Mini Project Report

File Development System

Submitted by
Aniket Banerjee (12000222049)
Shelly Bhattacharjee (12000222063)
Ratnesh Kumar Mandal (12000222047)
Spandan Das (12000222055)

Under Supervision of:
Prof. Md. Keramot Hossain Mondal

DEPARTMENT OF INFORMATION TECHNOLOGY
Dr. B. C. Roy Engineering College

October 2024

Contents

ACKNOWLEDGEMENTS

I would like to express my gratitude to my project guide Prof. Md. Keramot Hossain MondaL for their invaluable guidance and support throughout the project. I am also thankful to my family and friends for their constant encouragement. Additionally, I would like to acknowledge the faculty of the Department of Information Technology at Dr. B. C. Roy Engineering College Durgapur for providing the resources necessary to complete this project.

1. INTRODUCTION

The File Management System is a full-stack web application that simplifies file and directory management through a browser-based interface, making it especially useful for server environments. Built with Python's Flask framework, this application integrates HTML for structuring content, CSS for styling, and Bootstrap for responsive design, creating an organized and user-friendly experience.

At its core, the system allows users to perform essential file operations, including creating, renaming, deleting, and moving files and directories. Each operation is mapped to specific backend functions in Flask that utilize Python's 'os' and 'shutil' libraries, handling server-based file actions with precision and security. The backend code includes error-handling mechanisms, ensuring that any issues (such as trying to rename a non-existent file) are met with clear feedback. This feedback is facilitated by Flask's flash messaging system, which displays real-time notifications directly on the interface, so users are aware of each operation's outcome.

The system is designed to be highly accessible and mobile-responsive, thanks to the use of Bootstrap, which allows users to interact with the file manager on various devices, from desktops to mobile phones. The design is visually straightforward, with lists of files and directories, as well as options for each file action, presented in a clean, easily navigable layout. The application's modular and flexible architecture not only provides a smooth user experience but also lays a foundation for future features like user authentication, version control, or even file previews. In essence, this File Management System bridges backend file management with a web-friendly interface, making it a powerful solution for organized file handling in web or server-based environments.

1.1 Overview

The application begins by setting up the core backend operations for file and directory management, encapsulating each action—such as creating a directory, deleting a file, or renaming an item—within dedicated functions. These functions use Python's OS and `shutil` libraries, which provide robust file-handling capabilities directly on the server. For instance, the `create_directory` function leverages `os.makedirs()` to create new directories, while error handling ensures that users are notified if a directory already exists. Similarly, the `delete_directory` function recursively deletes directories using `shutil.rmtree()`, providing functionality for safely removing directories and their contents.

The web routes—defined in Flask—act as the interface between the user's actions in the browser and the server-side file operations. Each action has a dedicated route, such as `/create-directory` for creating directories or `/move-file` for moving files. Flask's flash messaging provides real-time feedback to users, informing them of successful or failed actions, improving the user experience and ensuring clarity. For example, if a user tries to delete a file that doesn't exist, they are immediately informed of the issue through a flash message, enhancing interaction reliability.

In the front end, HTML and CSS render a clean interface, displaying the current directory's contents and providing interactive buttons and forms for creating, renaming, and moving files. Each function is organized into sections within the UI, and form submissions for these actions are processed by Flask's backend routes. The Bootstrap framework and custom CSS are used to make the layout responsive, ensuring compatibility with a variety of devices and screen sizes.

Overall, this File Management System is an ideal tool for individuals or teams needing quick, reliable access to server-based files in an organized and responsive web environment. Its modular structure also makes it easily extensible, allowing additional features like authentication, file upload, or logging to be integrated as needed.

1.2 Methodology

For a File Management System project, the methodology generally involves structured phases to ensure thorough planning, development, testing, and implementation. Here's a typical methodology:

1. **Requirement Analysis:**

   Gather Requirements: Identify the needs of the end-users (students, employees, administrators) regarding file management, including file organization, security, and retrieval features.

   Define Scope and Objectives: Outline the main objectives and scope of the project, specifying key features like search functionality, file categorization, and permissions.

   Feasibility Study: Assess the technical and operational feasibility to determine resources, technologies, and potential constraints.

2. **System Design:**

   System Architecture: Plan the architecture (e.g., client-server, cloud-based) to ensure scalability and performance.

