# System Requirements Specification Index

For

# File System Explorer

Version 1.0

IIHT Pvt. Ltd.
fullstack@iiht.com

# TABLE OF CONTENTS

# File System Explorer

## System Requirements Specification

## 1 PROJECT ABSTRACT

TechSolutions Inc. needs a simple file system explorer tool that can analyze directory structures and search for specific files. This assignment focuses on implementing recursive functions to traverse a simulated file system structure, where directories are represented as dictionaries and files as integers (representing file sizes in bytes).

## 2 BUSINESS REQUIREMENTS:

| | |
|---|---|
| Screen Name | Console input screen |
| Problem Statement | 1. System must navigate dictionary-based directory structures using recursive traversal.<br>2. Tool must locate files by name or extension.<br>3. System must generate basic file distribution reports by type and size. |

## 3 CONSTRAINTS

### 3.1 INPUT REQUIREMENTS

1. Directory Structure:
   - Directories must be represented as nested dictionaries.
   - Files must be represented as key-value pairs (filename: size in bytes).
2. File Naming:
   - File extensions must be standard (.txt, .jpg, .png, etc.).

- - File names must be case-insensitive during search operations.

## 3.2 Function Definition Requirements

1. Recursive Structure:
   - Each function must include proper base case handling.
   - Each function must implement recursive traversal of the simulated file system.
   - No external libraries should be used (only built-in Python functions).
1. Docstrings:
   - Each function must include a docstring describing:
     - Purpose of the function
     - Parameters and return values
     - Base and recursive case descriptions
3. Parameter Types:
   - Functions must accept simulated directory structures (dictionaries).
   - Search functions must accept search criteria as strings.

## 3.3 Operations Constraints

**1.** Directory Traversal:
   - `list_all_files` must recursively find all files in a directory structure.
   - Function must handle nested directories of arbitrary depth.

**2.** File Search:
   - `find_by_extension` must locate all files with specific extensions.
   - `find_by_name` must locate files matching a name pattern.

**3.** File Analysis:
   - `calculate_directory_size` must sum file sizes recursively.
   - `count_files_by_type` must count files by extension.
   - `find_largest_files` must identify the N largest files in the structure.

**4.** Function Call Composition:
   - At least one function must call another function and use its return value.
   - Example: `total_size = calculate_directory_size(list_all_files(directory))`

### 3.4 OUTPUT CONSTRAINTS

1. Display Format:

- File paths should be displayed as relative paths.
- File sizes must be formatted in human-readable format (KB, MB, GB).

2. Output Format:

- "== FILE SYSTEM EXPLORER =="
- "Directory Summary" showing total files and size
- "File Type Distribution" showing count by extension
- "Search Results" showing files matching criteria
- "Largest Files" showing top N files by size

## 4. TEMPLATE CODE STRUCTURE:

**1.** Directory Traversal Functions:

- `list_all_files(directory, file_system=None, path_prefix="")` - returns all files in the directory tree.
- `calculate_directory_size(directory, file_system=None)` - computes total size of all files.

**2.** File Search Functions:

- `find_by_extension(directory, extension, file_system=None, path_prefix="")` - finds files with specific extension.
- `find_by_name(directory, pattern, file_system=None, path_prefix="")` - finds files matching name pattern.

**3.** File Analysis Functions:

- `count_files_by_type(directory, file_system=None)` - counts files by extension.
- `find_largest_files(directory, n, file_system=None)` - finds N largest files.

**4.** Helper Functions:

- `create_sample_file_system()` - creates a sample file system structure for testing.
- `format_file_size(size_bytes)` - converts bytes to human-readable format.

**5.** Main Program Function:

- `main()` - demonstrates all functions and produces formatted output.

## 5. EXECUTION STEPS TO FOLLOW:

1. Implement the required recursive functions according to specifications.
2. Test each function with the provided sample file system structure.
3. Format the output according to the requirements.
4. Ensure all recursive functions work correctly with different directory depths.
5. Run the main function to demonstrate all functionality.