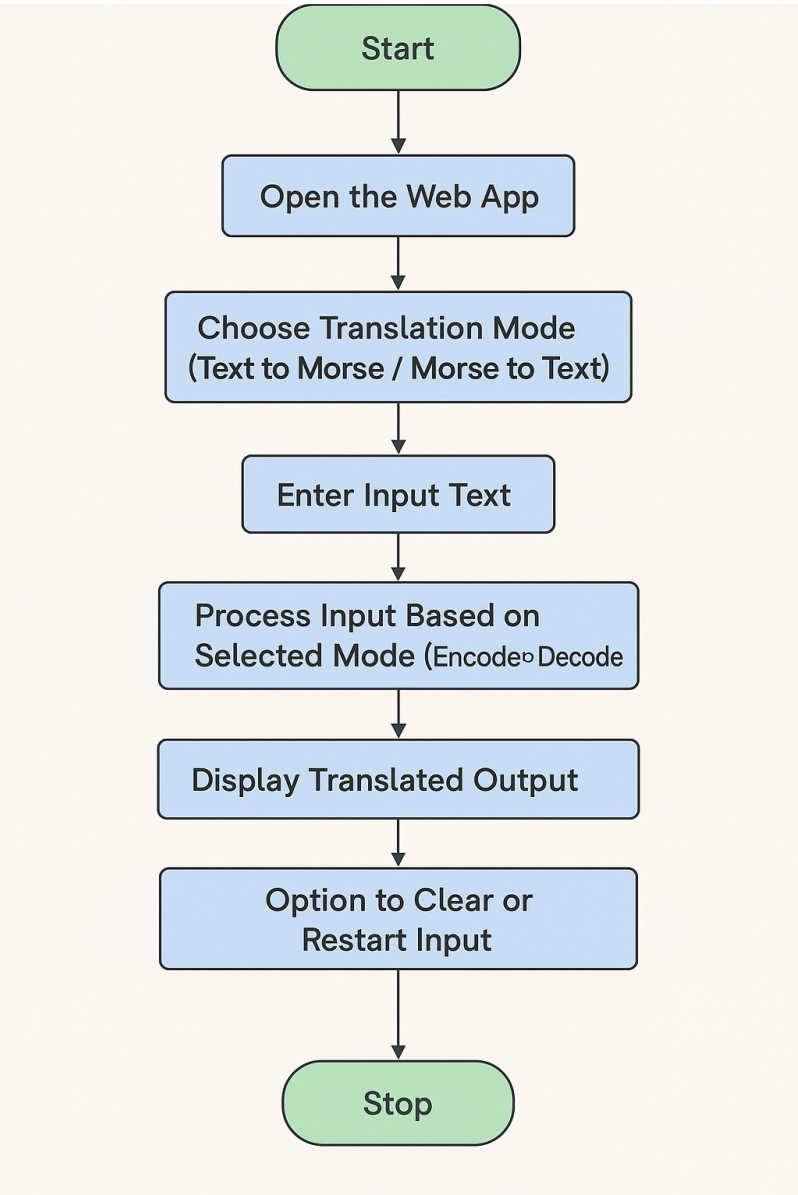


FIGURE 8.1 ACTIVITY DIAGRAM

8.2 CLASS DIAGRAM

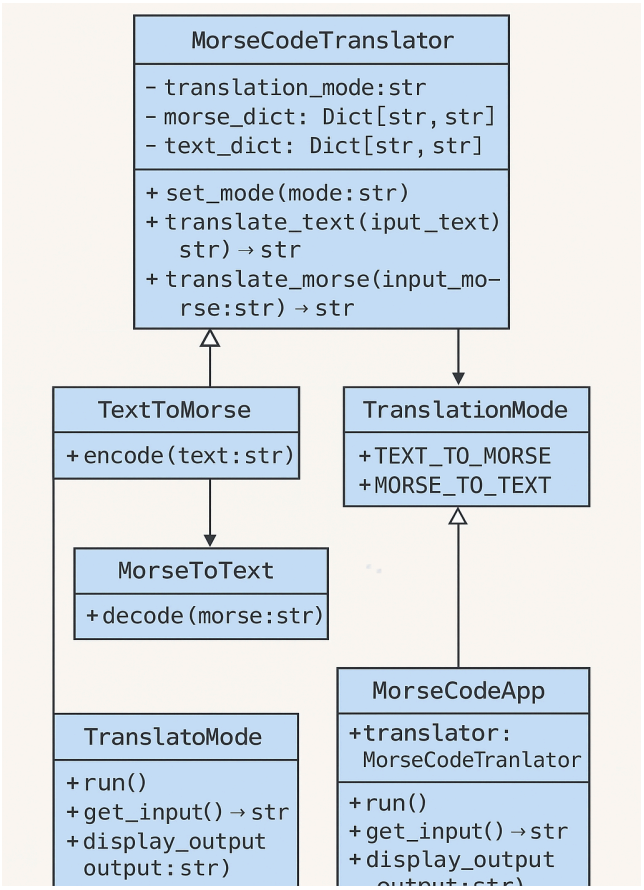


FIGURE 8.2 CLASS DIAGRAM

8.3 SEQUENCE DIAGRAM

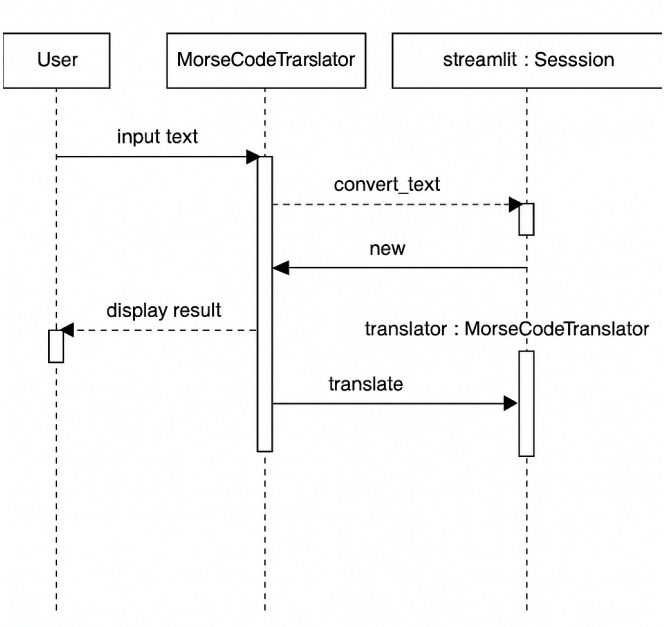


FIGURE 8.3 SEQUENCE DIAGRAM

8.4 USE CASE DIAGRAM

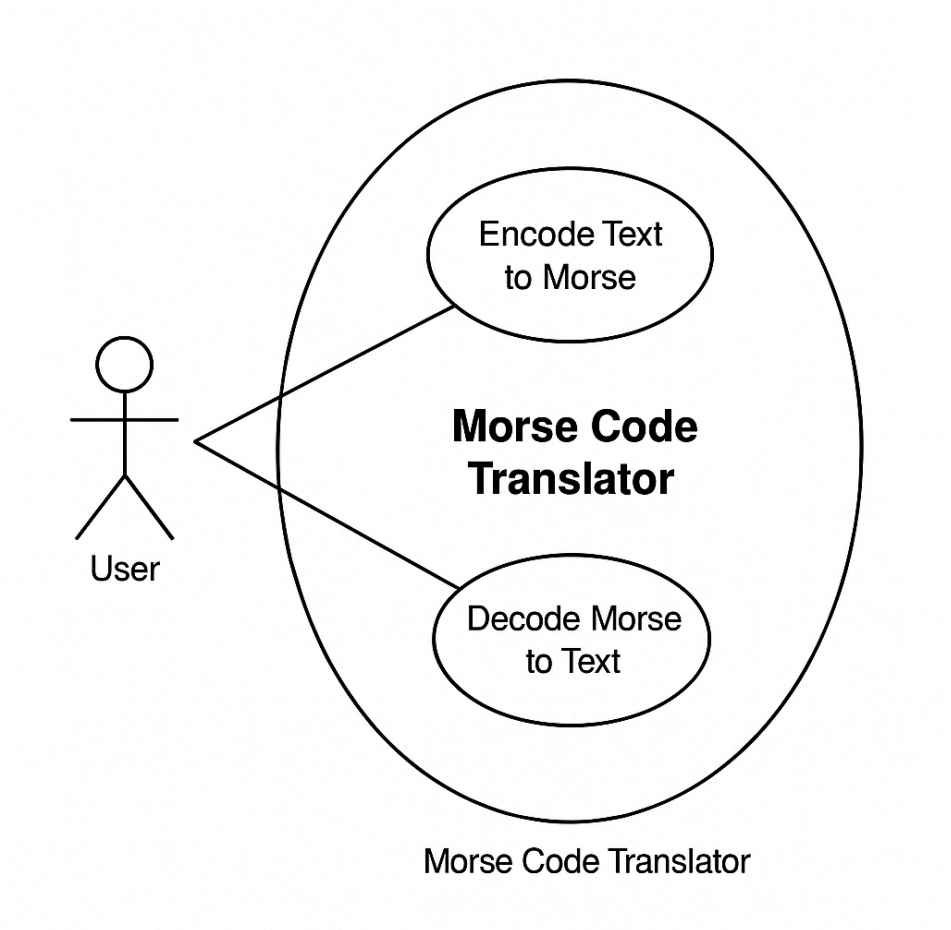


FIGURE 8.4 USE CASE DIAGRAM

CHAPTER 9
SOURCE CODE

morse_code_app.py

```
import streamlit as st
import pandas as pd
import tempfile
import os
from io import StringIO
import base64
import time
import numpy as np
from scipy.io.wavfile import
write

# Set page configuration
st.set_page_config(
    page_title="Morse Code
Translator",
    page_icon="🔊",
    layout="wide",

initial_sidebar_state="expan
ded"
)

# Custom CSS for better UI
st.markdown("""
<style>
    .main {
        background-color:
#f5f7f9;
    }
    .stApp {
        max-width: 1200px;
        margin: 0 auto;
    }
    .block-container {
        padding-top: 2rem;
        padding-bottom: 2rem;
    }
    h1, h2, h3 {
        color: #1e3a8a;
```

```
    }
    .stTextInput, .stTextArea,
.stFileUploader {
        background-color:
white;
        border-radius: 10px;
        padding: 10px;
        box-shadow: 0 2px 4px
rgba(0,0,0,0.05);
    }
    .stButton>button {
        background-color:
#1e3a8a;
        color: white;
        border-radius: 5px;
        padding: 0.5rem 1rem;
        font-weight: bold;
    }
    .output-box {
        background-color:
white;
        border-radius: 10px;
        padding: 20px;
        margin: 10px 0;
        box-shadow: 0 2px 8px
rgba(0,0,0,0.1);
    }
    .info-box {
        background-color:
#e5edff;
        border-left: 5px solid
#1e3a8a;
        padding: 15px;
        border-radius: 5px;
        margin: 10px 0;
    }
    .card {
        background-color:
white;
        border-radius: 10px;
        padding: 20px;
        box-shadow: 0 4px 8px
rgba(0,0,0,0.1);
        margin-bottom: 20px;
    }
```

```
    .translator-container {
        display: flex;
        flex-direction: column;
        gap: 20px;
    }
</style>
""",
unsafe_allow_html=True)

# Define Morse code
dictionary
MORSE_CODE_DICT = {
    'A': '.-', 'B': '-...', 'C': '-.-.',
'D': '-..', 'E': '.',
    'F': '..-.', 'G': '--.', 'H': '....',
'I': '..', 'J': '.---',
    'K': '-.-', 'L': '.-..', 'M': '--',
'N': '-.', 'O': '---',
    'P': '.--.', 'Q': '--.-', 'R': '.-.',
'S': '...', 'T': '-',
    'U': '..-', 'V': '...-', 'W': '.--',
'X': '-..-', 'Y': '-.--',
    'Z': '--..', '1': '.----', '2':
