save

Program

Group Project - Fall 2024Coordinator(s) Dr. Taghreed

CPIT-251 Software Engineering (I)

12901SO

Student ID	2206591	IS -IT-CS
Student Name	Raya Mohammed Alyoubi	
Section	A2	

Total Obtained Marks 15out of

	Max	Obtained Marks for SO
1	3	

Saturday, November 30 2024Printed ~~by :~~ D

Faculty of Computing and Information Technology

Department of Information Technology

Information Management Program

CPIT-251

Student name	**ID**
Aseel Alnefaie	2206932
Layan Baasoor	2206325
Raya Alyoubi	2206591
Section: A2	

Table of Contents

Figures

Abstract

The proposed Information Management Program aims to streamline information access, and comments from employees to administrators. By addressing issues such as poor search functionality and outdated content, the application improves operational efficiency. Developed using the Extreme Programming (XP) methodology, the application emphasizes iterative releases, test-driven development, and collaborative coding. The system incorporates key features such as robust search functionality, employee comments, and admin feedback mechanisms, ensuring a user-centric experience.

Introduction

This report outlines the development process for our Information Management Program. The team consists of Aseel Alnefaie, Layan Baasoor, and Raya Alyoubi.

The Program aims to address the challenges employees face in accessing and managing information efficiently. By leveraging Extreme Programming (XP) principles, we collaboratively developed a system designed to simplify searching, sharing, and updating information.

The document provides a detailed overview of the problem statement, proposed solution, software engineering processes, test driven development, implementation, testing, and future improvements. Each team member's contributions are highlighted, reflecting our commitment to teamwork and effective task distribution throughout the project.

Problem Statement

Employees have trouble finding the information they need because the current system is hard to use, has a poor search function, and outdated content. This slows down their work and creates delays.

Proposed Solution

To fix these issues, we propose developing an program to help employees find information easily. It will have a better search feature, and a way to give feedback. This will save time and make work easier.

Approach (Software Engineering Process)

We are using the **Extreme Programming (XP)** methodology to implement this project. XP emphasizes iterative development, collaboration, and frequent releases, which will help us respond quickly to any changing requirements and feedback. The steps involved in our approach include:

1. **Setting Functional Requirements**: Defining specific requirements through user stories that outline the desired features and functionalities.
2. **Implementation**: Developing each functionality iteratively, testing each feature to ensure usability and performance.
3. **Test-first development:** An automated unit test framework is used to write tests for a new piece of functionality before that functionality itself is implemented.
4. **Refactoring** All developers are expected to refactor the code continuously as soon as possible code improvements are found. This keeps the code simple and maintainable**.**
5. **Pair programming**: Developers work in pairs, checking each other's work and providing support to always do a good job.
6. **Continuous integration**: As soon as the work on a task is complete, it is integrated into the whole system. After any such integration, all the unit tests in the system must pass.
7. **Releases**: Releases of the system are frequent and incrementally add functionality to the first release.

