Pau Santana's FileSystem Project

Introduction

The purpose of this project is to simulate a file system using object-oriented programming principles. It provides functionality to create directories, files, navigate through the file system, read and update file contents, delete files and directories, and move files and directories to different locations.

Features

- Create directories and files
- Read and update file contents
- Delete files and directories
- Move files and directories to different locations
- List the contents of a directory
- Change the current directory
- Print the current directory path
- Clear the screen

Architecture/Design

The project follows an object-oriented design with three main classes: `Directory`, `File`, and `FileSystem`. The file system is represented as a tree structure, where each directory is a node in the tree and the root directory is the starting point. Directories and files are organized hierarchically, with each directory having a reference to its parent directory and a dictionary of its children.

Implementation

```
class Directory:
    def __init__(self, name, parent=None):
        self.name = name
        self.parent = parent
        self.children = {}


class File:
    def __init__(self, name, contents, parent=None):
        self.name = name
        self.contents = contents
        self.parent = parent


class FileSystem:
    def __init__(self):
        self.root = Directory('')
        self.current_dir = self.root

    # Methods for creating, reading, updating, and deleting files and directories

    def list_dir(self, path='.'):
        dir_ = self.get_node(path)
        if dir_ is None:
            return 'Directory does not exist'
        if isinstance(dir_, Directory):
            return list(dir_.children.keys())
        else:
            return 'Cannot list a file'

    def change_dir(self, path):
        dir_ = self.get_node(path)
        if dir_ is None:
            return 'Directory does not exist'
        if isinstance(dir_, Directory):
            self.current_dir = dir_
            return 'Changed directory'
        else:
            return 'Cannot cd into a file'

    def current_path(self):
        path = []
        dir_ = self.current_dir
        while dir_.parent is not None:
```

```
            path.append(dir_.name)
            dir_ = dir_.parent
        return '/' + '/'.join(reversed(path))
```

Usage

To use the project, follow these steps:

1. Instantiate the `FileSystem` class: `fs = FileSystem()`
2. Use the available commands to interact with the file system:

- `create(path, contents=None)`: Create a file or directory at the specified path.
- `read(path)`: Read the contents of a file.
- `update(path, contents)`: Update the contents of a file.
- `delete(path)`: Delete a file or directory.
- `move(source_path, dest_path)`: Move a file or directory to a different location.
- `list_dir(path='.')`: List the contents of a directory.
- `change_dir(path)`: Change the current directory.
- `current_path()`: Print the current directory path.
- `clear_screen()`: Clear the screen.

Security Considerations

This project does not specifically address security measures or vulnerabilities. However, when implementing a file system or any software project, it is important to consider security best practices, such as input validation, access control, and encryption, to protect against potential vulnerabilities and unauthorized access.

Challenges and Learnings

During the development of this project, some challenges were faced, such as:

- Designing an efficient data structure to represent the file system hierarchy.
- Implementing the logic for navigating through directories and managing file and directory operations.
- Handling edge cases and error conditions.

Through these challenges, valuable lessons were learned about object-oriented design, data structures, and file system operations.

Future Improvements

Some potential future improvements for this project could include:

- Adding support for file permissions and access control.
- Implementing file search functionality.
- Enhancing the user interface with a graphical interface or command-line autocompletion.

References

No external libraries or tools were used in this project.