'..---', '3': '...--', '4': '....-',
    '5': '.....', '6': '-....', '7':
'--...', '8': '---..', '9': '----.',
    '0': '-----', ', ': '--..--', '.':
'.-.-.-', '?': '..--..',
    '/': '-..-.', '-': '-....-', '(':
'-.--.', ')': '-.--.-',
    ' ': '/'  # Space is
represented as a forward
slash in Morse code
}

# Reverse lookup dictionary
for decoding
REVERSE_MORSE_COD
E_DICT = {v: k for k, v in
MORSE_CODE_DICT.item
s()}

def text_to_morse(text):
    """Convert text to Morse
code."""
```

```
    text = text.upper()
    morse_code = []

    for char in text:
        if char in
MORSE_CODE_DICT:

morse_code.append(MORS
E_CODE_DICT[char])
        else:
            # For characters not
in the dictionary, we'll just
keep them as is

morse_code.append(char)

    return "
".join(morse_code)

def
morse_to_text(morse_code)
:
    """Convert Morse code to
text."""
    # Split the morse code by
space
    morse_words =
morse_code.split(" / ")
    text_words = []

    for word in morse_words:
        morse_chars =
word.split(" ")
        text_chars = []

        for morse_char in
morse_chars:
            if morse_char in
REVERSE_MORSE_COD
E_DICT:

text_chars.append(REVERS
E_MORSE_CODE_DICT[
morse_char])
            elif morse_char:  # If
```

```
it's not empty

text_chars.append("[?]")  #
For unknown morse code


text_words.append("".join(t
ext_chars))

    return "
".join(text_words)

def
generate_beep(morse_text):
    """Generate beep sounds
for Morse code."""
    # Audio parameters
    sample_rate = 44100  #
Hz
    dot_duration = 0.1  #
seconds
    dash_duration = 0.3  #
seconds
    symbol_pause = 0.1  #
seconds
    letter_pause = 0.3  #
seconds
    word_pause = 0.7  #
seconds

    # Generate a sine wave
for the beep
    def
generate_sine_wave(freq,
duration):
        t = np.linspace(0,
duration, int(sample_rate *
duration), False)
        return np.sin(2 * np.pi
* freq * t)

    # Break morse text into
individual symbols
    audio_data = []
    frequency = 800  # Hz
```

```
    morse_letters =
morse_text.split(' ')

    for i, letter in
enumerate(morse_letters):
        if letter == '/':  # Word
separator
            silence =
np.zeros(int(sample_rate *
word_pause))

audio_data.append(silence)
            continue

        for j, symbol in
enumerate(letter):
            if symbol == '.':  #
Dot
                dot_sound =
generate_sine_wave(freque
ncy, dot_duration)

audio_data.append(dot_sou
nd)
            elif symbol == '-':  #
Dash
                dash_sound =
generate_sine_wave(freque
