LockedMe – Virtual Key for Repositories

The code for this project is hosted at
https://github.com/SwarajKokate/locked-me.

The project is developed by **Swaraj Kokate**.

1) Sprints planning:

Sprint Planning for 3 Weeks:

Sprint 1 (Week 1):

- Document the flow of the application and prepare a flow chart.
- List the core concepts and algorithms being used to complete this application.
- Code the welcome screen to display the application name and developer details.
- Implement the feature to accept user input and select one of the listed options.
- Implement the first option to return the current file names in ascending order.

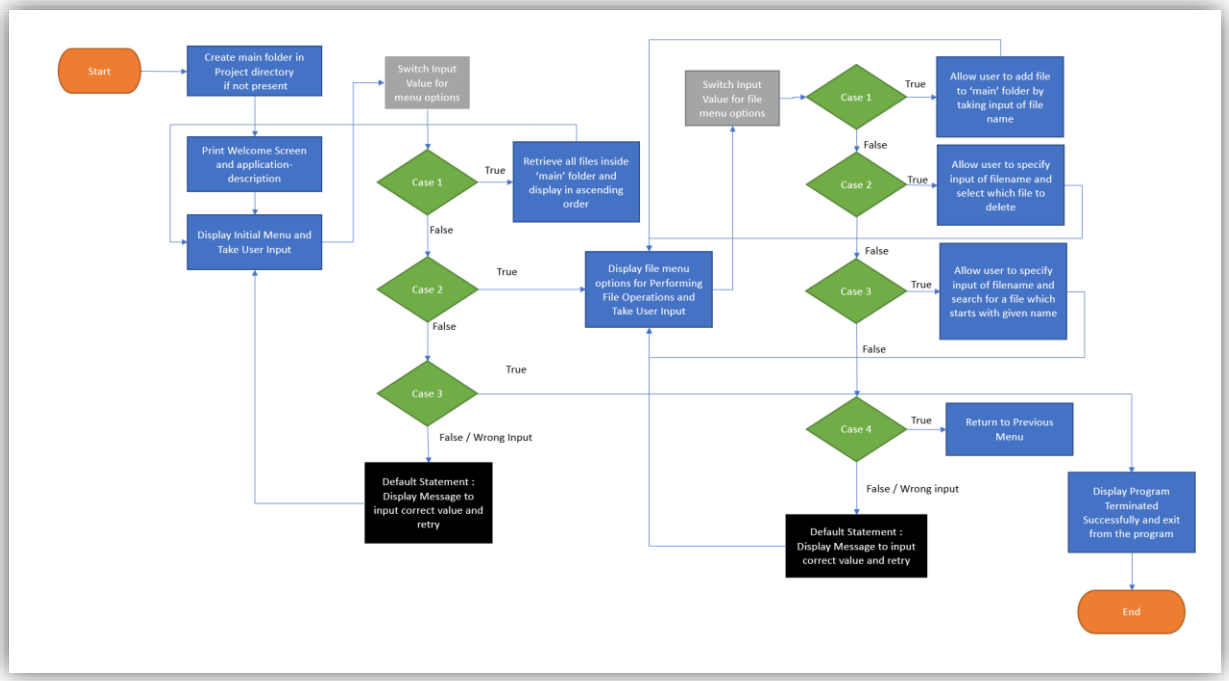
Sprint 2 (Week 2):

- Implement the second option to add a file to the existing directory list.
- Implement the second option to delete a user-specified file from the directory list.
- Implement the functionality to return a "File not found" message if the specified file is not in the directory list.
- Implement the option to navigate back to the main context.

Sprint 3 (Week 3):

- Implement the third option to search for a user-specified file in the main directory.
- Add case sensitivity to retrieve the correct file during the search.
- Display the result upon successful operation.
- Display the result upon unsuccessful operation.
- Implement the option to close the application.
- Optimize source code using appropriate concepts such as collections, and sorting techniques for increased performance.

2) Flow of the Application:



3) Core Concepts used:

The core concepts used in the application are collections, sorting.

4) Demonstrating the product capabilities, appearance, and user interactions

To demonstrate the product capabilities, below are the sub-sections configured to highlight appearance and user interactions for the project:

1. Create a project in Intellij named Locked-me
2. Write a program in java LockedMeMain.java in order to create entry-point for application
3. Writing a program in Java to display Menu options available for the user (**MenuOptions.java**)
4. Writing a program in Java to handle Menu options selected by user (**HandleMenuOptions.java**)

5 Writing a program in Java to perform the File operations as specified by user (**FileOperations.java**)
6 Pushing the code to GitHub repository

Step 1: Creating a new project in Intellij

- Open Eclipse
- Go to File -> New -> Project -> Java Project -> Next.
- Type in any project name and click on "Finish."