Requirements And Analysis

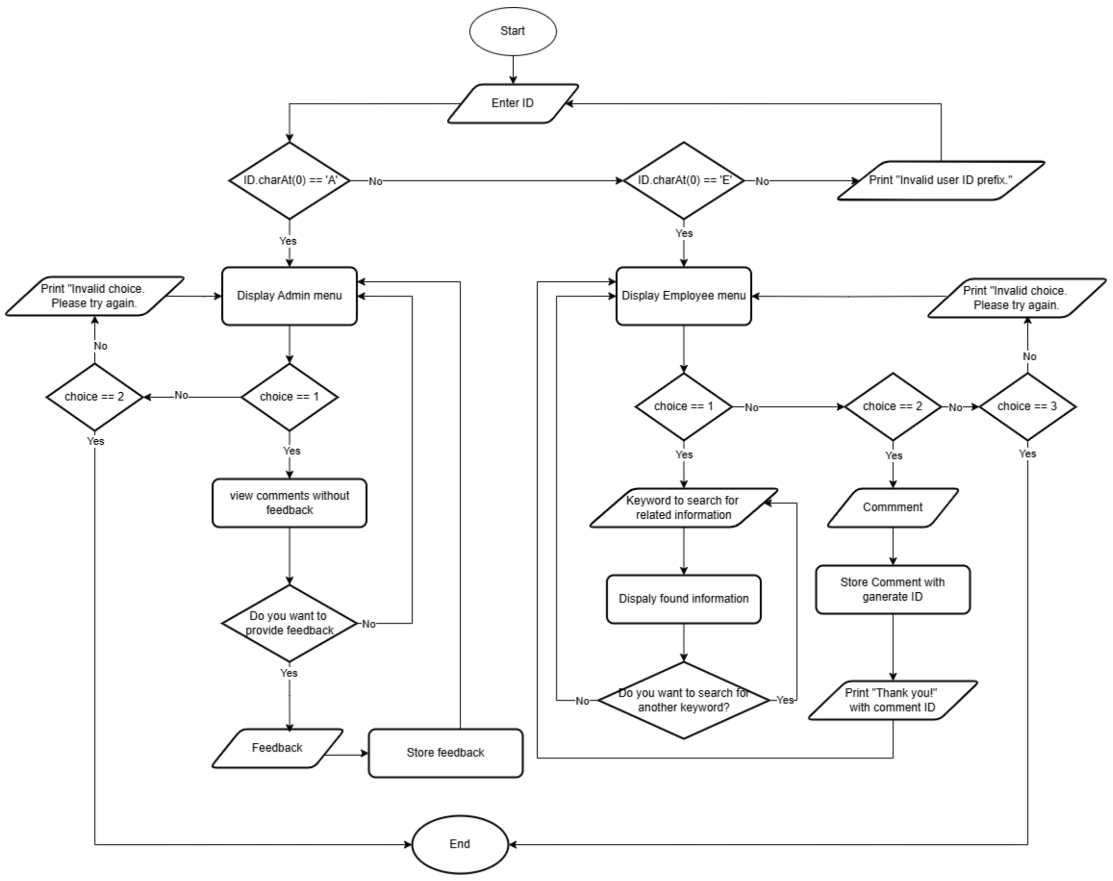

Start
Enter ID
ID.charAt(0) == 'A'
No
Yes
ID.charAt(0) == 'E'
Print "Invalid user ID prefix."
Display Admin menu
Print "Invalid choice. Please try again.
choice == 2
choice == 1
view comments without feedback
Do you want to provide feedback
Feedback
Store feedback
Display Employee menu
choice == 3
Keyword to search for related information
Dispaly found information
Do you want to search for another keyword?
Commment
Store Comment with ganerate ID
Print "Thank you!" with comment ID
End

Figure1 – Flow Chart Diagram

Functional Requirements

The following user stories guide the development of the application:

- Employee shall ask questions to view relevant information they need.

- Employee shall send comments on specific topics to contribute to discussions.

- Admin shall receive comments to monitor discussions and ensure the quality of content.

- Admin shall send feedback on employee comments to address concerns enhances communication.

- Admin shall update information in the system to ensure employees have access to current information.

- Admin shall alert employees about updates to keep everyone informed of important changes.

Use Case Diagram

This use case diagram shows how two types of users, Employee and Admin interact with the program. The Employee can do two things: ask a question to see information and add comments to the system. The Admin can see the comments added by employees and give feedback in response. The system helps both users do these tasks.