ncy, dash_duration)

audio_data.append(dash_so
und)

            # Add pause
between symbols within a
letter
            if j < len(letter) - 1:
                symbol_silence =
np.zeros(int(sample_rate *
symbol_pause))

audio_data.append(symbol_
silence)
```

```
        # Add pause between
letters
        if i < len(morse_letters)
- 1 and morse_letters[i+1]
!= '/':
            letter_silence =
np.zeros(int(sample_rate *
letter_pause))

audio_data.append(letter_sil
ence)

    # Concatenate all audio
data
    audio =
np.concatenate(audio_data)

    # Normalize audio (max
amplitude of 0.8 to avoid
clipping)
    audio = audio * 0.8 /
np.max(np.abs(audio)) if
np.max(np.abs(audio)) > 0
else audio

    # Save as WAV file
    with
tempfile.NamedTemporaryF
ile(delete=False,
suffix='.wav') as
temp_audio_file:

write(temp_audio_file.name
, sample_rate,
audio.astype(np.float32))
        return
temp_audio_file.name

def
get_binary_file_downloader
_html(bin_file,
file_label='File'):
    """Generate HTML code
for file download link."""
    with open(bin_file, 'rb')
```

```
as f:
        data = f.read()
    b64 =
base64.b64encode(data).dec
ode()
    href = f'<a
href="data:audio/wav;base6
4,{b64}"
download="{file_label}.wa
v"
class="download-button">D
ownload {file_label}</a>'
    return href

def main():
    st.markdown("<h1
style='text-align:
center;'>🔊 Morse Code
Translator</h1>",
unsafe_allow_html=True)

    # Information box
    with st.container():
        st.markdown("""
        <div class="info-box">
            <h3>About Morse
Code</h3>
            <p>Morse code is a
method of transmitting text
information as a series of
on-off tones,
            lights, or clicks. It
was developed by Samuel
Morse in the 1830s for use
with the telegraph.</p>
            <p>• Dot (.) = short
signal<br>• Dash (-) = long
signal<br>• Space between
letters = 3 units<br>• Space
between words = 7
units</p>
        </div>
        """,
unsafe_allow_html=True)
```

```
    # Create tabs for different
input methods
    tab1, tab2 = st.tabs(["📝
Text Input", "📄 File
Upload"])

    with tab1:
        st.markdown("<div
class='card'>",
unsafe_allow_html=True)

        # Direction selection
        direction = st.radio(
            "Select Translation
Direction:",
            ["Text to Morse",
"Morse to Text"],
            horizontal=True
        )

        # Input text area
        if direction == "Text to
Morse":
            input_text =
st.text_area("Enter Text to
Convert to Morse Code:",
height=150)