Step 2: Writing a program in Java for the entry point of the application (**LockedMeMain.java**)

```
import screens.MenuOptions;
import util.HandleMenuOptions;

/**
 * @author Swaraj Kokate
 */
public class LockedMeApplication {

    public static void main(String[] args) {
        MenuOptions.printWelcomeScreen();
        HandleMenuOptions.handleWelcomeScreenInput();
    }

}
```

Step 3: Writing a program in Java to display Menu options available for the user (**MenuOptions.java**)

- Select your project and go to src -> New -> Java Class.
- Enter **MenuOptions** in class name and click on "Finish."
- **MenuOptions** consists methods for -:

Step 3.1: Writing method to display Welcome Screen

```
public static void printWelcomeScreen() {
    String companyDetails =
String.format("**************************************************\n
```

```
"
            + "** Welcome to %s.com. \n" + "** This application was
developed by %s.\n"
            +
"**************************************************\n", "LockedMe",
"Swaraj Kokate");
    String appFunction = "You can use this application to :-\n"
            + "• Retrieve all file names in the \"main\" folder\n"
            + "• Search, add, or delete files in \"main\" folder.\n";
    System.out.println(companyDetails);
    System.out.println(appFunction);
}
```

Output:

```
**************************************************
** Welcome to LockedMe.com.
** This application was developed by Swaraj Kokate.
**************************************************

You can use this application to :-
• Retrieve all file names in the "main" folder
• Search, add, or delete files in "main" folder.
```

Step 3.2: Writing method to display Initial Menu

```
public static void displayMenu() {
    String menu = "\n\n****** Select any option number from below and
press Enter ******\n\n"
            + "1) Retrieve all files inside \"main\" folder\n" + "2)
Display menu for File operations\n"
            + "3) Exit program\n"
            + "Enter option : ";
    System.out.println(menu);

}
```

Output:

```
****** Select any option number from below and press Enter ******

1) Retrieve all files inside "main" folder
2) Display menu for File operations
3) Exit program
Enter option :
```

Step 3.3: Writing method to display Secondary Menu for File Operations

```
public static void displayFileMenuOptions() {
    String fileMenu = "\n\n****** Select any option number from below
and press Enter ******\n\n"
            + "1) Add a file to \"main\" folder\n" + "2) Delete a file
from \"main\" folder\n"
            + "3) Search for a file from \"main\" folder\n" + "4) Show
Previous Menu\n"
            + "Enter option : ";;

    System.out.println(fileMenu);
}
```

Output:

```
****** Select any option number from below and press Enter ******

1) Add a file to "main" folder
2) Delete a file from "main" folder
3) Search for a file from "main" folder
4) Show Previous Menu
Enter option :
```

Step 4: Writing a program in Java to handle Menu options selected by user (**HandleMenuOptions.java**)

- Select your project and go to src -> New -> Java Class.

- Enter **HandleMenuOptions** in class name and click on "Finish."
- **HandleMenuOptions** consists methods for -:

Step 4.1: Writing method to handle user input in initial Menu

```
public static void handleWelcomeScreenInput() {
    FileOperations fileOperations = new
FileOperationsImplementation();

    boolean executionStatus = true;
    Scanner scanner = new Scanner(System.in);
    do {
        try {
            MenuOptions.displayMenu();
            int input = scanner.nextInt();

            switch (input) {
                case 1:
                    output = fileOperations.displayFiles(true);
                    System.out.println(output);
                    break;
                case 2:
                    handleFileMenuOptions();
                    break;
                case 3:
                    System.out.println("Program exited
successfully.");
                    executionStatus = false;
                    scanner.close();
                    System.exit(0);
                    break;
                default:
                    System.out.println("Please select a valid option
from above.");
            }
        } catch (Exception e) {
            System.out.println(e.getClass().getName());
            handleWelcomeScreenInput();
        }
    } while (executionStatus == true);
}
```

Output:

```
****** Select any option number from below and press Enter ******

1) Retrieve all files inside "main" folder in ascending order
2) Display menu for File operations
3) Exit program
Enter option :
1
Files in main directory in ascending order :
[main\A.txt, main\C.txt, main\Z.txt]
```

