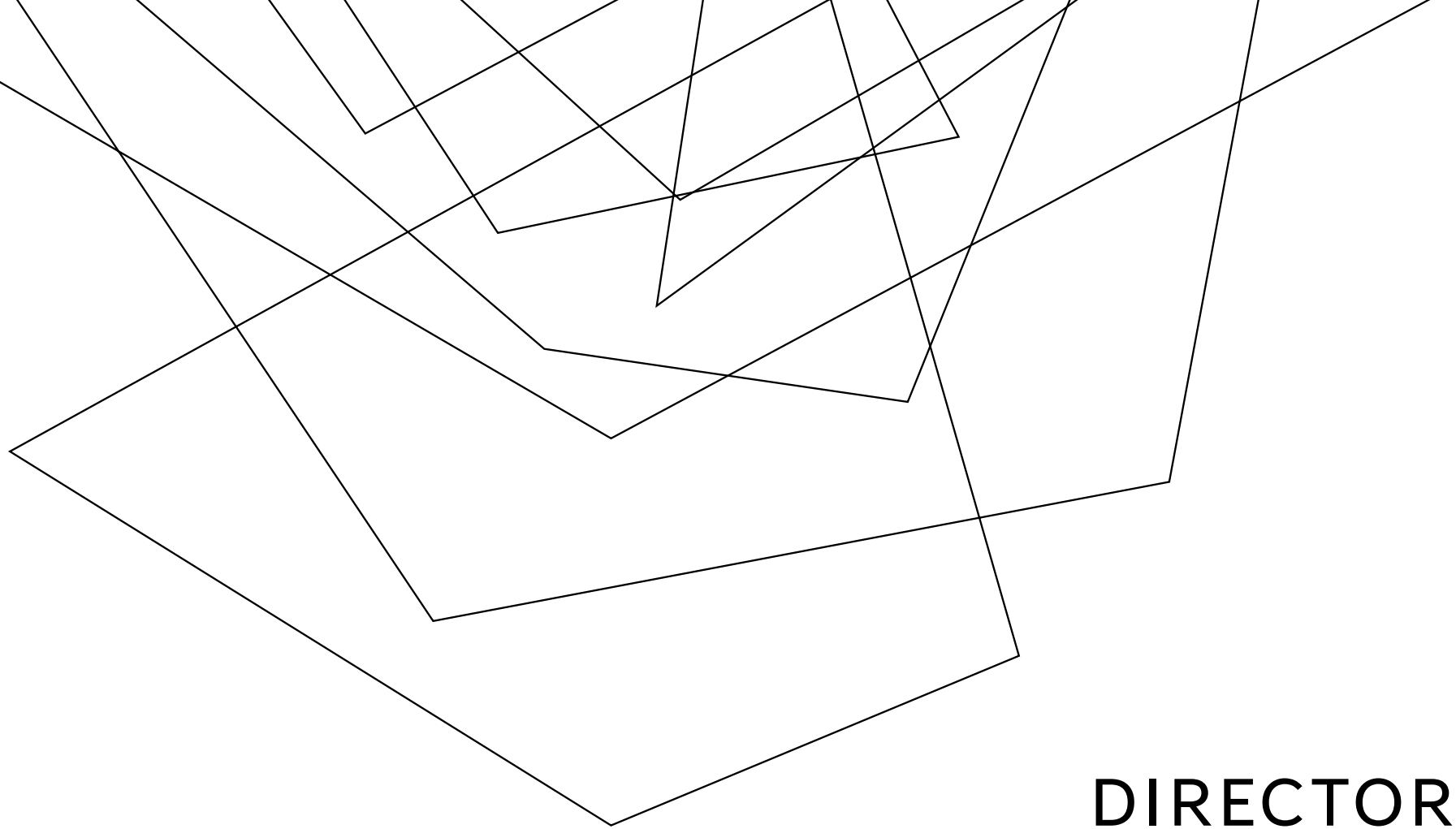