            placeholder = "Hello
World!"
        else:
            input_text =
st.text_area("Enter Morse
Code to Convert to Text:",
height=150)
            placeholder = ".... .
.-.. .-.. --- / .-- --- .-. .-.. -..
-.-.--"

        # Add example button
        if st.button("Load
Example"):
            input_text =
placeholder

st.session_state.input_text =
```

```
input_text

st.experimental_rerun()

        # Process button
        if
st.button("Translate"):
            if input_text:
                if direction ==
"Text to Morse":
                    morse_result =
text_to_morse(input_text)


st.markdown("<div
class='output-box'>",
unsafe_allow_html=True)

st.markdown("<h3>Morse
Code Output:</h3>",
unsafe_allow_html=True)

st.code(morse_result)

                    # Generate
audio for Morse code
                    with
st.spinner("Generating
audio..."):

audio_file_path =
generate_beep(morse_result
)

                    # Display audio
player

st.audio(audio_file_path,
format='audio/wav')

                    # Provide
download link

st.markdown(get_binary_fil
e_downloader_html(audio_f
```

```
ile_path,
'morse_code_audio'),
unsafe_allow_html=True)

st.markdown("</div>",
unsafe_allow_html=True)

            else:  # Morse to
Text
                text_result =
morse_to_text(input_text)


st.markdown("<div
class='output-box'>",
unsafe_allow_html=True)

st.markdown("<h3>Text
Output:</h3>",
unsafe_allow_html=True)

st.success(text_result)

st.markdown("</div>",
unsafe_allow_html=True)
        else:

st.warning("Please enter
some text to translate.")

    st.markdown("</div>",
unsafe_allow_html=True)

  with tab2:
    st.markdown("<div
class='card'>",
unsafe_allow_html=True)

    # Direction selection
    file_direction =
st.radio(
        "Select Translation
Direction:",
        ["Text to Morse",
"Morse to Text"],
```

```
            key="file_direction",
            horizontal=True
        )

        # File uploader
        uploaded_file =
st.file_uploader("Choose a
text file", type=["txt"])

        if uploaded_file is not
None:
            stringio =
StringIO(uploaded_file.getv
alue().decode("utf-8"))
            file_text =
stringio.read()

            # Show file content
            with
st.expander("File Content"):
                st.text(file_text)

            # Process button
            if
st.button("Translate File"):
                if file_direction
== "Text to Morse":
                    morse_result =
text_to_morse(file_text)


st.markdown("<div
class='output-box'>",