Step 4.2: Writing method to handle user input in Secondary Menu for File Operations

```
public static void handleFileMenuOptions() {
    FileOperations fileOperations = new
FileOperationsImplementation();

    boolean executionStatus = true;
    Scanner scanner = new Scanner(System.in);
    do {
        try {
            MenuOptions.displayFileMenuOptions();
            int input = scanner.nextInt();

            switch (input) {
                case 1:
                    // File Add
                    System.out.println("Enter the name of the file to
be added to the \"main\" folder");
                    String fileToAdd = scanner.next();

                    output = fileOperations.addFile(fileToAdd);
                    System.out.println(output);

                    break;
                case 2:
                    // File delete
                    output = fileOperations.displayFiles(false);
                    System.out.println(output);

                    if(!output.equals("The folder is empty !!")) {
                        System.out.println("Enter the name of the file
to be deleted from \"main\" folder");
                        String fileToDelete = scanner.next();

                        output =
```

```
fileOperations.deleteFile(fileToDelete);
                        System.out.println(output);
                    }

                    break;
                case 3:
                    // File Search
                    System.out.println("Enter the name of the file to
be searched from \"main\" folder");
                    String fileName = scanner.next();

                    output = fileOperations.searchFile(fileName);
                    System.out.println(output);
                    break;
                case 4:
                    // Go to Previous menu
                    return;
                default:
                    System.out.println("Please select a valid option
from above.");
                }
            } catch (Exception e) {
                System.out.println(e.getClass().getName());
                handleFileMenuOptions();
            }
        } while (executionStatus == true);
}
```

Step 5: Writing a program in Java to perform the File operations as specified by user (**FileOperations.java**)

- Select your project and go to src -> New -> Java Class.
- Enter **FileOperations** in interface name.

- Select your project and go to src -> New -> Java Class.
- Enter **FileOperationsImplementation** in class name which extends FileOperations Interface. Here, factory-pattern is being implemented

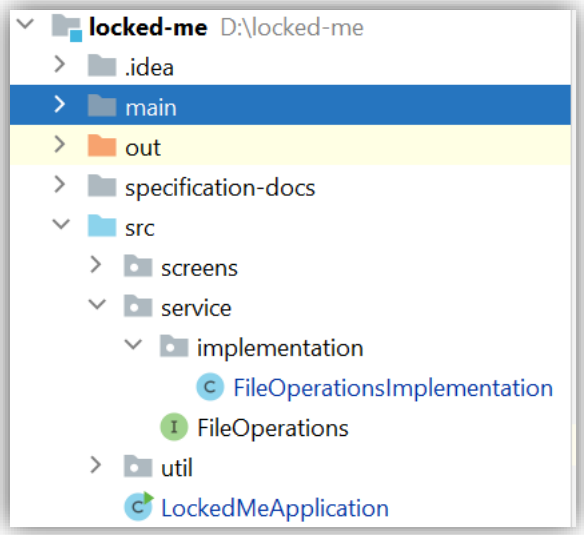
- **FileOperationsImplementation** consists methods for -:

Step 5.1: Writing method to create “main” folder in project if it’s not present

```
/**
 * A method used to create main directory
 * @return created or not
 */
private static boolean createMainFolderIfNotExist() {
```

```
        File file = new File("main");

        if(!file.exists()) {
            file.mkdirs();
            return true;
        }

        return false;
    }
```

Output:



Step 5.2: Writing method to display all files in "main" folder in ascending as well as in insertion-order:

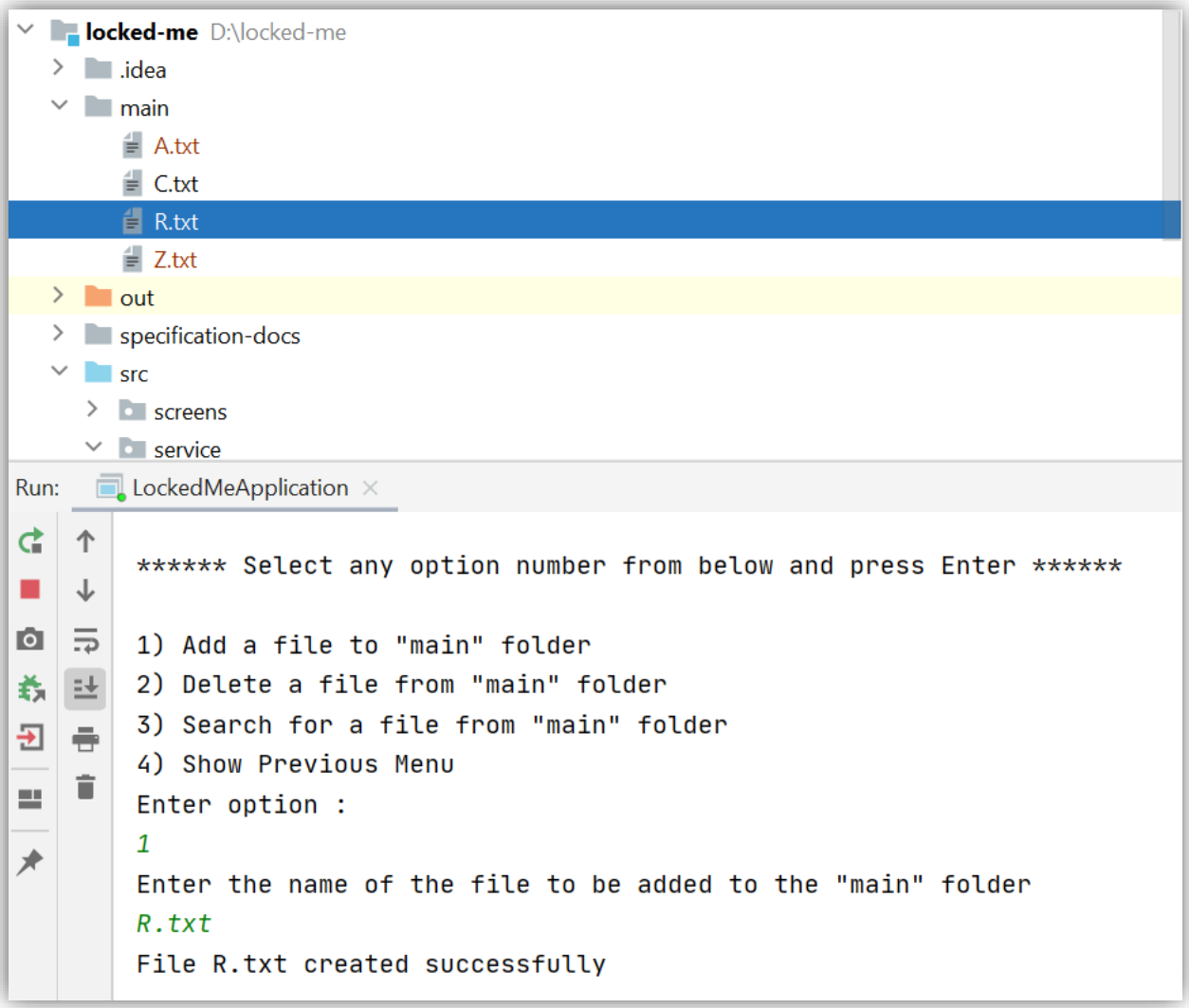
```
    /**
     * A method to display files in both sorted and unsorted order
     *
     * @param sortingEnabled
     * @return output
     */
    @Override
    public String displayFiles(boolean sortingEnabled) {
        createMainFolderIfNotExist();

        File file = new File("main");
        List<File> files = Arrays.asList(file.listFiles());

        if(!files.isEmpty() && files != null) {
            if (sortingEnabled) {
                Collections.sort(files);
                return "Files in main directory in ascending order :" +
    "\n" + files;
            } else {
```

```
                return "Files in main directory :" + "\n" + files;
            }
        } else {
            return "The folder is empty !!";
        }
    }
```

Step 5.3: Writing method to create a file as specified by user input.

```
    /**
     * A method to create user-specified file
     *
     * @param fileName
     * @return output
     */
    @Override
    public String addFile(String fileName) {
        boolean fileExists =
    FileOperationsImplementation.checkIfFileExists(Path.of("./main/" +
    fileName));

        try {
            if(fileExists) {
                return "File " + fileName + " already exists";
            } else {
                Files.createFile(Path.of("./main/" + fileName));
                return "File " + fileName + " created successfully";
            }
        } catch (IOException e) {
            e.printStackTrace();
        }

        return null;
    }
```

Output:

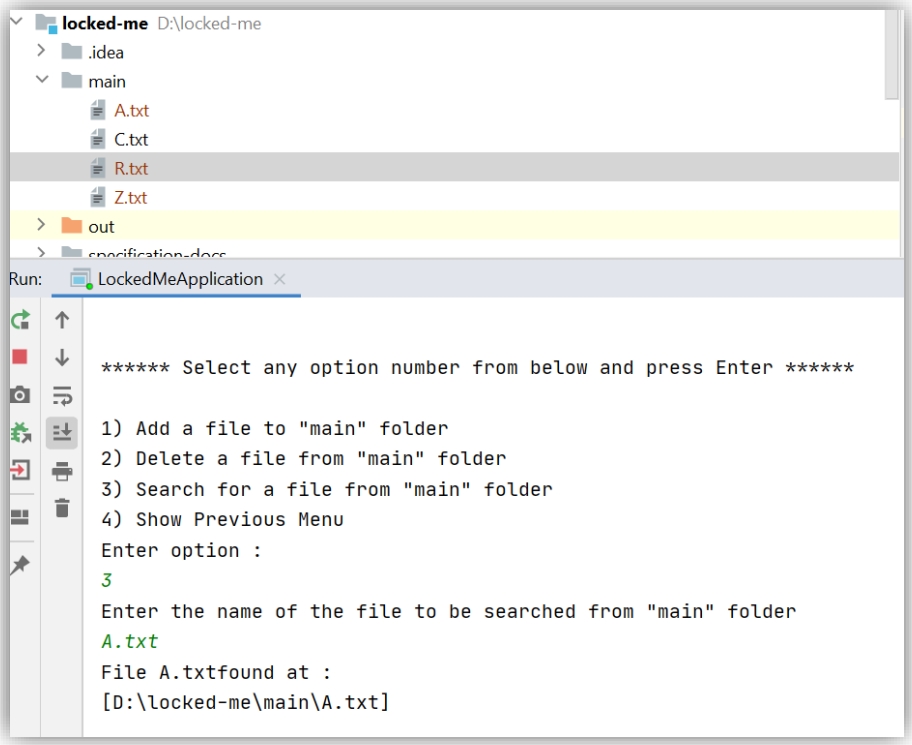


Step 5.4: Writing method to search for all files as specified by user input in "main" folder.

```
/**
 * A method to search user-specified file
 *
 * @param fileName
 * @return output
 */
@Override
public String searchFile(String fileName) {
    createMainFolderIfNotExist();

    List<String> fileLocations = new ArrayList<>();
```

```
        getAbsoluteFilePath("main", fileName, fileLocations);

        if(!fileLocations.isEmpty()) {
           return "File " + fileName + "found at :" + "\n" +
    fileLocations;
        } else {
            return "File with name " + fileName + " not found";
        }

    }

    /**
     * A method to retrieve absolute file paths for the files created
     *
     * @param path
     * @param fileName
     * @param fileLocations
     */
    private void getAbsoluteFilePath(String path, String fileName,
    List<String> fileLocations) {
        File file = new File(path);
        List<File> fileList = Arrays.asList(file.listFiles());

        if(!fileList.isEmpty() && fileList != null) {
            for(File indexFile : fileList) {
                if(indexFile.getName().equals(fileName)) {
                    fileLocations.add(indexFile.getAbsolutePath());
                }
            }
        }
    }
```