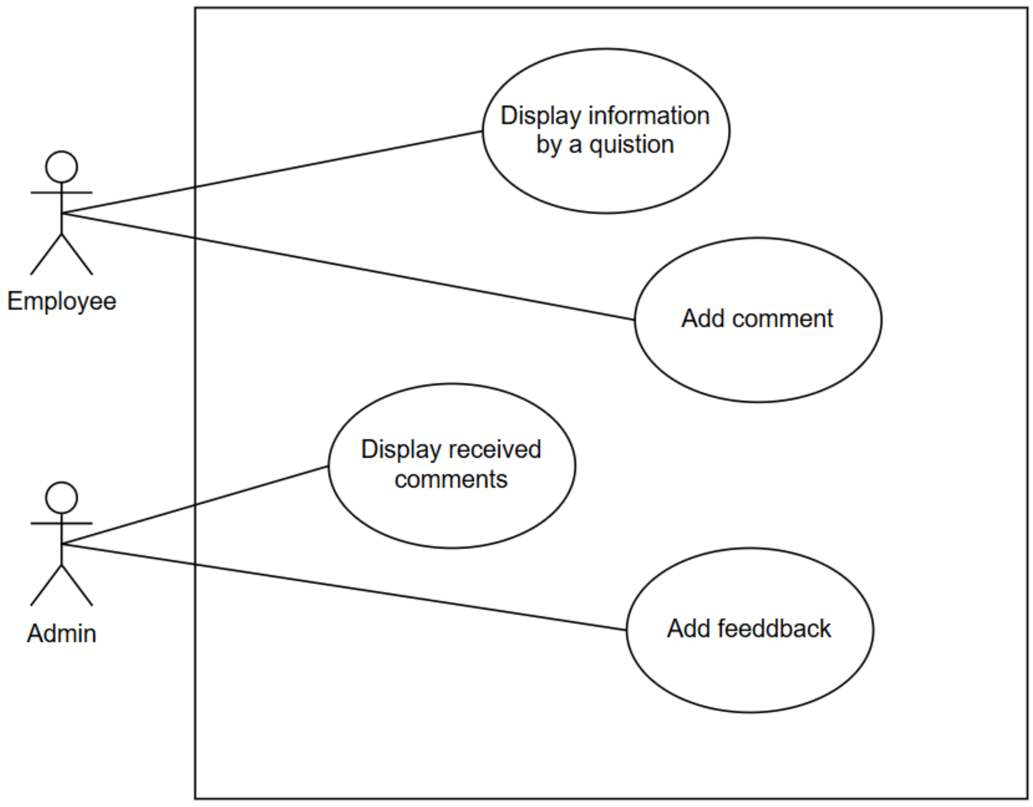


Figure 2 - Use Case Diagram

Sequence Diagram

The sequence diagrams show how the system handles key tasks. Employees can search for information, with valid inputs retrieving results from `Information.txt`, or send comments, which are saved in `comments.txt` with a confirmation. Admins can view comments; if none exist, a message is displayed, otherwise, comments are shown, and feedback is requested. These diagrams highlight smooth user interactions .and data processing

