Submitted By :

Umar Farooq 2022Ag-8043

Sajid Hameed 2022-Ag-8076

Alam Sher 2022-Ag-8081

Submitted To :

Mam Nabeela Ashraf

Subject :

SE – 503

Documentation for "MY Notepad" Python Project

Notepad Documentation

1.1 Introduction

Creating a Notepad application using Python's Tkinter library is a popular project for developers to enhance their programming skills and gain hands-on experience with graphical user interface (GUI) development. A Notepad application provides a convenient and user-friendly interface for creating, editing, and managing text files.

In this project, we leverage the Python programming language and the Tkinter library to develop a Notepad application that offers essential functionalities such as file operations (e.g., creating new files, opening existing files, saving files), text editing capabilities (e.g., copy, cut, paste, undo, redo), and additional features like search and replace, text formatting, and word count.

The Notepad application utilizes the Tkinter library to create the graphical interface, allowing users to interact with the application through menus, buttons, and text areas. File handling functionalities are implemented using Python's built-in modules for reading from and writing to text files. Throughout the project, we focus on ensuring a smooth and intuitive user experience, implementing error handling and validation to manage potential exceptions, and performing rigorous testing to ensure the stability and reliability of the application.

By developing a Notepad application in Python with Tkinter, you will gain valuable knowledge in GUI programming, file handling, event handling, and user interface design. This project serves as a foundation for future endeavors in software development and provides a practical understanding of building user-friendly applications.

1.2 Problem Statement

The problem at hand is to design and implement a Notepad application using the Python programming language and the Tkinter library. The application should provide a user-friendly interface that allows users to create, edit, save, and open text files. Additionally, the Notepad should support basic functionalities such as copy, cut, paste, undo, and redo operations. The application should also include features like find and replace, word count, font customization, and line numbering. Furthermore, the Notepad should have a menu bar with options to create a new file, open an existing file, and save the file. The application should be developed in such a way that it ensures a smooth and efficient user experience, with proper error handling and validation of user inputs. Overall, the goal of this project is to create a functional and user-friendly Notepad application that meets the requirements and expectations of users.

1.3 Objectives

1. Develop a Notepad application in Python using Tkinter that provides essential functionalities such as file operations, text editing capabilities, and search and replace functionality.
2. Implement a user-friendly graphical interface using Tkinter to ensure a smooth and intuitive user experience.
3. Ensure robustness and reliability by incorporating error handling, validation, and thorough testing throughout the development process.

1.4 Scope

Scope of Notepad Application

The scope of a Notepad application in Python using the Tkinter library typically includes the following functionalities:

1. File Operations:

- **Create new text files**: Allow users to create blank text files.
- **Open existing text files**: Enable users to open and view existing text files.
- **Save text files**: Provide the ability to save the current file.
- **Save text files with a different name or location**: Allow users to save the file under a different name or to a different directory.
- **Close files**: Support the functionality to close files, either after saving or discarding changes.

2. Text Editing:

- **Input and display text in a text area**: Allow users to type and display text in a large editable text area, utilizing Tkinter's `Text` widget.
- **Select and manipulate text using operations like copy, cut, and paste**: Implement standard text manipulation features for improving user interaction.
- **Undo and redo text modifications**: Enable undo and redo functionality using Tkinter's built-in commands.
- **Find and replace text within the document**: Include a search and replace feature for locating and modifying text.
- **Support for basic text formatting (font, size, style)**: Allow users to format the text with different fonts, sizes, and styles using Tkinter’s `Text` widget configuration.

3. User Interface:

- **Design a user-friendly graphical interface using Tkinter**: Utilize Tkinter to create an intuitive and responsive graphical user interface (GUI).
- **Implement menus and toolbars for easy access to file operations and text editing functionalities**: Provide standard menu options (e.g., File, Edit, View) and a toolbar for quicker access to features using Tkinter's `Menu` and `Button` widgets.
- **Provide a status bar to display relevant information such as current cursor position or file size**: Include a status bar using Tkinter's `Label` widget to show contextual information like the current line number and character count.
- **Option to customize the appearance of the application**: Implement functionality for users to change the theme, font, or layout of the application using Tkinter's styling options.

