Report on File System Search Functionality Solution

Introduction

The task was to create a file system search functionality **using PHP , with a front-end interface developed in ReactJs.** adhering to object-oriented principles and Model and Control design, and to develop a REST API for searching the file system. The goal was to provide a scalable, maintainable, and efficient solution, suitable for integration into a larger system.

Solution Overview

The solution consists of several components: a file system model, a search functionality, a REST API, and basic manual unit tests.

Components and Performance Analysis

File System Model: Implemented using object-oriented principles, the file system model represents files and directories. It utilizes the Composite design pattern for a hierarchical structure and encapsulates file system behavior in Element and ElementService classes.

Search Functionality: The search is implemented in a way that it can recursively traverse the file system, looking for files and directories matching a given keyword. This was achieved through a combination of the Model and Controller Method design patterns.

REST API: A RESTful API was created using native PHP to provide an interface for searching the file system. It handles HTTP GET requests, extracts the search keyword, and returns matching results in JSON format to Frontend.
Unit Testing: Basic manual unit testing was implemented to validate the functionality of the file system search. This was done without the use of any external testing framework.

Frontend
React Application: Developed using Vite for fast builds and an efficient development experience.
Search Component: A React component (Search.js) allowing users to input a search keyword and display results.
Integration with Backend: The frontend communicates with the PHP backend via HTTP requests to fetch search results.

Unit Testing: Components: Manual tests to verify search accuracy.
Performance: Testing does not significantly impact performance but lacks the comprehensiveness of automated testing frameworks.

Test Scenarios: The tests cover various scenarios, including matching file and directory names and ensuring correct recursive search behavior.

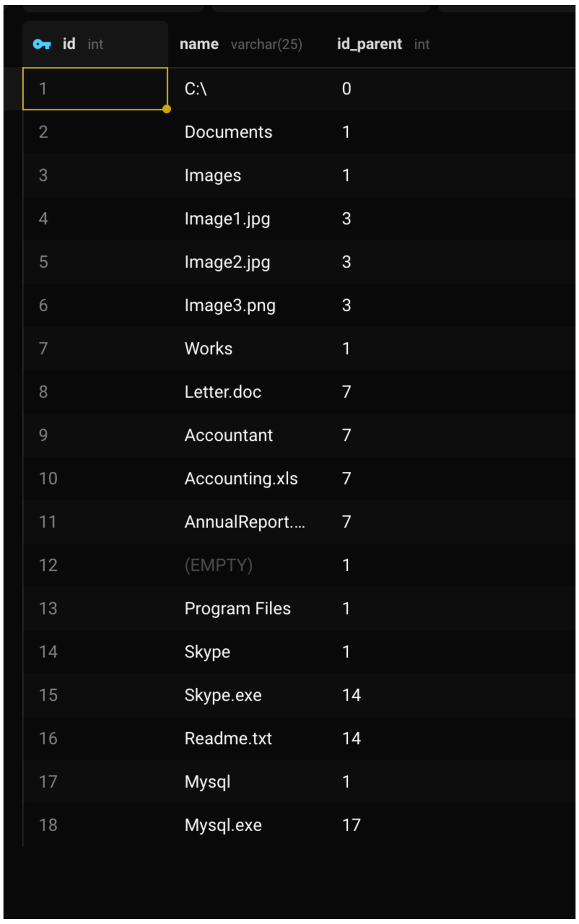
Performance: The recursive search is the main performance concern. For large file systems, this could lead to a stack overflow or increased processing time. Optimization strategies like caching search results or limiting recursion depth could be employed for improvement.

Database Design

To store the given file system, a simple relational database design can be used. Here's a proposed schema:

Table: files
id (Primary Key, Integer, Auto-increment)
name (Varchar)
parent_id (Integer, Foreign Key to files.id, NULL for root)

This design uses a single table to store both files and directories. The parent_id field establishes the parent-child relationship, with **NULL** representing the root level.



id int	name varchar(25)	id_parent int
1	C:\	0
2	Documents	1
3	Images	1
4	Image1.jpg	3
5	Image2.jpg	3
6	Image3.png	3
7	Works	1
8	Letter.doc	7
9	Accountant	7
10	Accounting.xls	7
11	AnnualReport....	7
12	(EMPTY)	1
13	Program Files	1
14	Skype	1
15	Skype.exe	14
16	Readme.txt	14
17	Mysql	1
18	Mysql.exe	17

Text File Manipulation

- Create a Text File Representing the File System Structure: A text file will be created to mirror the provided file system.
- Develop Code to Read and Insert the File System into the Database: A PHP script will be written to read the text file and insert its contents into the database. This script will parse the text file and use recursive functions to maintain the hierarchy.