unsafe_allow_html=True)

st.markdown("<h3>Morse
Code Output:</h3>",
unsafe_allow_html=True)

st.code(morse_result)

                # Generate
audio for Morse code
                with
st.spinner("Generating
```

```
audio..."):

audio_file_path =
generate_beep(morse_result
)

                # Display audio
player

st.audio(audio_file_path,
format='audio/wav')

                # Provide
download link

st.markdown(get_binary_fil
e_downloader_html(audio_f
ile_path,
f'morse_{uploaded_file.nam
e}'),
unsafe_allow_html=True)

                # Provide
download link for text
output
                text_file =
tempfile.NamedTemporaryF
ile(delete=False,
suffix='.txt')
                with
open(text_file.name, 'w') as
f:

f.write(morse_result)


st.download_button(

label="Download Morse
Code Text",

data=morse_result,

file_name=f"morse_{upload
ed_file.name}",
```

```
mime="text/plain"
                )

st.markdown("</div>",
unsafe_allow_html=True)

            else:  # Morse to
Text
                text_result =
morse_to_text(file_text)


st.markdown("<div
class='output-box'>",
unsafe_allow_html=True)

st.markdown("<h3>Text
Output:</h3>",
unsafe_allow_html=True)

st.success(text_result)

                # Provide
download link for text
output

st.download_button(

label="Download
Translated Text",

data=text_result,

file_name=f"translated_{upl
oaded_file.name}",

mime="text/plain"
                )

st.markdown("</div>",
unsafe_allow_html=True)

      st.markdown("</div>",
unsafe_allow_html=True)
```

```
    # Morse code reference
chart
    with st.expander("Morse
Code Reference Chart"):
        # Convert dictionary to
DataFrame for display
        morse_df =
pd.DataFrame(list(MORSE
_CODE_DICT.items()),
columns=['Character',
'Morse Code'])

        # Split the dataframe
into columns for better
display
        col1, col2, col3 =
st.columns(3)

        # Calculate split points
        n = len(morse_df)
        split1 = n // 3
        split2 = 2 * n // 3

        with col1:

st.dataframe(morse_df.iloc[:
split1],
use_container_width=True)

        with col2:

st.dataframe(morse_df.iloc[
split1:split2],
use_container_width=True)

        with col3:

st.dataframe(morse_df.iloc[
split2:],
use_container_width=True)

    # Footer
    st.markdown("""
    <div style="text-align:
```

```
center; margin-top: 40px;
padding: 20px; color:
#666;">
        <p>Created with ❤️
using Streamlit</p>
    </div>
    """,
unsafe_allow_html=True)

if __name__ ==
"__main__":
    main()
```