4. Additional Features:

- **Word count functionality**: Provide a feature that displays the word count of the document.
- **Line numbering**: Display line numbers in the margin of the text area to help users navigate through the document.
- **Support for keyboard shortcuts**: Implement keyboard shortcuts for quick access to frequently used operations (e.g., Ctrl+C for copy, Ctrl+V for paste) using Tkinter's event bindings.
- **Error handling and validation**: Ensure the application handles potential errors (e.g., file not found, invalid file format) gracefully, with proper validation for file operations.

2.1 Literature Review

The literature review for developing a Notepad application using the Tkinter library in Python focuses on GUI development, file handling, and text editing. Resources such as *"Python and Tkinter Programming"* by John E. Grayson and *"Tkinter by Example"* by Nicholas J. McGuire provide comprehensive guides to developing Python applications with Tkinter. Additionally, *"Python Cookbook"* by David Beazley and Brian K. Jones covers file handling in Python, which is essential for managing file operations like creating, opening, and saving text files. This review ensures a solid understanding of the key concepts needed for building the Notepad application.

2.2 User Classes and Characteristics

The user class for a Notepad application developed with Tkinter includes individuals who require an efficient and easy-to-use tool for text editing and file management. These users may include:

- **Casual Users**: Individuals using the application for basic text editing, such as writing notes or creating simple documents.
- **Students and Academics**: Users who need the application for tasks like note-taking, organizing research, or drafting assignments.
- **Programmers and Developers**: Users who may use the Notepad for writing code, and would benefit from features like syntax highlighting or code snippets.
- **Professionals and Business Users**: Individuals who use the Notepad for drafting business emails, creating to-do lists, or taking meeting notes.

Understanding these user groups helps in designing a Notepad application that meets their specific needs and provides an intuitive and efficient text editing experience.

2.3 Design and Implementation Constraints

- **Cross-Platform Compatibility**: The Notepad application should be designed to run seamlessly on different operating systems, such as Windows, macOS, and Linux.
- **Resource Efficiency**: The application should be efficient in terms of memory usage and processing power, particularly when handling large files.
- **Security and Privacy**: Implement basic security measures, such as the safe handling of files, to protect user data.

2.4 Assumptions and Dependencies

1. Assumptions:

- The user has Python and Tkinter installed on their system to run the Notepad application.
- The user's operating system supports necessary file handling operations.
- The user is familiar with basic text editing and graphical user interfaces.

2. Dependencies:

- **Python and Tkinter**: The application depends on Python for the programming language and Tkinter for creating the GUI.
- **File Handling Libraries**: The application utilizes Python's file handling classes for managing text files, such as `open()`, `os`, and `shutil`.

2.5 Functional Requirements

Functional Requirements for Notepad Application in Python using Tkinter:

File Operations:

- a. Create a new text file.
- b. Open an existing text file.
- c. Save the current file.
- d. Save the current file with a different name or location.
- e. Close the file.

Text Editing:

- a. Input and display text in a text area.
- b. Select and manipulate text using operations like copy, cut, and paste.
- c. Undo and redo text modifications.
- d. Find and replace text within the document.
- e. Support basic text formatting options, such as font, size, and style.

