Technical Specification: Find-It

Rust-Based File Explorer

WanderRust Project - EPITA

Date: May 27, 2025

Author: Jean Philippe ZANGA.

Technical Specification: Find-It

General Information

- **Project Name**: Find-It
- **Type**: Graphical File Explorer
- **Purpose**: Provide a lightweight, fast, and secure tool for navigating, managing, and searching files on Windows, macOS, and Linux.
- **Team**: WanderRust (Jean Philippe Z., Clémence D., Milan M-L)
- **Context**: Academic project developed at EPITA
- **Primary Language**: Rust
- **Version**: 1.0 (based on the code state as of May 27, 2025)

Technologies Used

- **Language**: Rust (chosen for its performance, memory safety, and portability).
- **Main Libraries**:
  - `iced`: Framework for the graphical user interface, providing reactive widgets and efficient rendering.
  - `walkdir`: Recursive directory traversal.
  - `strsim`: Levenshtein distance calculation for fuzzy search.
  - `std::fs`: File and directory management (read, write, delete).
- **Data Structures**:
  - `HashMap<String, Vec<PathBuf»`: File indexing for search.
  - `HashMap<String, (Vec<SearchResult>, Instant)>`: Cache for search results.
  - `HashSet<PathBuf>`: Elimination of duplicates in search results.
  - `Vec<FileEntry>`: List of displayed files.
- **Supported Platforms**: Windows, macOS, Linux (via platform-specific commands like `cmd /C start`, `open`, `xdg-open`).

Main Features

1. **Directory Navigation**:
   - Browse directories using a tree structure and file list.
   - Support for system locations (drives, user directories like "Documents", "Desktop").
   - Navigate to parent directory via a dedicated button.
2. **File Search**:
   - Search bar with dynamic input and instant results.
   - Fuzzy search based on Levenshtein distance, tolerant to typos.
   - Prioritization of important directories ("Documents", "Downloads") with a score bonus.
   - Search result caching (validity: 60 seconds) to avoid recalculations.
   - Optimized indexing with `HashMap` for fast search.

3. **File Management**:
    - Create files and directories via an input field.
    - Delete with confirmation to prevent errors.
    - Copy, cut, and paste files/directories (recursive support for directories).
4. **File Preview**:
    - Preview for images (`jpg`, `png`, etc.), text (`txt`, `md`, etc.), and metadata for PDFs/other files.
    - Display metadata (name, size, modification date, type).
5. **Hidden Files Display**:
    - Button to toggle the display of hidden files/directories (starting with a dot).
    - Update of index and views upon activation.
6. **File Opening**:
    - Open files with the system's default application (`start` on Windows, `open` on macOS, `xdg-open` on Linux).

Software Architecture

- **Model**: `iced` application based on the ELM pattern (model-state-message).
- **Main Components**:
    - `FileExplorer`: Main structure holding the state (current path, file list, search results, etc.).
    - `Message`: Enumeration of user events (directory change, file selection, search, etc.).
    - `update`: Handles state transitions in response to messages.
    - `view`: Generates the graphical interface (navigation bar, file list, details, search results).
- **Key Functions**:

– `load`$_{f}iles : Loadsfilesfromadirectorywithfilteringforhiddenfiles.$

Optimizations

- **Search**:
    - Indexing with `HashMap` for fast access (O(1) on average).
    - Caching of results to reduce recalculations.

– Filtering of ignored directories (`/.git/`, `node`$_{m}odules/, etc.$).

- **Display**:
    - Limiting search results to 100 to avoid overload.
    - Asynchronous loading of files and previews via `Command`.
- **Resources**:
    - Efficient memory management thanks to Rust.
    - Prevention of infinite loops during directory traversal with `walkdir`.

Code Example

Here is an excerpt from the search handling in `search`$_{files}$:

```
for term in search_terms {
    for (indexed_term, paths) in &self.file_index {
        let distance = levenshtein(term, indexed_term) as f64;
        let max_len = term.len().max(indexed_term.len()) as f64;
        let mut score = 1.0 - (distance / max_len);
        if indexed_term == term {
            score *= 1.5; // Bonus for exact match
        }
        // ... (score calculation and result addition)
    }
}
```

Listing 1: Search Handling

Constraints and Limitations

- **Icon Dependency**: Requires icon files (icons/folder.png, etc.) in the specified directory.
- **Search**: Limited to the current directory and its subdirectories (no global system indexing).
- **Preview**: Limited support for certain formats (e.g., no direct PDF rendering).
- **Permissions**: May fail on files/directories without access rights.

Comparison with Alternatives

- **macOS Finder**:
  - Advantage: Fast and integrated Spotlight search.
  - Find-It: More flexible fuzzy search, but limited to the current directory.
- **Linux Thunar**:
  - Advantage: Lightweight and customizable.
  - Find-It: Adds efficient search and preview features.
- **Windows File Explorer**:
  - Advantage: Broad compatibility.
  - Find-It: Faster and less resource-intensive.

Future Improvements

- Global system indexing for broader search.
- Support for previewing more formats (PDFs, videos).
- Add search filters (by type, date, size).
- Advanced handling of permissions and errors.
- Internationalization of the interface (multilingual).

Conclusion

Find-It is a modern file explorer built in Rust to deliver performance, security, and portability. Its features for navigation, optimized search, file management, and preview make it a competitive alternative to existing tools, with significant potential for future enhancements.