CHAPTER 10
OUTPUT SCREEN

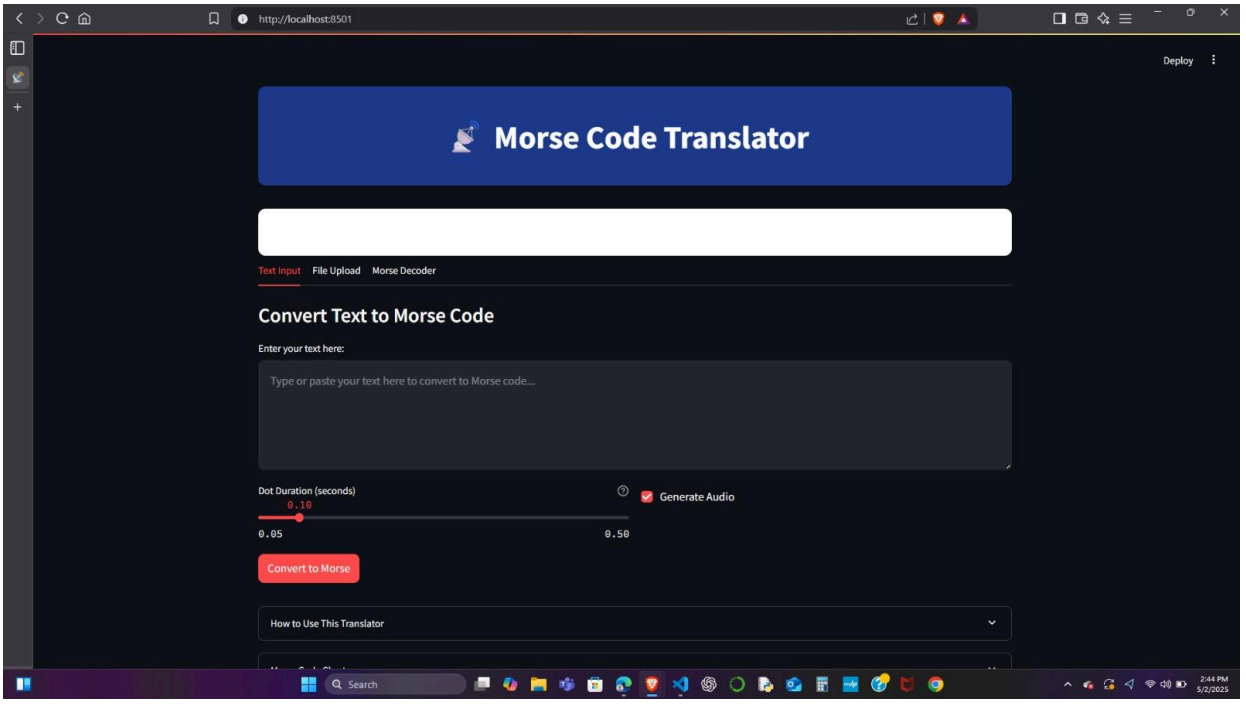


FIGURE 10.1 USER HOME PAGE