Implementation Steps for Text File Manipulation

Implementation Steps for Text File Manipulation Create a Text File:
Create a text file named file_system.txt with the content, mirroring the file system structure.

PHP Code to Read and Insert into the Database:
This script will parse file_system.txt and insert each entry into the database. For simplicity, we'll use PDO for database interaction. The functions is implemented on ElmentService->loadFile()

System Implementation

To implement a feature that recursively searches for a keyword in the file system structure we've designed, we can prepared the ElementInterface and ElementServiceInterface classes to include a search method. This method will recursively search through directories and files to find matches with the given keyword.

File System Representation
Element: Represents a file or Directory element, extends Element Interface.
ElementInterface: Interface for add and update file element strategies.
Strategy Pattern for File Parsing and Database Insertion
LoadFile: for parsing the text file representation.
ElementService: Class for handling database insertion or searching.
ElementServiceInterface: Interface for handling database file element strategies.
Implement the Web services

Search Functionality:
SearchInterface An interface for the search strategy.
Search: A concrete strategy for searching by name.
SearchService: A service encapsulating the logic to search within the file system.
REST API:
Implemented in homeController.php, which processes GET requests and calls the search service.

The API accepts a keyword as a query parameter and returns search results in JSON format.
Unit Testing:
Manual tests were written to check the functionality of the file system search.
Tests were designed to confirm the accuracy of search results for given input keywords.

To create a REST API using native PHP for keyword search in the file system, This API will have a single endpoint that accepts a GET request with a keyword as a query parameter and returns the search results in JSON format to frontend. HomeController.php on Controllers folders. This file will be the entry point of our API.

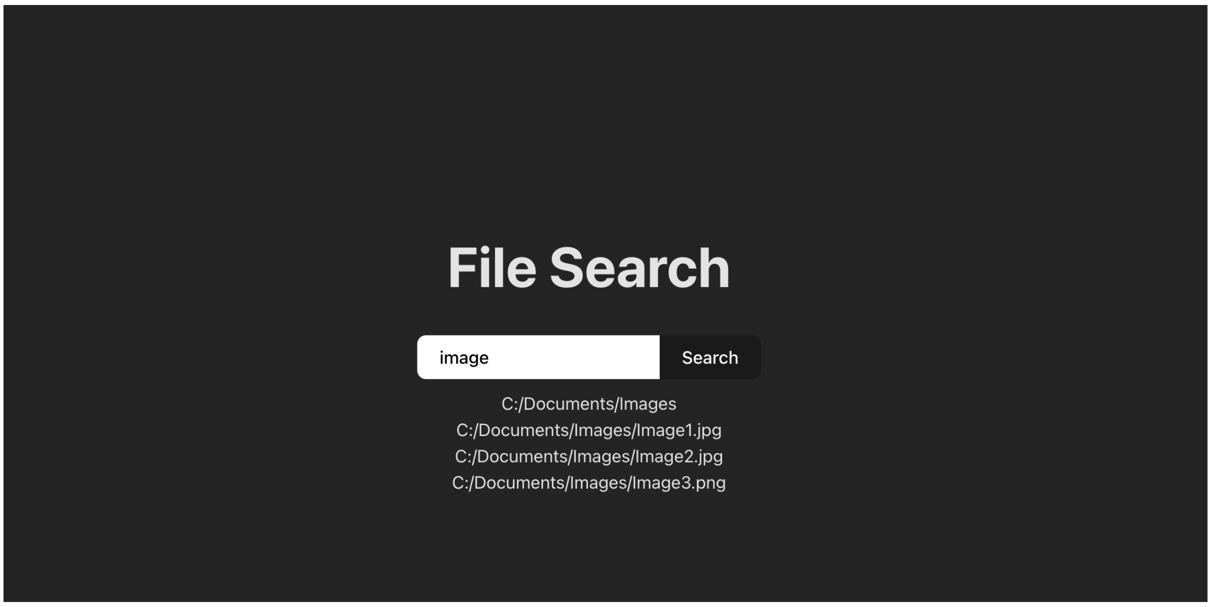
System Implement

Build and Run the Docker Container
Dockerfile for running a PHP application is a straightforward process. Docker provides a convenient and consistent environment for development, testing, and deployment. I will guide you through creating a basic Dockerfile for running your PHP application which includes the file system search functionality.

Frontend Implementation

The React application, set up with Vite, offers a simple and user-friendly interface. It includes an input field for search terms and displays the search results returned from the PHP backend. The application communicates with the backend via a REST API, parsing and presenting the JSON data received.

