USER INTERFACE DESIGN

EX No: 3 **240701463**

Develop and Compare CLI, GUI and VUI for the Same Task and Assess User Satisfaction

---

Aim

To develop a Smart Study Assistant using Command Line Interface (CLI), Graphical User Interface (GUI) and Voice User Interface (VUI) and compare their usability and user satisfaction.

Objective

- Implement the same application using three different interfaces.
- Understand differences between CLI, GUI and VUI interaction styles.
- Analyse which interface provides better usability and accessibility.

Introduction

Human Computer Interaction (HCI) focuses on designing user-friendly systems.
Different interfaces provide different levels of usability, speed and accessibility.

In this experiment, a **Smart Study Assistant** is developed using:

- CLI (Text based)
- GUI (Visual based)
- VUI (Voice based)

All three versions use the same backend logic.

Chosen Application – Smart Study Assistant

The application helps students manage their study tasks.

Features

- Add study topics
- Set deadline and priority
- Mark topic as completed
- View overdue topics
- View completion percentage
- Get daily study suggestion

Tools Used

• Python
• VS Code
• CustomTkinter (GUI)
• SpeechRecognition & pyttsx3 (VUI)
• JSON File Storage

System Architecture

User → CLI / GUI / VUI → Study Manager → JSON Storage → Analytics

This ensures that all interfaces work on the same data.

MODULE DESCRIPTION

Module	Purpose
storage.py	Saves and loads tasks
study_manager.py	Handles task operations
analytics.py	Calculates progress and suggestions
voice_engine.py	Handles speech input and output

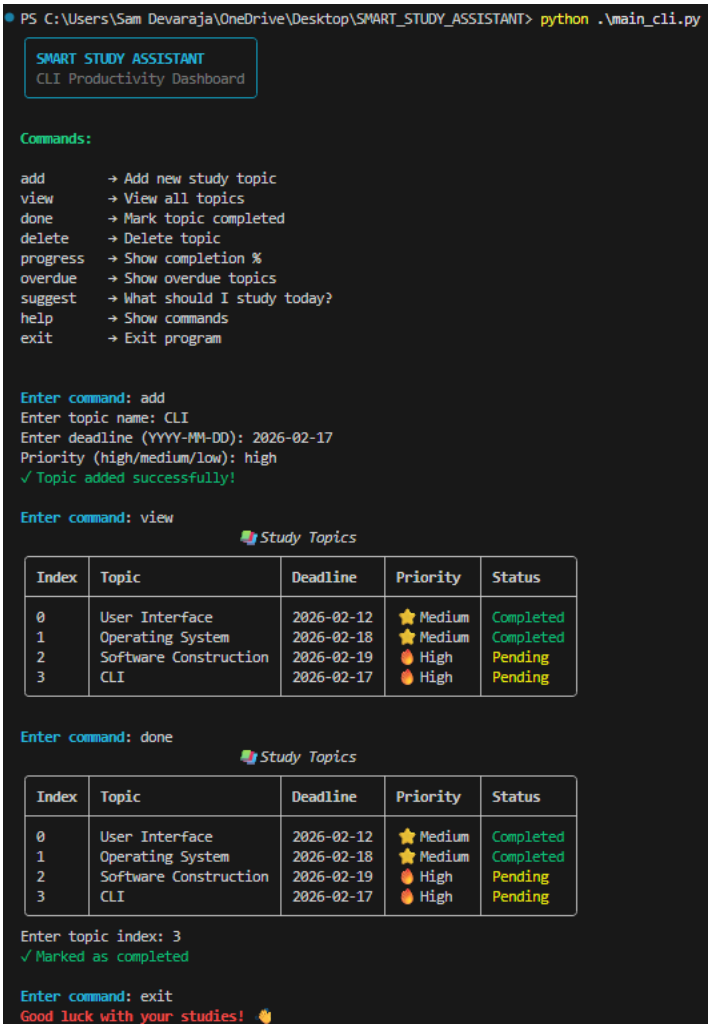
COMMAND LINE INTERFACE (CLI)

Description

CLI is a text-based interface where users type commands to interact with the system.
It is fast and efficient for experienced users.