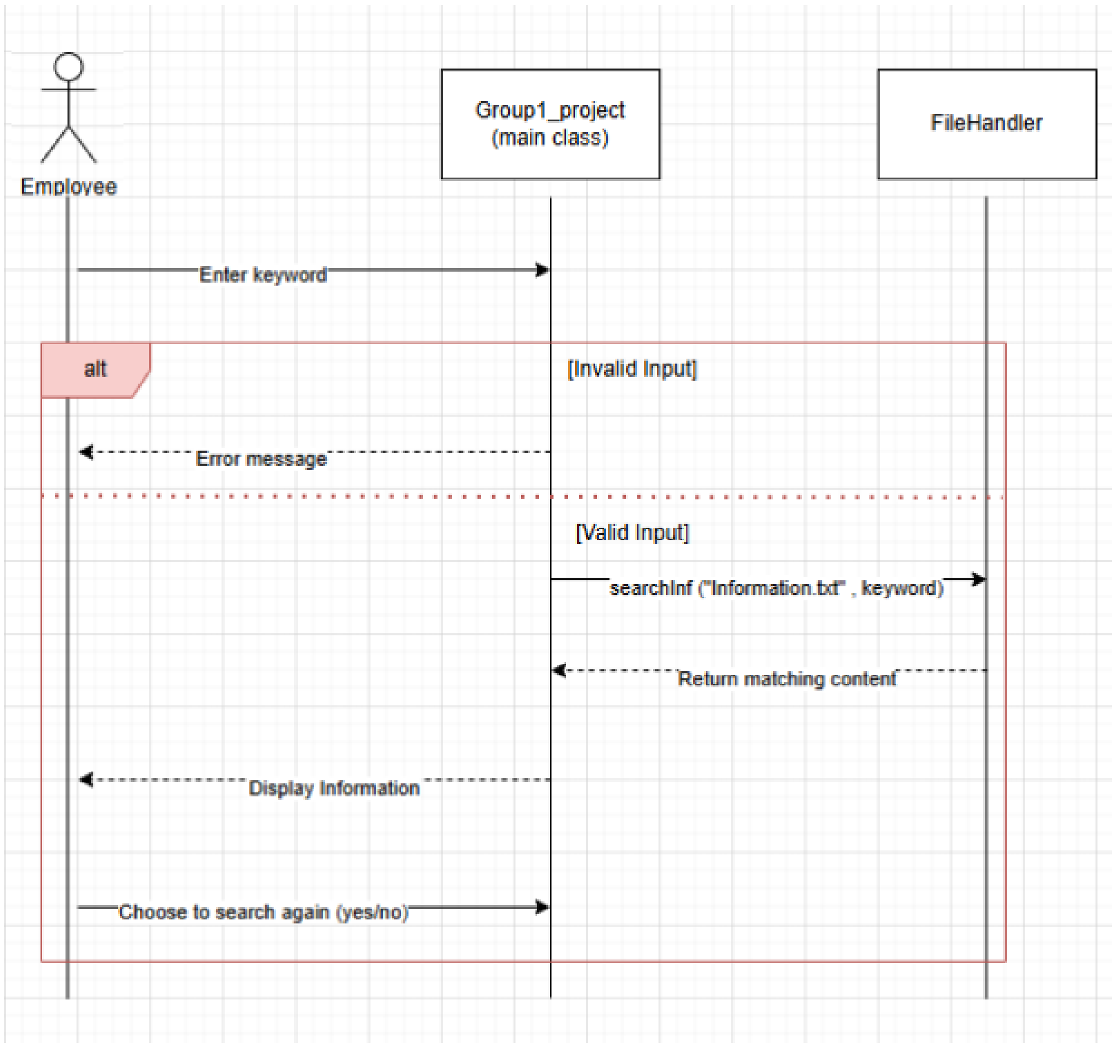


Figure3 - Sequence diagram of Search Information

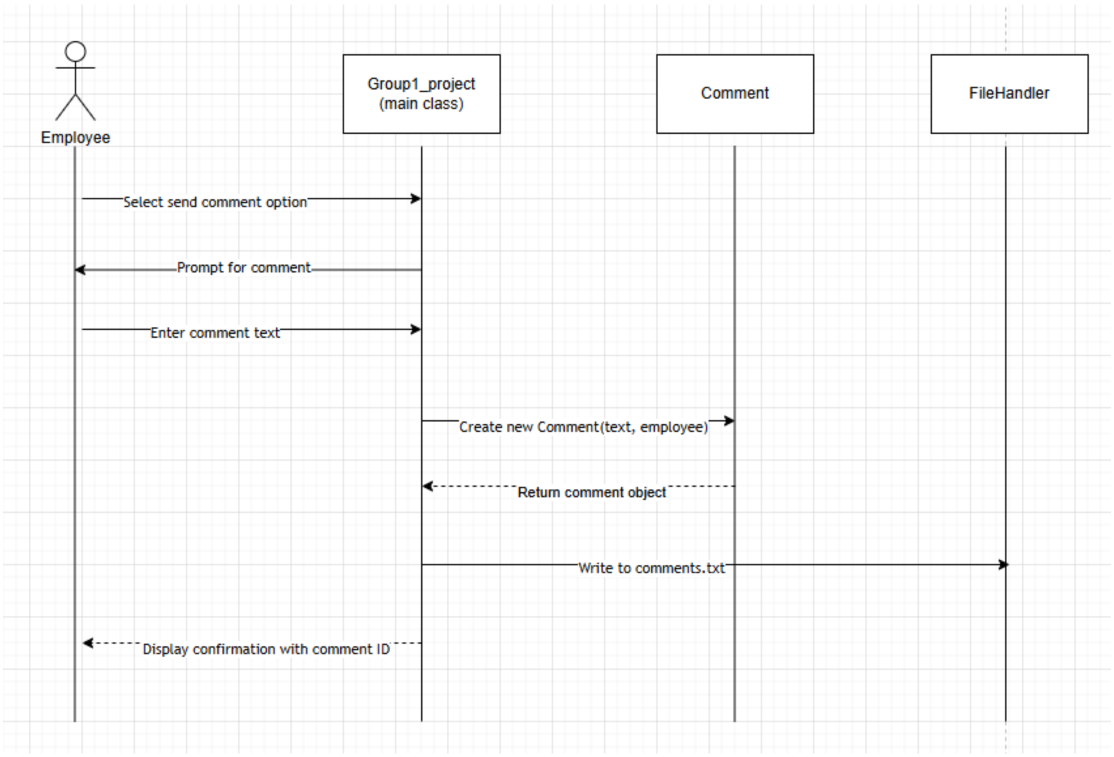