   Database Design: Design a database schema for storing file metadata, user permissions, and logs.

   User Interface Design: Develop a wireframe and mockups of the user interface, focusing on usability and intuitive navigation.

   Security Design: Specify access control, encryption, and authentication protocols to ensure data security.

3. **Technology Selection:**

   Choose Development Tools: Select programming languages (e.g., Java, Python), frameworks (e.g., Django, Spring Boot), and databases (e.g., MySQL, MongoDB).

   Platform Decisions: Decide on operating systems, whether the system will be cross-platform, and compatibility with various file types.

4. **Development and Implementation:**

   Modular Development: Develop the system in modules, such as file upload, search, categorization, and security.

   Version Control: Use version control (e.g., Git) to manage code changes, collaborate efficiently, and maintain version history.

   Integration of Components: Integrate modules to ensure the system functions cohesively and data flows smoothly.

5. **Testing and Quality Assurance:**

   Unit Testing: Test individual modules to confirm that each component functions correctly.

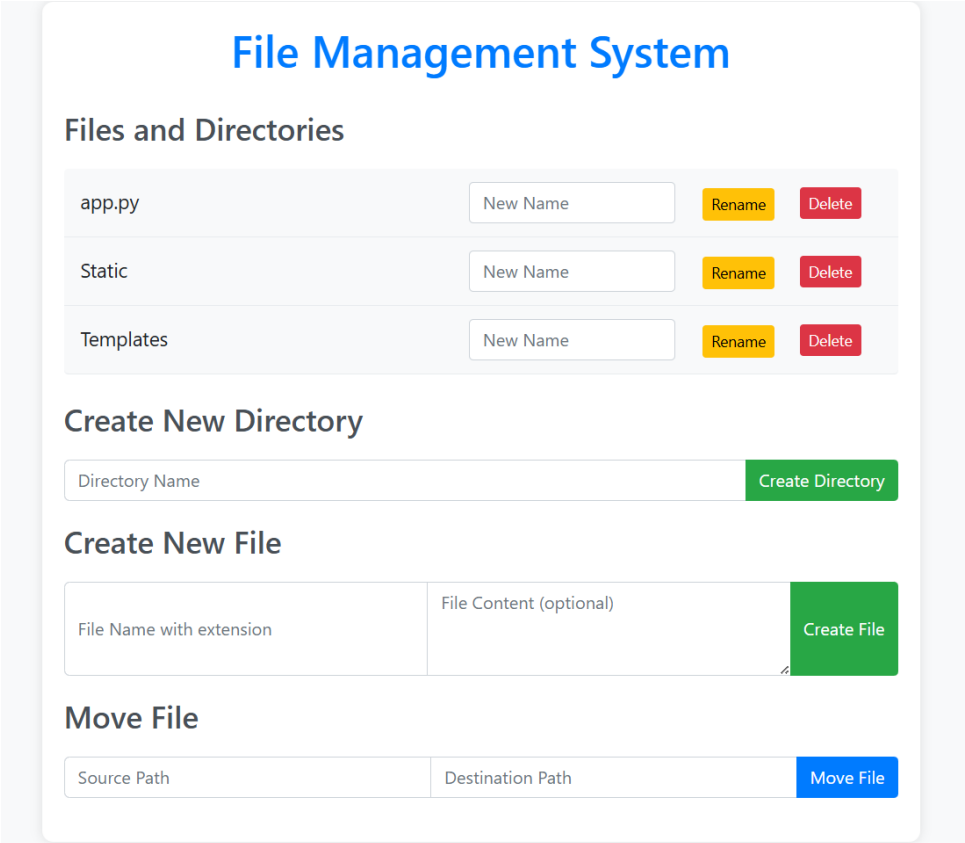
   Integration Testing: Verify that all modules work together as expected.
   Performance Testing: Test system efficiency with large datasets to confirm scalability.

   User Testing: Gather feedback from real users to identify any usability or functionality issues.