Features

• Add, view and delete topics
• Mark topics completed
• View overdue tasks
• Show completion percentage



Sample CLI Code

```
from rich.console import Console

from rich.table import Table

from rich.prompt import Prompt

from core.study_manager import add_task, get_tasks

console = Console()

def view_tasks():

    tasks = get_tasks()

    table = Table(title="Study Topics")

    table.add_column("Index")

    table.add_column("Topic")

    for i,t in enumerate(tasks):

        table.add_row(str(i), t["title"])
```

```
    console.print(table)
def add_topic():
    title = Prompt.ask("Enter topic name")
    deadline = Prompt.ask("Enter deadline")
    add_task(title, deadline, 2)
    console.print("Topic added successfully")
while True:
    cmd = Prompt.ask("Enter command")
    if cmd == "add":
        add_topic()
    elif cmd == "view":
        view_tasks()
    elif cmd == "exit":
        break
```

Advantages

- Very fast execution
- Uses minimal system resources

Disadvantages

- Requires typing skills
- Not beginner friendly

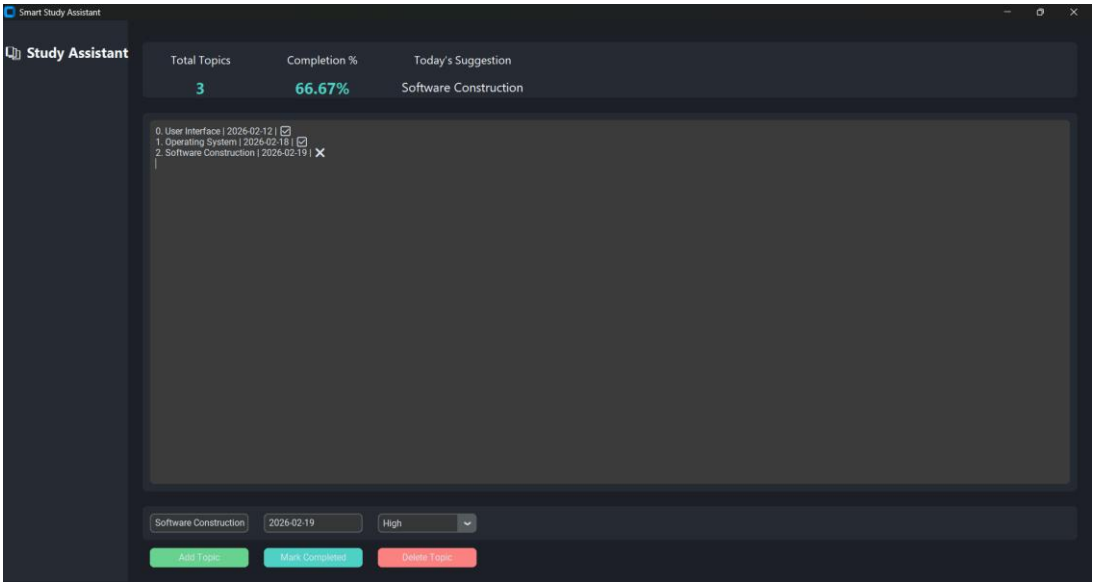
GRAPHICAL USER INTERFACE (GUI)

Description

GUI uses visual components like buttons, forms and lists to interact with the system.
This interface improves usability and reduces user errors.

Features

- Add topic form with deadline & priority
- Task list display
- Completion percentage display
- Study suggestion panel



Sample GUI Code

```
import customtkinter as ctk
from core.study_manager import add_task, get_tasks
from core.analytics import get_completion_rate, suggest_today_task
# App setup
ctk.set_appearance_mode("dark")
app = ctk.CTk()
app.title("Smart Study Assistant")
# Dashboard labels
total_label = ctk.CTkLabel(app, text="Total Topics")
completion_label = ctk.CTkLabel(app, text="Completion %")
total_label.pack()
completion_label.pack()
# Add topic function
def add_topic():
    title = title_entry.get()
    deadline = deadline_entry.get()
    add_task(title, deadline, 2)
# Input fields
title_entry = ctk.CTkEntry(app, placeholder_text="Topic Name")
```

```
deadline_entry = ctk.CTkEntry(app, placeholder_text="YYYY-MM-DD")

title_entry.pack()

deadline_entry.pack()

# Buttons

ctk.CTkButton(app, text="Add Topic", command=add_topic).pack()

app.mainloop()
```