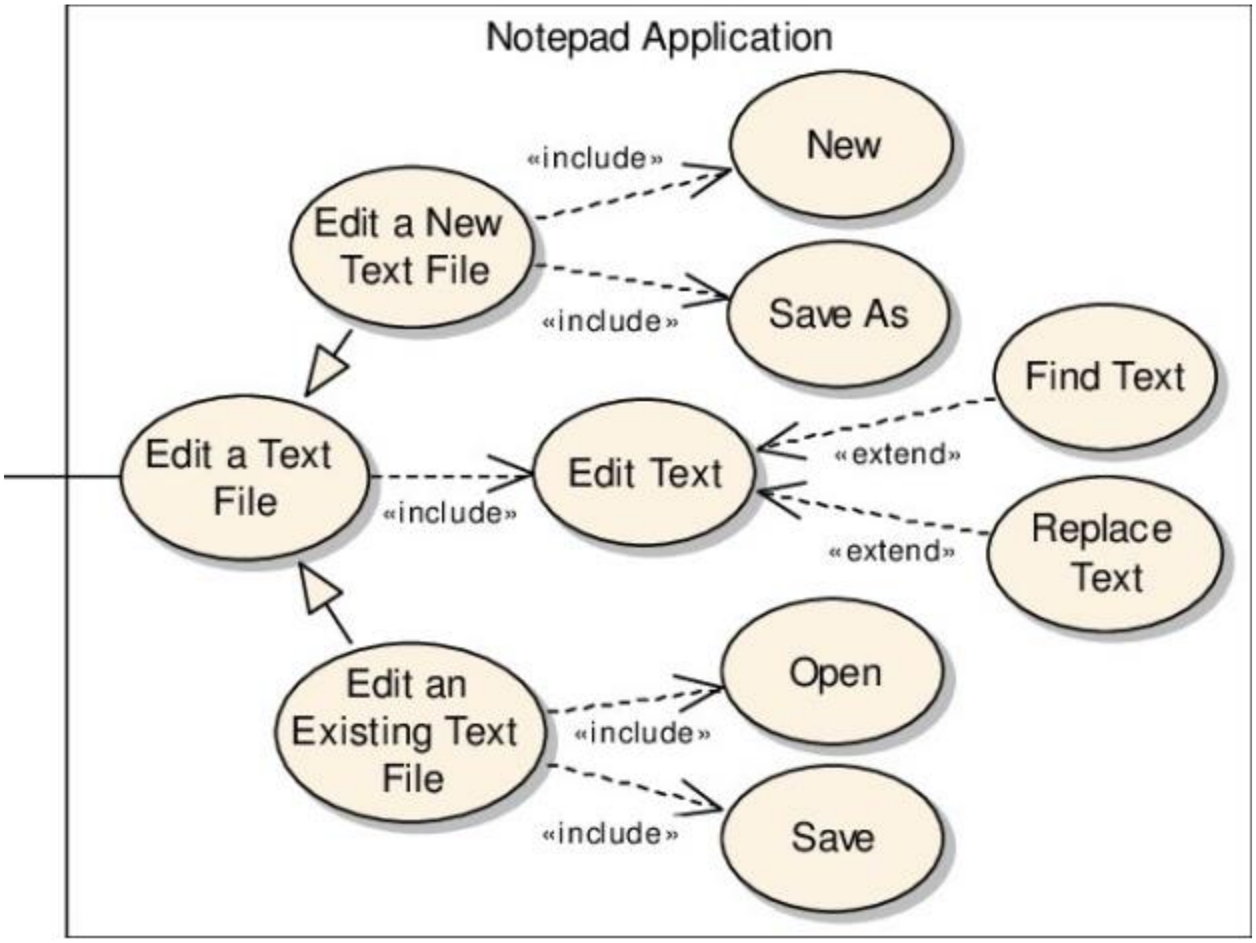
User Interface:

- a. Design a user-friendly graphical interface with menus, toolbars, and a text area for efficient user interaction.
- b. Provide options for customizing the appearance of the application, such as theme selection.
- c. Display relevant information in a status bar, including current cursor position, file size, and word count.

Additional Features:

- a. Implement word count functionality to display the number of words in the document.
- b. Include line numbering to indicate the position of the cursor within the document.
- c. Support keyboard shortcuts for commonly used operations to enhance user productivity.
- d. Handle and display appropriate error messages for file-related exceptions and user input errors.

2.6 USE-CASE Diagram



2.4 Nonfunctional Requirements

Non-Functional Requirements for Notepad Application in Python using Tkinter:

Performance:

1. The Notepad application should provide a responsive and smooth user experience, with minimal lag or delays during text editing and file operations, even when handling large files.
2. The application should efficiently manage system resources, such as memory usage, to ensure optimal performance.

Usability:

1. The user interface should be intuitive, user-friendly, and visually appealing, allowing users to easily navigate and perform tasks without confusion.
2. The application should support common keyboard shortcuts and provide tooltips or contextual help to assist users in understanding various functionalities.

Reliability and Error Handling:

1. The application should handle file-related exceptions, user input errors, and unexpected scenarios gracefully, providing informative error messages to guide users in troubleshooting and resolving issues.
2. The Notepad application should include mechanisms to prevent data loss or corruption, such as automatic backups or recovery options in case of application crashes or unexpected shutdowns.