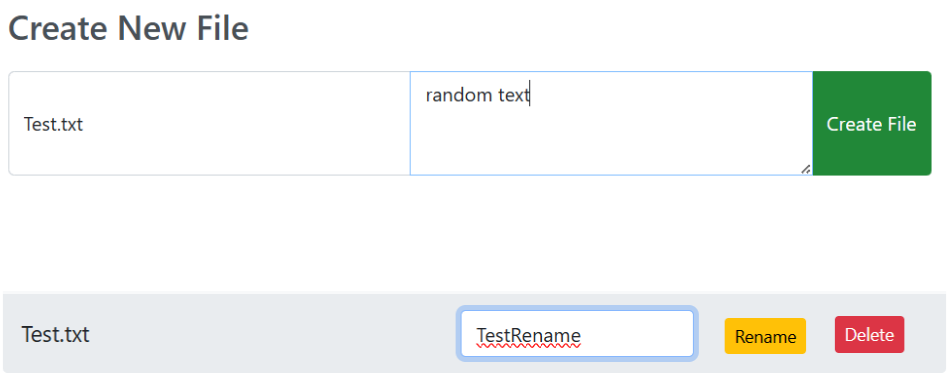
2. TOOL DESCRIPTION

2.1 User Interface

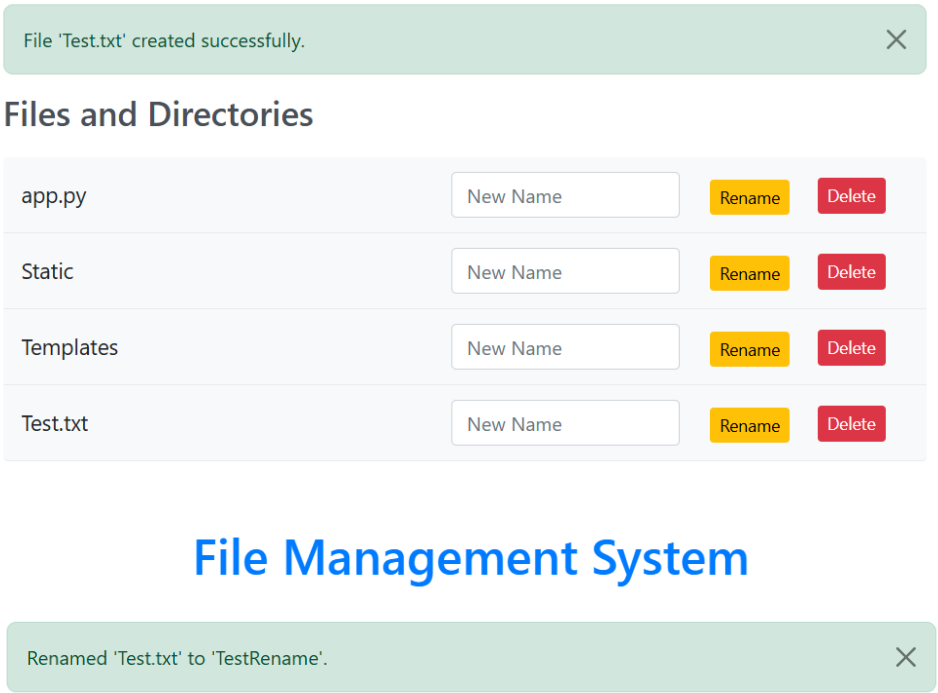
1. **Dashboard Layout:** The main dashboard displays a hierarchical view of files and directories. Users can navigate through folders, view file details, and access file actions easily. Icons and breadcrumbs guide users through the directory structure, simplifying navigation and reducing cognitive load.



2. **File Operations Panel:** The operations panel on the right side (or in a dropdown menu on mobile) provides options like "Create," "Rename," "Delete," and "Move." Each action is accompanied by form fields, with clear labels and submit buttons, allowing users to perform tasks efficiently.



3. **Flash Messaging System:** Feedback is provided to users through a flash messaging system, which displays success or error messages at the top of the page. This system ensures users receive immediate responses to their actions, enhancing the overall user experience.



2.2 Technology Stack

1. **HTML:**

   Purpose: HTML provides the structure for the web page, defining elements such as headings, forms, and lists for file management functionality.

   Key Features Used: The `<!DOCTYPE html>` declaration specifies HTML5, ¡head¿ defines metadata (title, links to CSS), and ¡body¿ contains the main content, including forms and interactive elements.

2. **Jinja2 Templating (Flask)**

   Purpose: Jinja2 is a templating engine for Flask, allowing dynamic content rendering within HTML. It provides control structures (like loops and conditionals) to manage file lists, flash messages, and forms.