Figure4 - Sequence Diagram of Add Comment

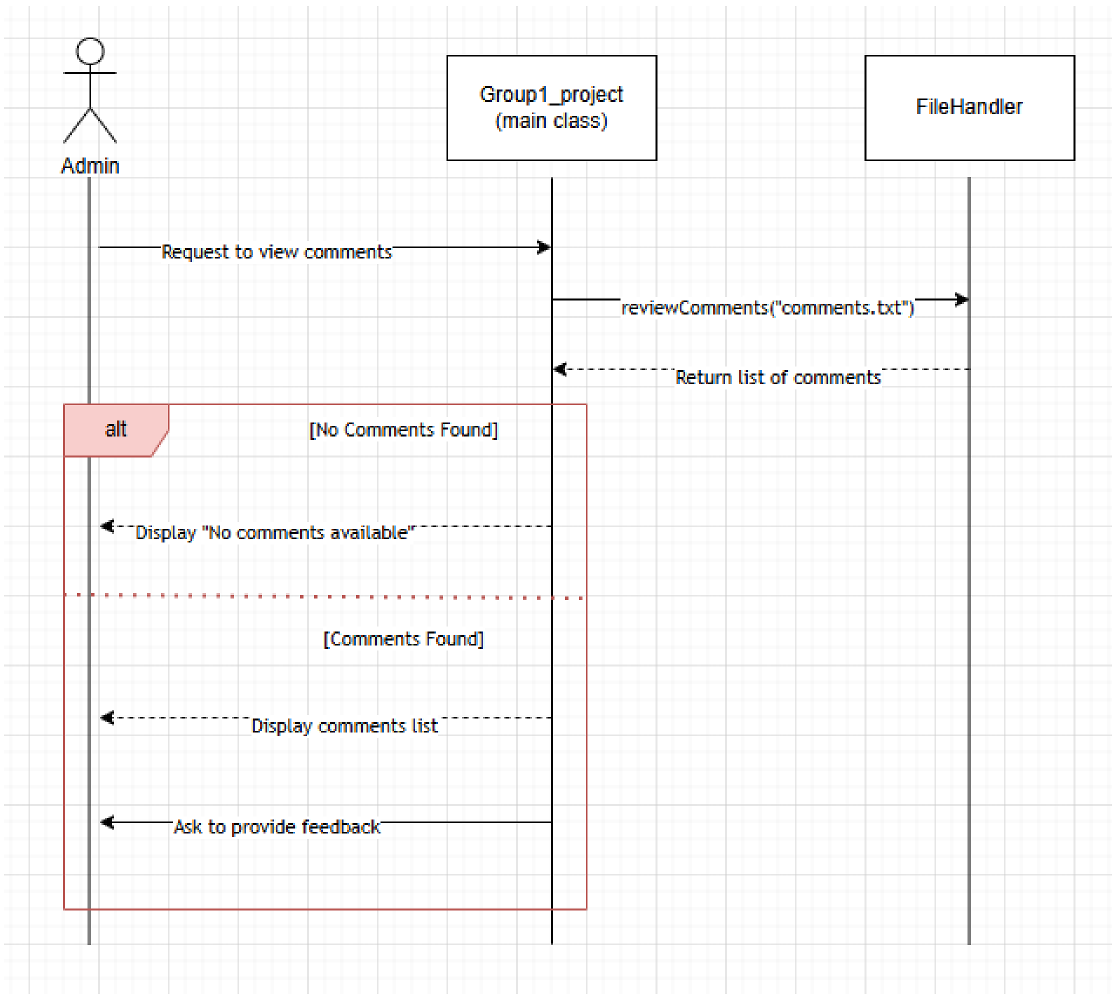


