UNIX-like File Search System Design – Interview Prep Deck

Q: What are the functional requirements of a UNIX-like file search system?

A: 1. Search by name
2. Search by path
3. Search by type (file/dir)
4. Search by metadata (size, date, etc.)
5. Recursive and non-recursive search
6. Wildcard and regex support
7. Logical operators (AND, OR)
8. Result output options
9. CLI/API interface
10. Permission handling

Q: How can search filters be logically combined in a UNIX-like file search system?

A: Using logical operators like AND/OR. E.g., an AndFilter can take multiple filters and return true only if all match.

Q: What are the non-functional requirements of a UNIX-like file search system?

A: 1. Performance
2. Scalability
3. Extensibility
4. Reliability
5. Maintainability
6. Testability
7. Portability
8. Concurrency
9. Security

Q: What does the FileSystemNode abstract class represent?

A: It represents a common interface for both files and directories with attributes like name, path, size, createdAt, etc.

Q: What is the purpose of the DirectoryNode class?

A: It extends FileSystemNode and contains children (files or directories). Used to represent directory structures.

Q: What is the role of the SearchFilter interface?

A: It provides a strategy interface for filtering FileSystemNode objects based on custom conditions.

Q: Give examples of classes implementing the SearchFilter interface.

A: 1. NameFilter
2. SizeFilter
3. ExtensionFilter
4. ModifiedDateFilter
5. TypeFilter
6. AndFilter/OrFilter

Q: How does the SearchEngine class work?

A: It performs DFS/BFS on the file system starting at a root node, applies a filter on each node, and collects matching results.

Q: What design patterns are used in this file search system?

A: 1. Strategy (SearchFilter)
2. Composite (File/Directory Nodes)
3. Decorator or Composite (Logical filters)
4. Possibly Singleton/Factory for filter creation