Submitted by:
Name: Nancy
Github

LinkedIn

Personal projects:

1. Tale Travels
   Tech Stack: HTML, CSS, Bootstrap, Node.js, Express.js, MongoDB
   ● An application for exploring and sharing travel destination information.
   ● It offers two acess levels. Implemented user authentication which provide verified users with CRUD functionality and non-verified users with view-only access.
   ● Integrated interactive mapping for users to visualize visited locations. Other funtionalities include commenting and reviews over the post for enhanced engagement.
   ● Designed and implemented a MongoDB database for efficient storage and retrieval of user-entered details.

2. Optical Character Recognition
   Tech Stack: Python, OpenCV, Pytesserect
   ● Python based OCR application in which the code effectively extracts text from images, ensuring accurate character recognition for diverse applications.
   ● Incorporated advanced image processing techniques which ensures extraction of text from images in challenging conditions.
   ● Utilized bounding box detection and visualization methods to present the recognized text on the image, offering a user-friendly interface for easy verification and analysis of OCR results.

In-Memory File System Documentation

Introduction:

The application is built in Node.js and it supports various functionalities like: **mkdir, cd, ls, grep, cat, touch, echo, mv, cp, rm**. All the operations handle appropriate error cases, invalid inputs and edge cases gracefully.

Getting Started:

1. Clone the Repository from Github. [Github link]
2. Do environment setup:
   - Install Node.js as per your system (windows/mac).
     - You can download and install Node.js from the official website: Node.js
     - Verify Installation: Open a terminal and run the following commands to verify that Node.js is installed. **node -v**
     - You should see the version numbers for Node.js, indicating that the installation was successful.
     - Navigate to the cloned directory.
     - Run the following command to initialise the project: **npm init**
     - This will create a package.json file.
     - Now you can run the application using the following command: **node app.js**

Implementation Details:

System Implementation

- **File System Object Classes:-**

  FileSystemObject Class
  - Serves as the base class for files and directories.
  - Contains a name property and an abstract display method.

  File Class
  - Extends FileSystemObject and represents files.
  - Includes a content property for storing file content.
  - Provides methods for displaying, getting, and setting file content.

  Directory Class
  - Extends FileSystemObject and represents directories.
  - Contains an array (contents) to store files and subdirectories.
  - Provides methods for adding files and directories, listing contents, finding objects, and setting the parent directory.

- **Terminal Class**
  - **Overview**
    - The Terminal class serves as the main interface for the file system.
    - Utilizes the readline module for user input.
    - Provides methods for starting the terminal, running the main loop, and executing various file system commands

  - **Execute Commands**
    - The executeCommand method processes user commands, providing all the functionalities.
    - It parses and executes the user-entered command by invoking the appropriate method based on the command type.
    - It also contains method definitions of all the functionalities.

Data Structures Used:

- The primary data structures used in the implementation are classes representing file system objects, files, and directories.
- Arrays are used to store the contents of directories.

Design Decisions:

- Implemented an object-oriented design with classes for different file system elements for better modularity and maintainability.
- Utilized recursion for navigating through directories, making the code more concise and readable.
- Choose a CLI approach for user interaction, providing a familiar interface for users accustomed to command-line operations.

Functionalities:

The in-memory file system supports the following functionalities:

1. **mkdir**
   - Creates a new directory.
   - Example inputs:
     - **mkdir folder_name**: Create a directory named folder_name in the current working directory.
     - **mkdir folder1 folder2**: Create two directories named folder1 and folder2 in the current working directory.
     - So on, we can create as many directories in a single command.
2. **cd**
   - Used to navigate through the directories.
   - Example inputs:
     - **cd ..** : Move to the parent directory.
     - **cd /** : Move to the root directory.
     - **cd folder1/folder2/folder3** : Navigate to the specified relative path.
     - **cd /folder1/folder2/folder3** : Navigate to the specified absolute path.
3. **ls**
   - List the contents of the current directory.
   - Example inputs:
     - **ls** : List the contents of the current directory.
     -
4. **grep**
   - Search for a specified pattern in a file.
   - Example inputs:
     - **grep seach_pattern** : Search for the pattern “seach_pattern” in all the current directories files and list all the files containing that pattern.
5. **cat**
   - Display the contents of the file.
   - Example input:
     - **cat file_name.extension** : Displays the content of the file.
6. **touch**
   - Create a new empty file.
   - Example input:
     - **touch file_name.extension** : Creates a new empty file of that name.
7. **echo**
   - Write text to a file.
   - Example input:
     - **echo "input text" > file_name.extension** : Write “input text” to that file.
8. **mv**
   - Move a file or directory to another location.
   - Example inputs:
     - **mv folder1 folder2** : Move folder1 to folder2.
     - **mv folder1/folder2/folder3 folder4** : Move folder3 to folder4.
     - **mv folder1/folder2/file_name.txt folder3**: Move the file_name.txt to folder3.
     - **mv folder1/folder2/file_name.txt folder3/folder4** : Move the file_name.txt to folder4.

9. **cp**
    - Copy a file or directory to another location.
    - Example inputs:
        - **cp folder1 folder2:** Copy folder1 and its contents to folder2.
        - **cp folder1/folder2 folder3:** Copy folder2 and its contents to folder3.
        - **cp folder1/file_name.txt folder2:** Copy the file_name.txt and its contents to folder2.
        - **cp folder1/file_name.txt folder1/folder2**: Copy file_name.txt and its contents to folder2.

10. **rm**
    - Remove a file or directory.
    - Example inputs:
        - **rm file_name.txt:** Remove (delete) file_name.txt from the current working directory.
        - **rm folder1/folder2**: Remove (delete) folder2 from folder1.
        - **rm folder1/file_name.txt:** Remove (delete) the file_name.txt from folder1.