Figure5 - Sequence diagram of Review Comments

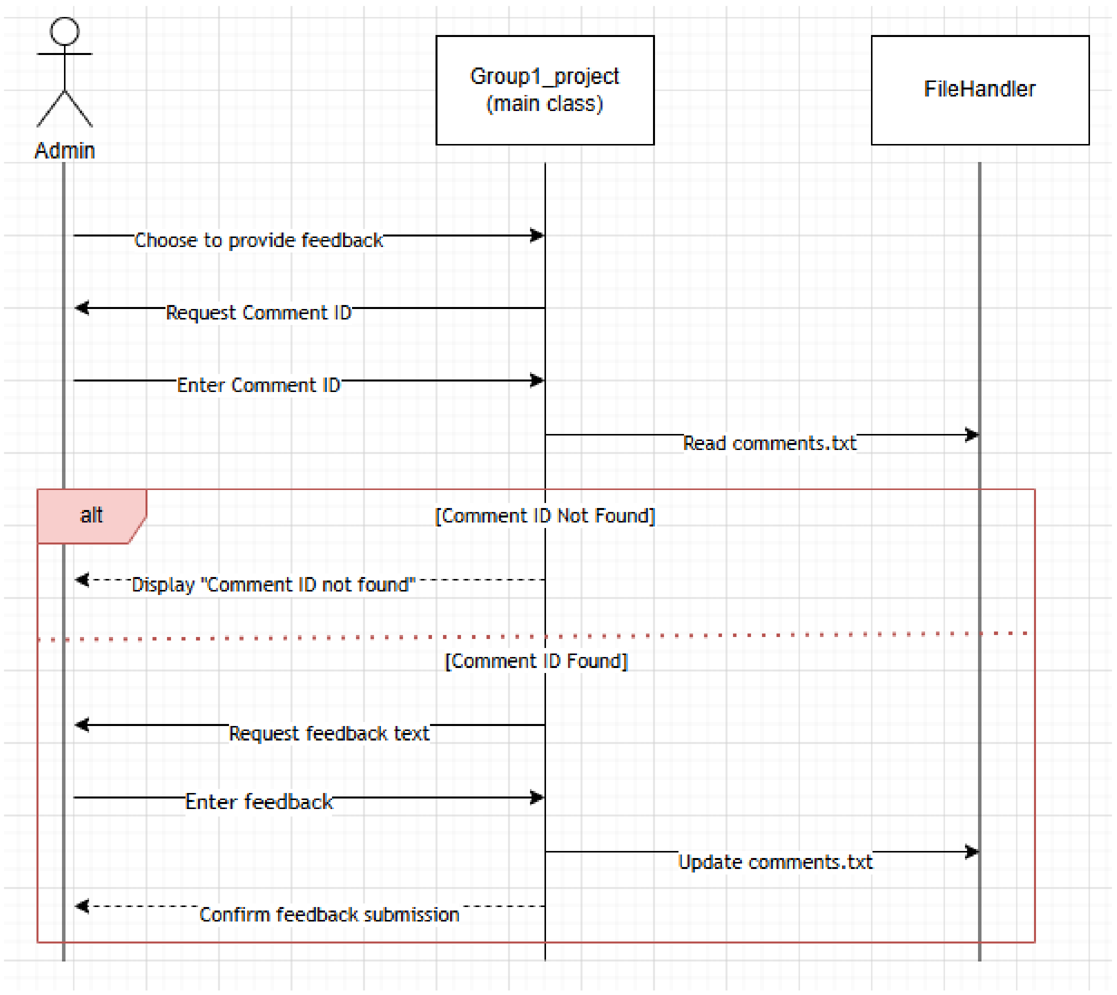
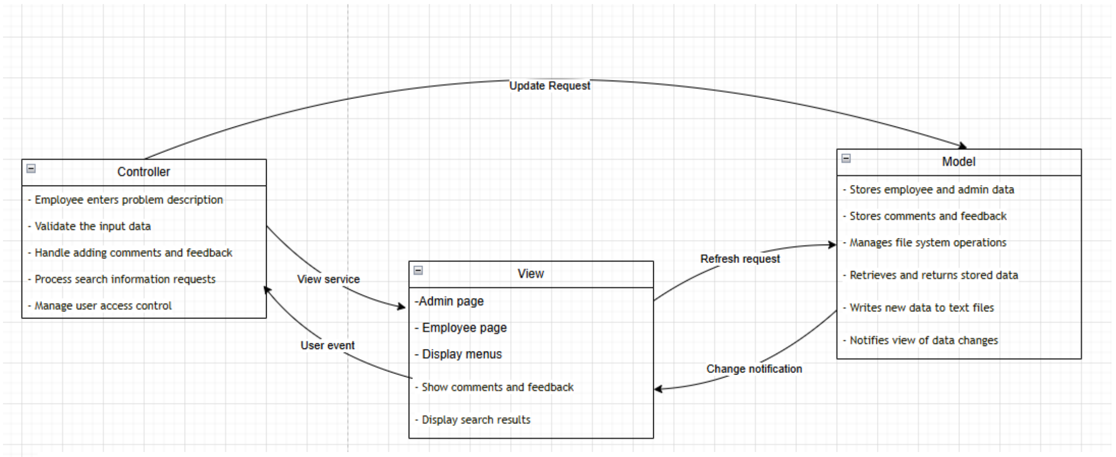