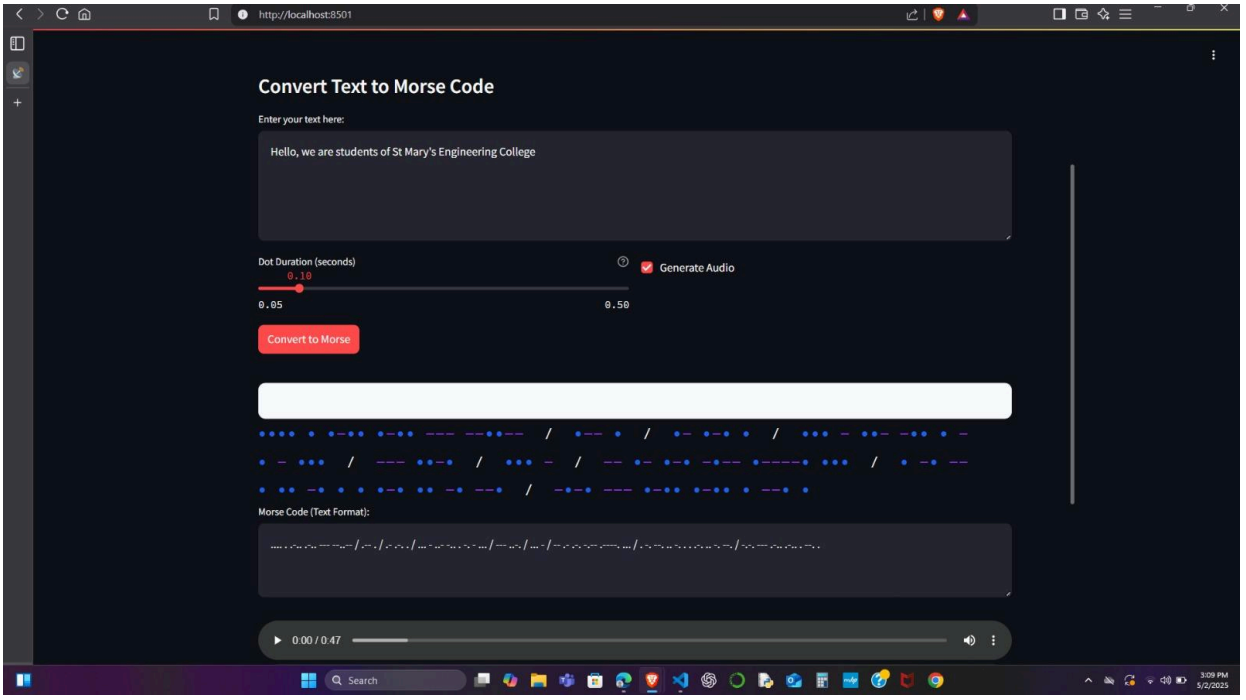
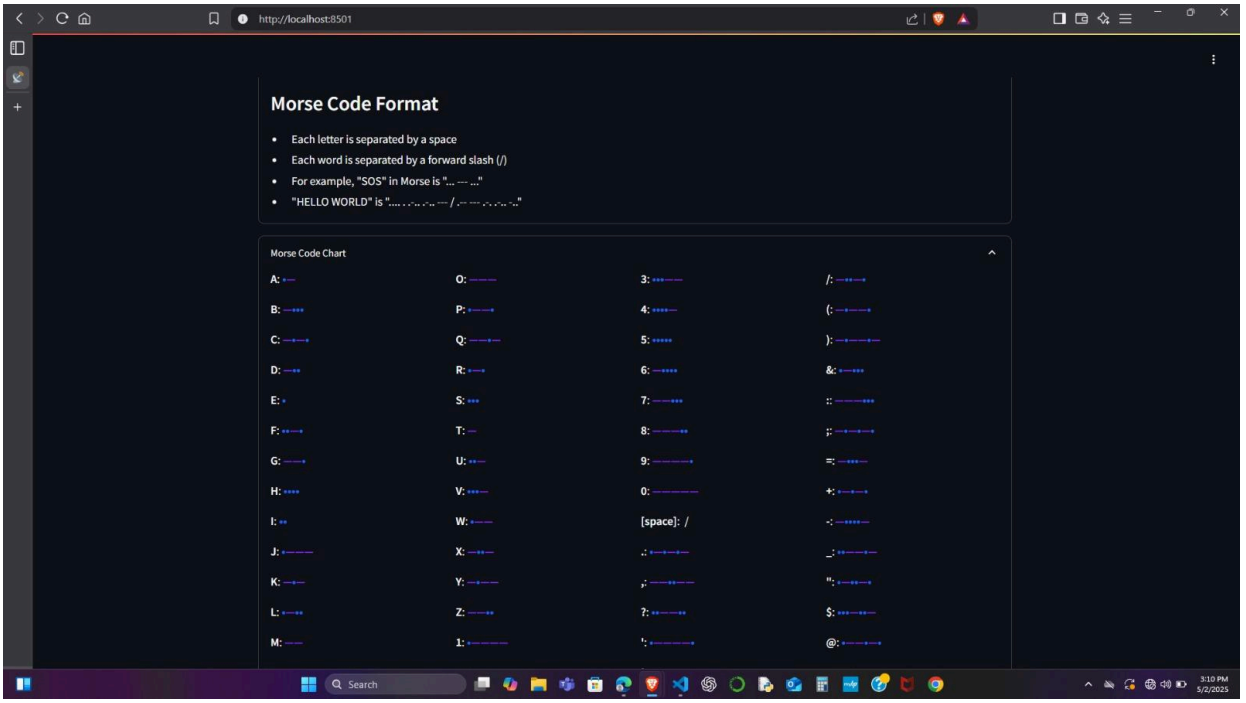


