Team D Terminal Application Design Document

TEAM D MEMBERS:

1. Jordan Chimeremeze Nwabuike jn22316086@utg.edu.gm
2. Isatou Begay Panneh ip22316076@utg.edu.gm
3. MUSTAPHA B.C SOWE ms22316051@utg.edu.gm
4. Abdoulie Njie an22216005@utg.edu.gm

Architecture Overview

The application follows a modular, object-oriented design pattern with clear separation of concerns between the UI components and terminal functionality and was built with Python and PyQt6. The architecture consists of three main components:

Core Classes

1. **MainWindow** (`main_window.py`)
   - Primary application window class
   - Manages the overall UI layout and styling
   - Handles global shortcuts and command routing
   - Key attributes:
      - `terminal`: Instance of TerminalWidget
      - `current_dir_label`: Displays current working directory

2. **TerminalWidget** (`terminal_widget.py`)
   - Core terminal emulation class
   - Inherits from `QPlainTextEdit`
   - Manages command processing, history, and file operations
   - Key components:
      - Command history management
      - File system operations
      - Custom key event handling
      - Terminal styling and formatting

Data Structures

1. **Command History**
   - Implementation: Python list (`self.command_history`)
   - Tracks previously entered commands

- Supports up/down arrow navigation
- Maintains current history index

2. **File System State**
- Uses `pathlib.Path` for path manipulation
- Tracks current directory (`self.current_directory`)
- Maintains last prompt position (`self.last_prompt_position`)

Key Algorithms

1. **Command Processing**
- Pattern: Command pattern
- Implementation: `process_command()` method
- Parses input into command and arguments
- Routes to appropriate handler method

2. **Directory Navigation**
- Uses path resolution algorithm
- Handles relative/absolute paths
- Supports multiple-level navigation (e.g., `...`)

3. **File Operations**
- CRUD operations for files and directories
- Recursive directory removal
- Safe file system manipulation with error handling

Component Details

Terminal Widget Features

1. **Input Handling**
- Custom `keyPressEvent` handler
- Prevents editing of prompt area
- Supports command history navigation
- Handles special keys (Home, Backspace, etc.)

2. **Display Formatting**
- Custom prompt generation
- ANSI-style color support
- Formatted directory listings
- Welcome message and help text formatting

3. **File System Operations**
- Create/delete files and directories

- Read/write file contents
- Directory listing with metadata
- Safe recursive operations

Event System

1. **Signals**
   - `commandEntered`: Emitted when command is executed
   - Used for command routing and UI updates

2. **Command Confirmation**
   - State-based confirmation system for dangerous operations
   - Maintains operation context during confirmation

Technical Implementation

Path Management

```python
# Example of path resolution algorithm
def change_directory(self, directory):
        try:
        if directory.startswith('.'):
        dot_count = directory.count('.')
        new_dir = self.current_directory
        for _ in range(dot_count):
                new_dir = new_dir.parent
        else:
        normalized_path = directory.replace('\\', '/')
        new_dir = (Path(normalized_path) if normalized_path.startswith('/')
                else self.current_directory / normalized_path)
        new_dir = new_dir.resolve()
        # Additional validation...
        except Exception as e:
        self.appendPlainText(f"\nError changing directory: {e}")
```

Command Processing

```python
def process_command(self, command):
        parts = command.split()
        if not parts:
        return
```

```
        command_map = {
        "help": self.show_help_message,
        "mkdir": self.create_directory,
        "rmdir": self.remove_directory,
        # ... additional commands
        }

        if parts[0] in command_map:
        command_map[parts[0]](*parts[1:])
```

Performance Considerations

1. **Memory Management**
   - Command history size limitation
   - Efficient text buffer management
   - Smart prompt position tracking

2. **File Operations**
   - Asynchronous file operations for large files
   - Buffered reading/writing
   - Safe path manipulation

3. **UI Responsiveness**
   - Debounced updates
   - Efficient rendering
   - Command queue management