Figure 6 - provide feedback sequence diagram

Software Architecture

This diagram shows how the program follows the Model-View-Controller (MVC) structure to organize its functions. The **Controller** handles user input, such as when employees enter problem descriptions, add comments, or search for information, and it sends these requests to the **Model** or updates the **View**. The **View** is the user interface, displaying pages for employees and admins, showing menus, search results, comments, and feedback, and responding to user actions by sending them to the controller. The **Model** manages all the program data, including employee and admin information, comments, and feedback. It handles file operations, retrieves or updates stored data, and notifies the **View** when changes occur so it can refresh the displayed information. The components interact by sending requests and updates to keep the program running smoothly and ensure users can access the information they need.



Update Request
Controller
- Employee enters problem description
- Validate the input data
- Handle adding comments and feedback
- Process search information requests
- Manage user access control
View service
User event
View
-Admin page
- Employee page
- Display menus
- Show comments and feedback
- Display search results
Refresh request
Change notification
Model
- Stores employee and admin data
- Stores comments and feedback
- Manages file system operations
- Retrieves and returns stored data
- Writes new data to text files
- Notifies view of data changes

Figure 7 - software architecture

Class Diagram

The class diagram provides a detailed view of the system's data structures, covering classes such as Comment, Admin, Employee, and main class. It explains the relationships and interactions between these classes, ensuring a foundation for implementation.