Security:

1. The Notepad application should prioritize the security and privacy of user data. If needed, it should handle file encryption to protect sensitive information.
2. The application should implement secure file handling mechanisms to prevent unauthorized access or modifications to files.

Compatibility:

1. The Notepad application should be compatible with different operating systems (e.g., Windows, macOS, Linux) to ensure broad accessibility and usability.
2. The application should support various file formats commonly used in text editing, such as plain text files (TXT), Rich Text Format (RTF), and possibly others.

Maintainability and Extensibility:

1. The codebase of the Notepad application should be well-structured, modular, and maintainable to facilitate future enhancements, bug fixes, and code reuse.
2. The application should be designed with extensibility in mind, allowing for the addition of new features or functionalities with minimal code modifications.

These non-functional requirements define the qualities and characteristics that the Notepad application should possess beyond its core functionalities. Adhering to these requirements ensures that the application is reliable, user-friendly, performant, secure, and adaptable for future needs.

3.1 Application and Data Architecture

The architecture of the Notepad application developed with Tkinter in Python follows a layered approach with key components focused on the user interface, file operations, and data management.

1. Presentation Layer:

- **Graphical User Interface (GUI)**: The GUI layer is responsible for handling user interactions and displaying visual components of the application. It is implemented using Tkinter, providing a simple, intuitive interface for text editing and file management. The Tkinter widgets, such as `Text`, `Menu`, and `Button`, are used to build the user interface.

2. Business Logic Layer:

- **File Handling**: This layer manages all file operations like creating, opening, saving, and closing text files. It communicates with the underlying operating system for file interactions and ensures data integrity during these operations.
- **Text Editing**: This component handles text manipulation, including actions such as copy, cut, paste, undo, redo, find, replace, and basic text formatting. The text area, typically represented by Tkinter's `Text` widget, is used to perform these operations.

3. Data Persistence Layer:

- **File System**: This layer handles the storage and retrieval of text files from the file system. It manages reading from and writing to text files using Python's built-in file handling mechanisms (`open()`, `read()`, `write()`).

4. Error Handling and Logging:

- **Error Handling**: This component handles any exceptions that occur during file operations or user input, displaying appropriate error messages to guide users in troubleshooting.
- **Logging**: Logs events, errors, and actions to aid in debugging and monitoring the performance of the application.

5. Additional Components:

- **Configuration**: Manages application settings like default fonts, themes, and user preferences.
- **Utilities**: Helper functions or classes for tasks such as word counting, line numbering, and managing keyboard shortcuts.

This architecture enables the Notepad application to be modular and easily maintainable, with a clear separation of concerns.

3.2 System Architecture

The system architecture for the Tkinter-based Notepad application follows a layered model, where the components are divided into distinct layers that handle specific tasks:

1. **Client-Server Architecture**: In the case of a desktop application, the "client" component is the Tkinter-based GUI running on the user's machine. File operations and data persistence are handled locally, so there is no server component.
2. **Layered Architecture**:
   a. The **Presentation Layer** (GUI) interacts with the user and receives input.
   b. The **Business Logic Layer** handles text editing and file operations.
   c. The **Data Persistence Layer** handles saving and retrieving files to/from the file system.
3. **Event-Driven Architecture**: The application listens for user inputs (such as keyboard presses, mouse clicks, and menu selections) and responds by triggering appropriate actions in the business logic layer, such as editing text or opening files.

The overall architecture ensures efficient text editing and file management with a clear and maintainable structure.

3.3 Component-External Entities Interface

1. GUI Component:

- **External Entities**: User
- **Interfaces**:
  - **User Input**: The GUI receives user input from keyboard actions (e.g., typing text, using shortcuts) or mouse actions (e.g., clicking buttons or menus).
  - **Display Output**: The GUI updates visual elements, including the text area, menus, toolbars, and status bar.

2. File Handling Component:

- **External Entities**: File System, User
- **Interfaces**:

    - **File Operations**: The File Handling component interacts with the file system to open, save, and close files.
    - **User Requests**: It listens for user requests, such as selecting a file to open or saving the file to a new location.

3. Text Editing Component:

- **External Entities**: GUI Component, User
- **Interfaces**:
    - **Text Manipulation**: This component receives user input for text editing actions, such as copy, cut, paste, undo, redo, and text formatting.
    - **User Requests**: It updates the text area based on user commands like modifying text or applying formatting.