FIGURE 10.2 RESULT



USER MANUAL & DOCS

CHAPTER 11
SYSTEM TESTING

11.1 Unit Testing

Unit testing involves testing individual components or sections of the website, such as navigation bars, buttons, or from validation scripts. Each feature is checked to ensure it performs its intended function without errors.

11.2 Integration Testing

Integration testing verifies that different parts of the website work well together. For example, ensuring that the navigation links load the correct sections and that styled components render properly in all resolutions.

11.3 Functional Testing

This testing focuses on validating the functionality of the website based on the requirements. It checks whether users can browse menus, view, toggle navigation and interact with buttons as intended.

11.4 System Testing

System testing is the complete end-to-end testing of the website as a whole. It ensures that all integrated components work together and that the system behaves correctly in different environments.

11.5 Acceptance Testing

Acceptance Testing is the final phase where the project is tested against the client or end-user requirements. If the website meets the expectations for design, usability, and responsiveness, it is considered ready for deployment.

FUTURE ENHANCEMENT

There are several promising directions for enhancing this project. A notable upgrade would be integrating visual Morse output, such as a blinking light or screen flash, to represent Morse code signals, improving accessibility and engagement. Another potential enhancement is multi-language text support, allowing users to input and translate Morse code in various languages beyond English, thereby expanding the project's global usability. Additionally, developing a cross-platform mobile app version would make the tool more accessible on the go. Using frameworks like Flutter or React Native, users on both Android and iOS could benefit from a seamless, intuitive interface and native device features. These improvements not only broaden the application's functionality but also make it more inclusive, interactive, and widely usable across different demographics and use cases. Future updates could also include voice input/output, customizable Morse timing, and real-time communication features to further enhance user experience and utility.

CONCLUSION

The Web-Based Morse Code Translator built with Python stands out as an effective and user-friendly tool for both learning and practical communication. By addressing the limitations of traditional systems, it provides a bi-directional and interactive platform that seamlessly translates between text and Morse code. The clean, responsive interface enhances user experience, while features like audio playback support reinforce auditory learning and engagement. The application's modular architecture ensures flexibility, allowing for future enhancements and easy integration with other systems. Its offline functionality further adds to its reliability and accessibility in various environments, including educational and remote settings. Whether for hobbyists, students, or individuals with specific communication needs, this tool serves a wide range of users. Overall, the project not only fulfills its core purpose effectively but also lays a strong foundation for future development, making it a valuable contribution to accessible and educational Morse code tools.