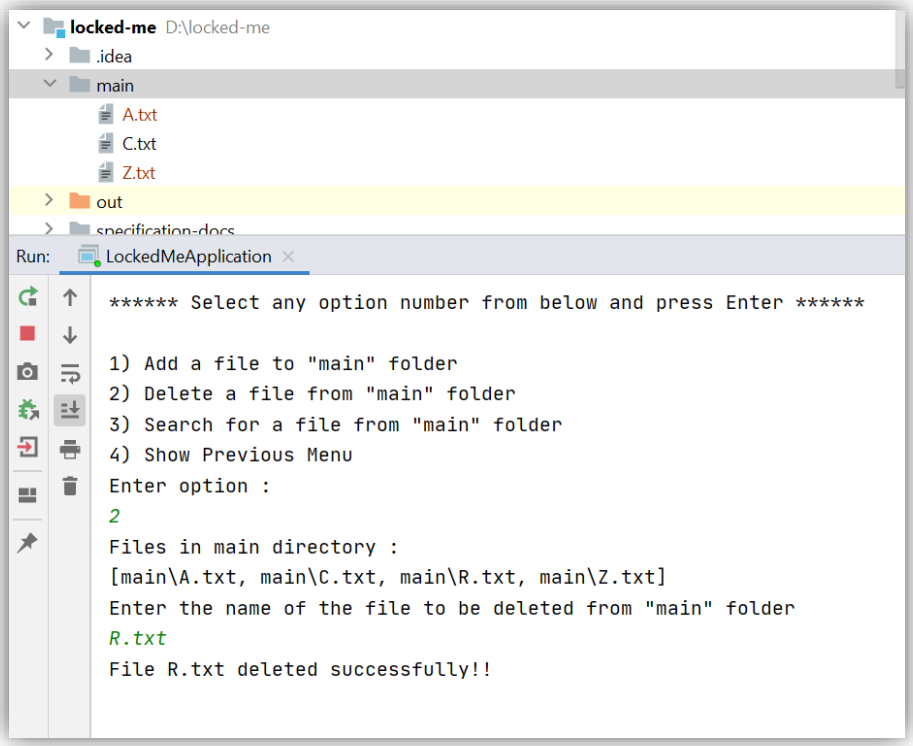
Output:



Step 5.5: Writing method to delete file specified by user input in "main" folder.

```
/**
 * A method to delete user-specified file
 *
 * @param fileName
 * @return output
 */
@Override
public String deleteFile(String fileName) {
    createMainFolderIfNotExist();

    try {
           boolean fileDeleted =
Files.deleteIfExists(Path.of("./main/" + fileName));
            return (fileDeleted) ? "File " + fileName + " deleted
successfully!!" : "File " + fileName + " does not exists";
    } catch (IOException e) {
        e.printStackTrace();
    }

    return null;
}
```

Output:




```
****** Select any option number from below and press Enter ******

1) Add a file to "main" folder
2) Delete a file from "main" folder
3) Search for a file from "main" folder
4) Show Previous Menu
Enter option :
4


****** Select any option number from below and press Enter ******

1) Retrieve all files inside "main" folder in ascending order
2) Display menu for File operations
3) Exit program
Enter option :
3
Program exited successfully.

Process finished with exit code 0
```

Step 6: Pushing the code to GitHub repository

- Open your command prompt and navigate to the folder where you have created your files.

  cd <folder path>

- Initialize repository using the following command:

  git init

- Add all the files to your git repository using the following command:

  git add .

- Commit the changes using the following command:

  git commit . -m <commit message>

- Push the files to the folder you initially created using the following command:

  git push -u origin main

5) Unique Selling Points of the Application

1. **Intuitive User Interface**: The application provides a user-friendly and intuitive interface, making it easy for users to navigate and interact with the features. The options are clearly presented, and the flow of the application is designed to enhance user experience.
2. **Efficient File Management**: The application offers efficient file management capabilities, allowing users to view, add, and delete files from the directory list.
3. **Quick File Search**: Users can quickly search for specific files within the main directory. The application supports case sensitivity, enabling users to retrieve the correct file easily. The search functionality helps users locate files promptly, saving time and effort.

4. **Source Code Optimization**: The application employs concepts like collections and sorting techniques for source code optimization, resulting in increased performance. It ensures efficient processing of file operations, enhancing the overall responsiveness and speed of the application.
5. **Developer Support**: The application provides comprehensive developer details, fostering transparency and trust. Users can easily access information about the application's development team, promoting a sense of credibility and accountability.

6) Conclusions:

1. In conclusion, the file management application offers a user-friendly interface and efficient file handling capabilities, making it a valuable tool for users seeking an organized and streamlined approach to managing their files.
2. With features like sorting techniques, case sensitivity options, and quick file search functionality, users can easily navigate through their files, add new ones, delete unwanted files, and locate specific files with ease.
3. Additionally, the application's source code optimization techniques contribute to increased performance, delivering swift and responsive file operations.
4. The flexibility and customization options empower users to tailor the application to their preferences, enhancing their file management experience.
5. With comprehensive developer support and a commitment to continuous improvement, the application ensures user satisfaction and adapts to meet evolving needs.
6. Overall, this file management application is a reliable and efficient solution for users looking to optimize their file organization and improve productivity.
7. Future enhancements can be included as :
    a) Adding a file at specific location
    b) Searching file based on various criteria such as Last Modified, Date Added, etc.