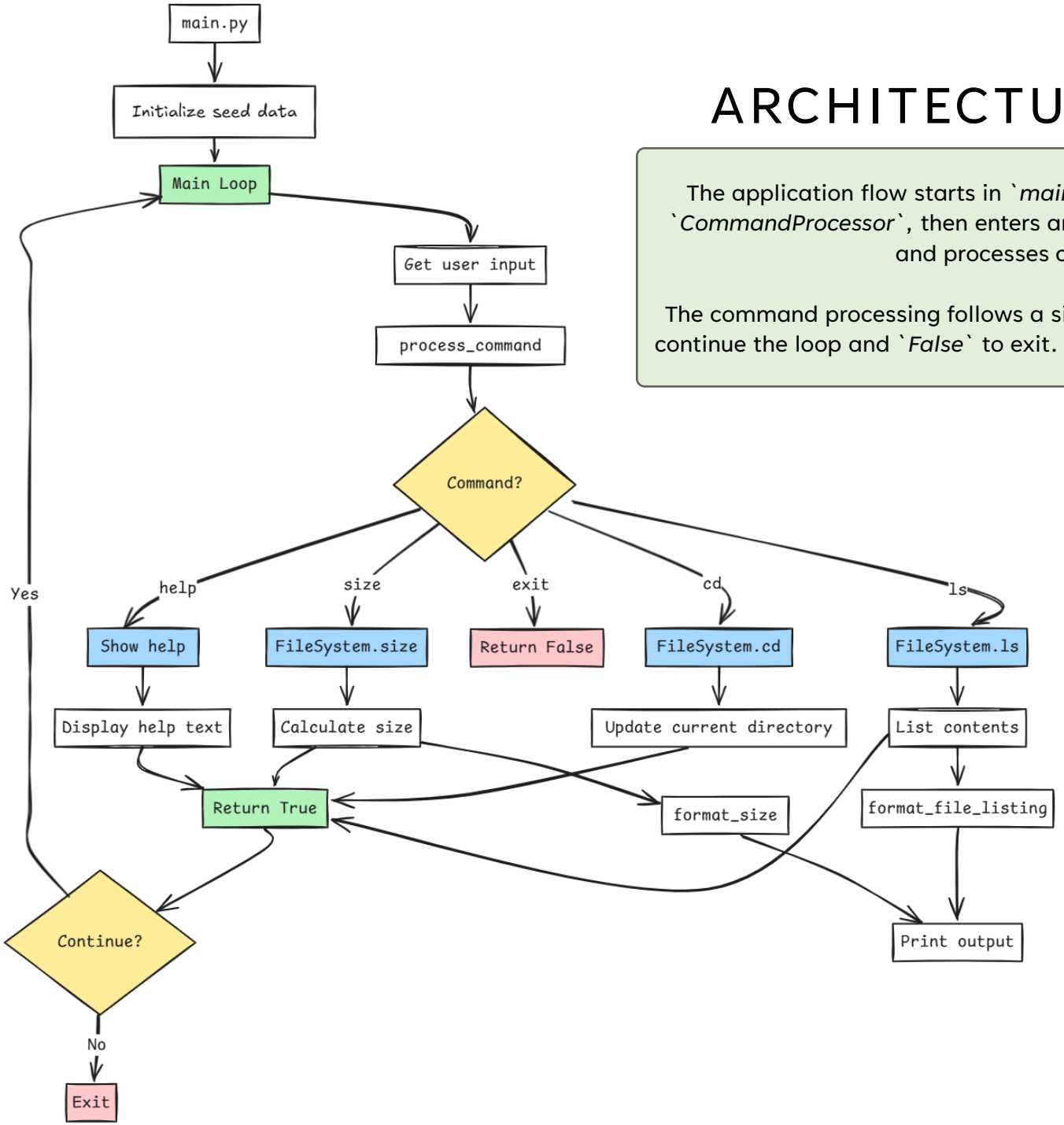