Navigate to **http://localhost:3000** in your web browser. Enter 'image' into the input box and click search. The app should display the results from your PHP backend, similar to:



```
C:/Documents/Images
C:/Documents/Images/Image1.jpg
C:/Documents/Images/Image2.jpg
C:/Documents/Images/Image3.png
```

Key Components

Search.js: The React component that handles the search input and displays results.
App.js: The main React component that integrates Search.js.
package.json: Contains dependencies and scripts for building the React app.
Functionality

The React app sends search queries to the PHP back-end and displays the results.
Basic input validation is implemented to ensure user inputs are handled correctly.

Backend (PHP API)

In the search function on home models, you would execute the logic to search your file system. This is where you would integrate the Element Service classes and methods for searching the file system.

Key Components

homeController.php: A PHP script that processes search requests and returns results in JSON format.
Additional ElementService and Element classes for file system modeling and search logic.
Functionality

Handles HTTP GET requests, performs file system searches, and returns JSON responses.

Project Structure
Make sure your project has the following structure:

```
|-- /frontend     # React application
|   |-- /src      # React source files
|   |-- /public   # Public assets for React app
|   |-- package.json
|   `-- ...
|
|-- /backend      # PHP API# PHP script for file search
`-- Dockerfile    # Docker configuration for building the app
```

Security and Best Practices

CORS Policy: The backend's CORS policy must be carefully managed, especially when deploying the application in a production environment.

For testing, the system set the header like this:
 header('Access-Control-Allow-Origin: *');

Input Validation: Both the backend and frontend should include robust input validation to prevent security vulnerabilities. The search function will provided during the keyword input length is great than 2.

Planned System Testing
To create a unit test for the file search functionality in PHP, you'll need to use a testing framework like PHPUnit. For our file search functionality, Unit testing involves testing the ElementService and Element class of an application in isolation. it tests the method responsible for searching files and directories based on a given keyword.

Explanation
setUp() method: Initializes the file system structure for each test.
testSearchReturnsCorrectFiles(): Tests if the searchFileSystem method returns the correct files for a given keyword.
This test checks whether the search functionality works as expected for a predefined file system structure. You can add more test methods to cover different scenarios, such as case-insensitive searches, searches with no results, etc.

At the End, Performing unit testing without a testing framework like PHPUnit in PHP is uncommon and requires manually writing the setup, execution, and assertion checks. However, it's still possible to create a basic structure to mimic unit testing behavior.

Create a function to test the search function . The testSearchFileSystem function conducts the test by calling search and then checking if the output matches the expected results.

Error Handling: The current implementation is basic. It will add comprehensive error handling for a production environment.
Security Considerations: Always validate and sanitize input data to prevent security vulnerabilities, such as SQL injection or XSS attacks.
Performance: For extensive file systems, optimize the search logic and consider caching results for frequently searched keywords.

Extensibility and Reusability
Composite Pattern: The Model/Element.php with ElementInterface, Element classes use the Composite pattern to represent the hierarchical file system. This makes the structure easy to extend with new types of components.
Strategy Pattern: The ElementService interface allows for different search implementations. New Services can be added without modifying the existing classes.
Controller Method Pattern: The search method in ElementService classes follows the Model and Control pattern (Home Models), providing a flexible framework for searching that can be extended in subclasses.

This design allows for extending and modifying parts of the system on ElementService and ElementServiceInterface (like adding different types of parsers) without affecting other components, adhering to the principles of reusability and extendability.

Performance Considerations and Recommendations

Caching: Implementing caching mechanisms can significantly improve response times for frequently searched terms.
Optimizing Recursive Calls: Investigating non-recursive algorithms or implementing iterative approaches could enhance performance, especially for deep file hierarchies.
Load Balancing: For the REST API, deploying it in a load-balanced environment could help manage higher request volumes effectively.
Asynchronous Processing: For very large file systems, considering asynchronous search operations could prevent blocking and improve user experience.

Challenges and Considerations

Performance: The recursive search can be intensive for large file systems. Optimization may be required for scalability.
Error Handling and Validation: The current implementation is basic and would need enhancements for robust error handling and input validation in a production environment.
Security: Special attention should be given to security considerations, especially for the REST API, to prevent potential vulnerabilities.

Summary

The developed solution effectively combines a PHP backend with a React frontend to provide a functional and user-friendly file system search tool, all within a Dockerized environment. The REST API provides a simple and effective way to interface with the file system search functionality. The implementation, while basic, lays the foundation for a more comprehensive and robust system that could be scaled and enhanced for specific applications.