4. Error Handling Component:

- **External Entities**: User, System
- **Interfaces**:
    - **Error Handling**: This component receives error messages or exceptions from different components and provides feedback to the user.
    - **System Notifications**: It can generate logs or system notifications for debugging purposes.

5. Data Persistence Component:

- **External Entities**: File System
- **Interfaces**:
    - **File Operations**: This component interacts with the file system to read and write text files, ensuring data is stored correctly and retrieved when needed.

These interfaces define how the internal components of the application communicate with external entities (the user and the file system) to perform the necessary tasks for text editing and file management.

3.4.1 Workflow

- **Cut, Copy, Paste, and Print**:
    - When the user selects the **Cut**, **Copy**, **Paste**, or **Print** menu item, the corresponding inbuilt functions of the Tkinter `Text` widget will be invoked:

        - `cut()` for cutting selected text.
        - `copy()` for copying selected text.
        - `paste()` for pasting copied or cut text.
        - `print()` will be used to trigger printing functionality, depending on the implementation of the print feature in Tkinter.

- **Save**:
    - When the user selects the **Save** menu item, a **File Chooser** will open to allow the user to select a file location and name. After a file is selected, a `FileWriter` or `BufferedWriter` (depending on your implementation) will write the contents of the `Text` widget to the selected file and then close the writer objects to save the changes.
- **Open**:
    - When the **Open** menu item is selected, a **File Chooser** will be invoked. After the user selects a file, a `FileReader` or `BufferedReader` will read the content of the file, and the text of the `Text` widget will be updated with the file's contents.
- **New**:
    - If the user selects the **New** menu item, the text of the `Text` widget will be cleared, setting it to blank for a new document.
- **Close**:
    - When the **Close** menu item is selected, the application window will close by setting the window's visibility to `False` using the `withdraw()` method in Tkinter (or `destroy()` to fully close the window).

:

4.4 Problems Faced

1. **Graphical User Interface (GUI) Design**:
   - **Challenges**: Designing a simple yet functional user interface that users could easily navigate was tough. Striking the right balance between simplicity and feature-richness required significant effort.
   - **Solution**: Prioritizing core features and maintaining a clean, intuitive design helped create a user-friendly interface.
2. **File Handling and Error Management**:
   - **Challenges**: Handling file operations like opening, saving, and closing files while managing errors effectively was complex. This included handling file permissions, error conditions, and unexpected file-related issues.
   - **Solution**: Implementing proper error handling mechanisms and informative error messages helped users resolve issues quickly.
3. **Text Manipulation and Formatting**:
   - **Challenges**: Implementing features like copy, cut, paste, undo, redo, and text formatting while managing the internal state of the text area was tricky. Handling different edge cases, user input, and text changes created challenges.
   - **Solution**: Carefully managing state and ensuring efficient operations helped streamline the text editing process.
4. **Cross-platform Compatibility**:
   - **Challenges**: Ensuring the application worked seamlessly on different operating systems was difficult. File handling and system integration often behaved differently depending on the platform.
   - **Solution**: Extensive testing across different environments helped address platform-specific issues and ensure a consistent experience.

Lessons Learned

1. **Importance of User-Centric Design**:
   - **Lesson**: A user-friendly, intuitive interface is essential for user satisfaction. Collecting user feedback and conducting usability testing provide valuable insights for continuous improvement.
2. **Effective Error Handling**:
   - **Lesson**: Proper error handling mechanisms, along with clear and informative messages, enhance the user experience by making the app more reliable and easier to use.
3. **Test-Driven Development**:
   - **Lesson**: Rigorous testing (both automated and manual) throughout the development process ensures that issues are caught early and resolved efficiently, making the final product more stable.
4. **Continuous Learning and Adaptation**:
   - **Lesson**: Staying updated with new technologies, frameworks, and best practices ensures that the application is built using the most efficient and modern tools available.

Developed By:

Umar Farooq **https://github.com/UmarFarooq9720**

Alam Sher **https://github.com/AlamSher125**

Sajid Hameed **https://github.com/SajidHameed223/**

Resources:

We use this Youtube video for creating this project :

https://youtu.be/d1HyXxeCRg8?feature=shared