Advantages

- Easy to use
- Provides visual feedback
- Reduces cognitive load

Disadvantages

- Requires mouse interaction
- Slightly slower than CLI

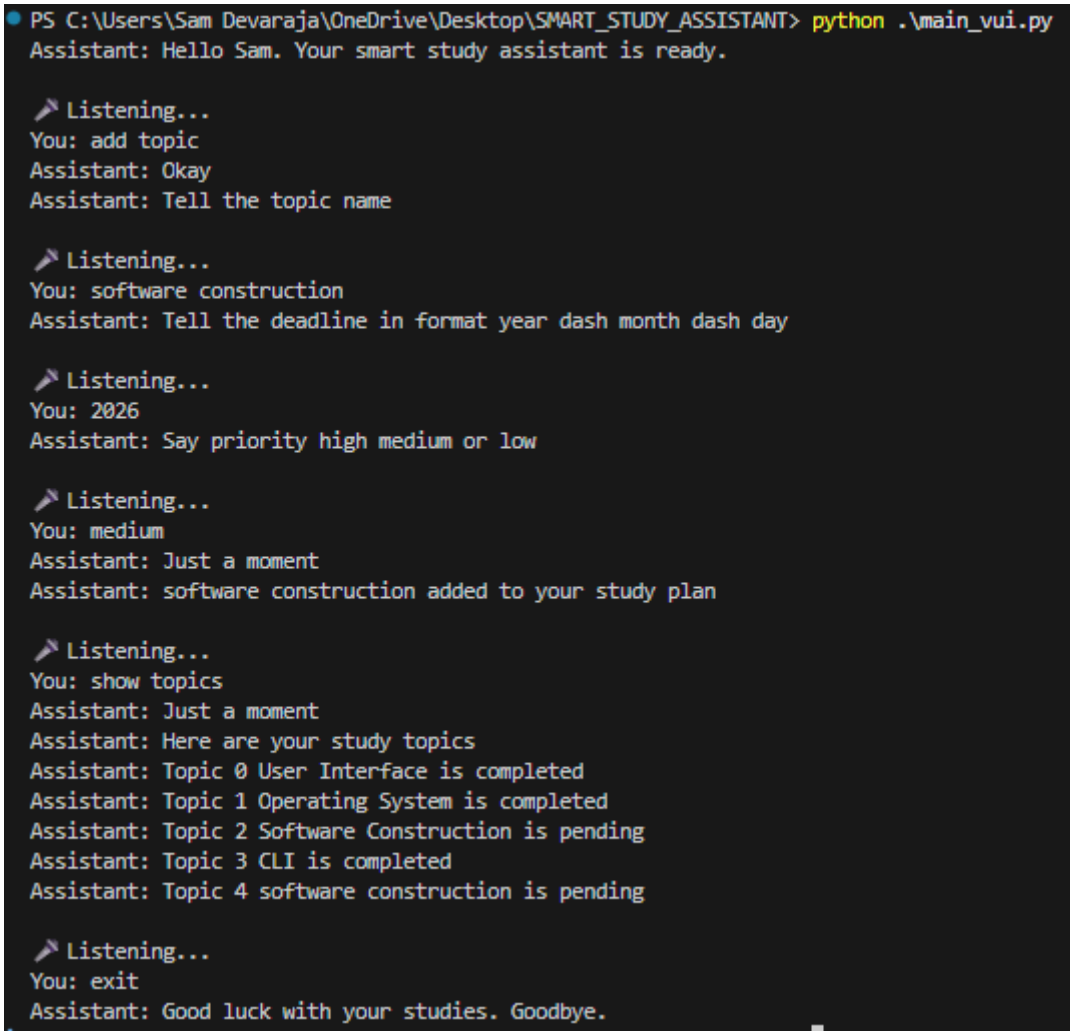
VOICE USER INTERFACE (VUI)

Description

VUI allows users to interact with the system using voice commands.
The assistant listens, processes commands and responds using speech.

Voice Commands

User can say:
- Add topic
- Show topics
- Complete topic
- Progress
- Overdue
- Suggest topic
- Exit



Sample VUI Code

```
from utils.voice_engine import say, listen
from core.study_manager import add_task, get_tasks
def add_topic_voice():
    say("Tell the topic name")
    title = listen()
    say("Tell the deadline")
    deadline = listen()
    add_task(title, deadline, 2)
    say("Topic added successfully")
def show_topics():
    tasks = get_tasks()
    for t in tasks:
```

```
        say(t["title"])
say("Voice Assistant Started")
while True:
    command = listen()
    if "add topic" in command:
        add_topic_voice()
    elif "show topics" in command:
        show_topics()
    elif "exit" in command:
        say("Goodbye")
        break
```

Advantages

- Hands-free interaction
- Accessible for visually impaired users
- Natural communication

Disadvantages

- Requires quiet environment
- Accuracy depends on microphone quality

Comparison of Interfaces

Interface	Ease of Use	Speed	Accessibility
CLI	Low	Very High	Low
GUI	High	Medium	Medium
VUI	Very High	Medium	Very High

User Satisfaction Analysis

- CLI is preferred by technical users.
- GUI is preferred by general users.
- VUI provides best accessibility and modern interaction.

Result

GUI and VUI showed higher user satisfaction compared to CLI because they reduce cognitive effort and provide better interaction.

Conclusion

This experiment shows that modern applications benefit from supporting multiple interfaces.
CLI is efficient, GUI improves usability and VUI improves accessibility.
A combination of these interfaces provides the best user experience.