# DIRECTORY SIZE CALCULATOR APPLICATION

RISHABH KUMAR

Capgemini




# ARCHITECTURE FLOW DIAGRAM

The application flow starts in `*main.py*` where it initializes a `*FileSystem*` instance and `*CommandProcessor*`, then enters an infinite loop that continuously prompts for user input and processes commands through the processor.

The command processing follows a simple boolean return pattern where it returns `*True*` to continue the loop and `*False*` to exit. All commands (cd, ls, help and size) follow this pattern.

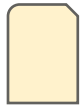
# APPROACH AND DESIGN

- Used layered architecture for different parts of the program - **CLI** handles user commands, **filesystem** manages the virtual directory and files, and **models** store the actual object data.
- Used **dependency injection pattern** where *CommandProcessor* receives *FileSystem* instance (Directory containing files and directories) for loose coupling between components.
- Used composition over inheritance with Directory class aggregating File and Directory objects in a **tree structure** for easy navigation and size calculation.

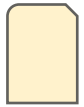
# PROJECT STRUCTURE

```
Capgemini/
├── main.py                 # Main application entry point
├── test.py                 # Comprehensive test suite
├── README.md
├── models/                 # Data models
│   ├── file.py             # File class
│   └── directory.py        # Directory class
├── filesystem/
│   ├── filesystem.py       # FileSystem class
│   └── initializer.py      # Seed data initializer
├── cli/
│   └── command_processor.py  # Command-line interface
└── utils/
    └── formatters.py       # Formatting utilities
```

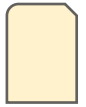
# KEY FILES AND FOLDERS



command_processor.py



filesystem.py



directory.py

- **CommandProcessor.py**: Handles all user input parsing and routes commands to the right filesystem operations, making it the central command hub.
- **Filesystem.py**: Manages the virtual filesystem navigation (cd, ls, size commands) and keeps track of current location, acting as the core filesystem controller.
- **Directory.py**: Defines the tree structure where folders can contain both files and other folders, using recursion to calculate total sizes by adding up all nested files and folders.

# KEY MODALS:

File Modal

```python
class File:
    def __init__(self, name: str, size: int):
        self.name = name
        self.size = size
```

Directory Modal

```python
class Directory:
    def __init__(self, name: str, parent: Optional['Directory'] = None):
        self.name = name
        self.parent = parent
        self.children: Dict[str, Union[File, 'Directory']] = {}
```

# PROCESS TO RUN & TEST

Once you have cloned the repository, go to the project directory and execute:

*PS: Test Data and Seed Data have been added in the code directly*

Running Application: `python main.py`

```
╔═══════════════════════════════════════════════════╗
║            Directory Size Calculator Application  ║
║                                                   ║
║  A file system simulator with cd, ls, and size commands ║
╚═══════════════════════════════════════════════════╝

Type 'help' for available commands.

/> ls
documents
downloads
photos

/> cd documents
/documents> ls
notes.txt (512.0 B)
projects
report.txt (1.0 KB)

/documents> size
7.5 KB

/documents> cd projects
/documents/projects> ls
project1.doc (2.0 KB)
project2.doc (4.0 KB)

/documents/projects> exit
Goodbye!
```

Running Tests: `python test.py`

```
PS C:\Users\risha\Documents\Project\Directory-Size-Calculator-Application> python test.py
============================================================
Directory Size Calculator - Test Suite
============================================================

Available tests:

  1. File Model
  2. Directory Model
  3. FileSystem Operations
  4. Command Processor
  5. Formatters
  6. Sample Filesystem
  7. Edge Cases

Please type one of the following:
  • A number (1-7) to run a specific test
  • 'all' to run all tests
  • 'exit' to quit the test suite

Enter your choice (number 1-7, 'all', or 'exit'): 2

==================== Directory Model ====================
Testing Directory model...
✓ Directory model tests passed
✓ Directory Model passed successfully!

Run another test? (y/n): N

Exiting test suite. Goodbye!
```

## Benefits of This Structure

1. **Separation of Concerns**: Each module has a specific responsibility
2. **Maintainability**: Easy to modify individual components
3. **Testability**: Each module can be tested independently
4. **Extensibility**: Easy to add new features or commands
5. **Type Hints**: Full type annotation support for better IDE experience
6. **Clean Code**: Minimal comments with self-documenting code

# Thank you!