3. **Bootstrap (v5.1.3):**

   Purpose: Bootstrap provides a responsive, mobile-first CSS framework, with pre-styled components and utilities, to make the interface visually appealing and user-friendly.

   Key Features Used: Bootstrap classes like container, mt-5, and btn define layout, spacing, and button styles. alert-dismissible allows alert messages to be dismissed by the user.

4. **CSS (External File):**

   Purpose: Custom styles are applied using CSS in style.css, allowing further customization beyond Bootstrap's pre-set styles.

5. **Forms and User Input:**

   Purpose: Forms allow users to interact with the system, performing tasks like renaming, creating, deleting, and moving files and directories.

   Key Features Used: Input fields and buttons (e.g., input, button) for each action, and POST methods for secure data handling.

6. **JavaScript (Bootstrap Bundle):**

   Purpose: JavaScript, particularly through the Bootstrap JS bundle, enables dynamic interactions, such as closing flash messages.

   Key Features Used: Bootstrap's btn-close component, dismissible alerts, and JS-based form validations allow for a smooth, interactive user experience.

7. **OS module (Python):**

   Purpose: The os module provides functions for interacting with the operating system, including creating, listing, and renaming files and directories.

   Key Features Used: File and Directory Management: Functions like os.makedirs, os.rename, os.remove are used for directory creation, file renaming, and deletion. File Listing: os.scandir lists files and directories, allowing the app to dynamically display file content.

8. **Shutil module (Python):**

   Purpose: The shutil module is used for high-level file operations, such as recursively deleting directories and moving files.

   Key Features Used: Directory Deletion: shutil.rmtree removes a directory and its contents. File Moving: shutil.move moves files between directories, handling more complex paths and overwriting rules than os.rename.

2.3 Specification

This section outlines the technical specifications of the File Management System, covering the structure and functionality of various components:

1. **HTML Structure and Meta Information**

   DOCTYPE: `<!DOCTYPE html>` indicates HTML5, making the document modern-browser-compatible.

   Language and Charset: Sets the page language to English (`lang="en"`) and character encoding to UTF-8 for multi-language compatibility.

   Viewport Configuration: Ensures the page is responsive on different screen sizes.

   Page Title: Sets the title in the browser tab to "File Management System."

   Bootstrap CSS: Loads Bootstrap from a CDN for grid layout and component styling.

   Custom CSS: Links a stylesheet (`style.css`) located in Flask's static directory, allowing for additional styling customizations.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>File Management System</title>
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css">
    <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
</head>
```

2. **Container and Main Title**

   Bootstrap Container: A `<div>` with the class `container mt-5` wraps the main content, applying a margin-top for spacing and centering the content in a responsive container.

   Page Header: Displays "File Management System" as the main header.

```
<div class="container mt-5">
    <h1>File Management System</h1>
```

3. **Flash Messages Section**

   Purpose: Displays feedback messages to users after performing file operations.

   Flask's flash Messaging System: Flash messages are retrieved from the backend. Each message has a category (e.g., success, error) that corresponds to the outcome of a user action.

```
<!-- Flash messages -->
{% with messages = get_flashed_messages(with_categories=true) %}
    {% if messages %}
        <div class="mt-3">
            {% for category, message in messages %}
                <div class="alert alert-{{ category }} alert-dismissible fade show" role="alert">
                    {{ message }}
                    <button type="button" class="btn-close" data-bs-dismiss="alert" aria-label="Close"></button>
                </div>
            {% endfor %}
        </div>
    {% endif %}
{% endwith %}
```

4. **Files and Directories List**

   Bootstrap List Group: Uses an unordered list with `list-group` classes to display files and directories. Each item (`item in files`) represents a file or directory.

   Jinja2 Template Logic: Loops through each file or directory item, which is passed from the backend. Each item displays within a list item (`<li>`) for an organized view.

   Rename and Delete Options:

   - **Rename Form:** Inline form for renaming a file/directory.
     - **Input Fields:** A hidden field `old_name` holds the current name, and a text input named `new_name` allows users to enter the new name.

- **Button:** Submits the form to the `handle_rename` route when the rename button is clicked.

```python
@app.route('/rename', methods=['POST'])
def handle_rename():
    old_name = request.form.get('old_name')
    new_name = request.form.get('new_name')
    rename_file(old_name, new_name)
    return redirect(url_for('index'))
```

- **Delete Form:** Inline form to delete a file or directory.
  - **Hidden Input:** `file_path` carries the item's path to identify it on the server.
  - **Button:** Submits the form to `handle_delete_file` route, triggering the deletion action.

```python
@app.route('/delete-file', methods=['POST'])
def handle_delete_file():
    file_path = request.form.get('file_path')
    delete_file(file_path)
    return redirect(url_for('index'))
```

5. **Create New Directory Section**

   Form Layout: Uses an `input-group` to collect the new directory name and a submit button.
   Fields:

   - **Text Input:** Named `directory_name` for entering the directory name.
   - **Submit Button:** Triggers the form's action to the `handle_create_directory` route to create the directory on the server.

6. **Create New File Section**

   Form Layout: Similar to the directory creation form but includes a text area for optional file content.

Fields:

- **File Path:** Text input for the new file name with an extension.
- **Content:** A text area where users can add optional content to the new file.
- **Submit Button:** Submits to the `handle_create_file` route, creating the file.

```python
@app.route('/create-file', methods=['POST'])
def handle_create_file():
    file_path = request.form.get('file_path')
    content = request.form.get('content')
    create_file(file_path, content)
    return redirect(url_for('index'))
```

7. **Move File Section**

   Purpose: Allows users to move a file from one location to another.

   Form Layout: Uses two text input fields (source and destination) within an `input-group`.

   Submit Button: Triggers form submission to the `handle_move_file` route, which moves the file server-side.

```python
@app.route('/move-file', methods=['POST'])
def handle_move_file():
    source = request.form.get('source')
    destination = request.form.get('destination')
    move_file(source, destination)
    return redirect(url_for('index'))
```

2.4 Analysis

The file management system developed using Flask demonstrates essential functionality for managing files and directories in a web environment. The project's analysis considers aspects such as functionality, user experience, security, and scalability:

1. **Functionality:**

   The system successfully integrates core file management functions, including creating, renaming, deleting, and moving files and directories. These features are implemented using Python's standard libraries, ensuring compatibility and efficiency.

   Each function is mapped to a dedicated Flask route, making it straightforward to call each operation based on user actions. This modularity allows for easier testing, debugging, and potential expansion of features.

2. **User Experience:**

   The system provides a user-friendly interface with Bootstrap styling, making the web application visually appealing and intuitive. Users can perform file operations directly from the interface without needing command-line access, making it accessible to those without technical expertise.

   Flash messages provide real-time feedback on operations, informing users of successful actions or any errors, which enhances usability and helps in identifying issues promptly.

3. **Scalability and Limitations:**

   While the application is effective for small-scale file management, handling extensive directories or very large files may affect performance due to the limitations of synchronous operations and Flask's default single-threaded environment.

   Future versions could implement asynchronous handling or integration with a backend file storage system for better scalability.

2.5 Future Work

1. **File Upload and Download**

   Integrate file upload and download capabilities for handling external files and backing up local content.

2. **Enhanced Search Functionality**

   Add advanced search features, including filtering by file type, date, and size for efficient file retrieval.

3. **Cloud Storage Integration**

   Allow for integration with cloud storage services (e.g., Google Drive, Dropbox) for external file management options.

4. **User Authentication and Access Control**

   Implement user login with role-based access to restrict file operations to authorized users only.

5. **File Preview and Editing**

   Enable users to preview common file types (e.g., PDFs, images, text files) directly within the application.

6. **Version Control and History Tracking**

   Include version tracking for files to maintain a history of changes and facilitate rollback to previous versions.

7. **Error Logging and Monitoring**

   Implement error logging to track issues in real-time, improving system reliability and troubleshooting.

REFERENCES

1. **Linux Foundation, Filesystem Hierarchy Standard (FHS) 3.0**
   https://refspecs.linuxfoundation.org/FHS_3.0/fhs-3.0.pdf

2. **Micorsoft, File Systems in Windows**
   https://refspecs.linuxfoundation.org/FHS_3.0/fhs-3.0.pdf

3. **Python Standard Library - os Module**
   https://docs.python.org/3/library/os.html

4. **Python Standard Library - shutil Module**
   https://docs.python.org/3/library/shutil.html

5. **Flask Documentation (3.0.x)**
   https://flask.palletsprojects.com/en/stable/

6. **Jinja Documentation (3.1.x)**
   https://jinja.palletsprojects.com/en/stable/