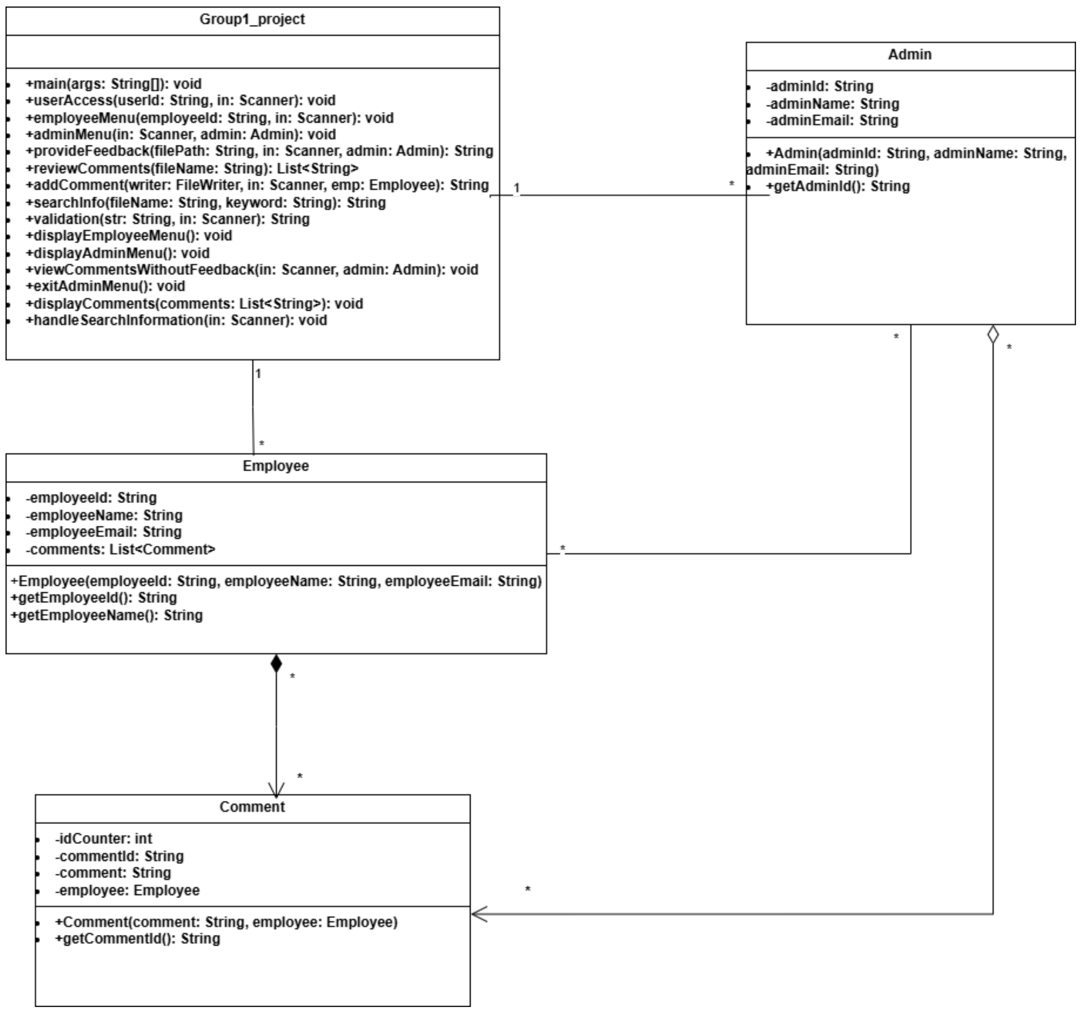

Group1_project
+main(args: String[]): void
+userAccess(userId: String, in: Scanner): void
+employeeMenu(employeeId: String, in: Scanner): void
+adminMenu(in: Scanner, admin: Admin): void
+provideFeedback(filePath: String, in: Scanner, admin: Admin): String
+reviewComments(fileName: String): List<String>
+addComment(writer: FileWriter, in: Scanner, emp: Employee): String
+searchInfo(fileName: String, keyword: String): String
+validation(str: String, in: Scanner): String
+displayEmployeeMenu(): void
+displayAdminMenu(): void
+viewCommentsWithoutFeedback(in: Scanner, admin: Admin): void
+exitAdminMenu(): void
+displayComments(comments: List<String>): void
+handleSearchInformation(in: Scanner): void
Admin
-adminId: String
-adminName: String
-adminEmail: String
+Admin(adminId: String, adminName: String, adminEmail: String)
+getAdminId(): String
Employee
-employeeId: String
-employeeName: String
-employeeEmail: String
-comments: List<Comment>
+Employee(employeeId: String, employeeName: String, employeeEmail: String)
+getEmployeeId(): String
+getEmployeeName(): String
Comment
-idCounter: int
-commentId: String
-comment: String
-employee: Employee
+Comment(comment: String, employee: Employee)
+getCommentId(): String
1
*
1
*
*
*
*
*
*
*

Figure 9 - Class Diagram

Software Development Process

Releases

Release 1 :

The first release focused on implementing the use case **“Display information by a question”**. As this was our initial release, our team of three (Raya, Layan, and Aseel) chose to work together to ensure we were aligned in understanding the implementation requirements and approach.

Planning Phase :

We began with a series of **three in-person meetings** to discuss and prioritize requirements. These meetings addressed key considerations:

- What is the most critical feature for this release?
- How should information be stored to ensure efficient searching?
- What format should the information take?
- How should file I/O be handled?

Functionality:

Our main functionality was to enable employees to search for information stored in a file. We decided to:

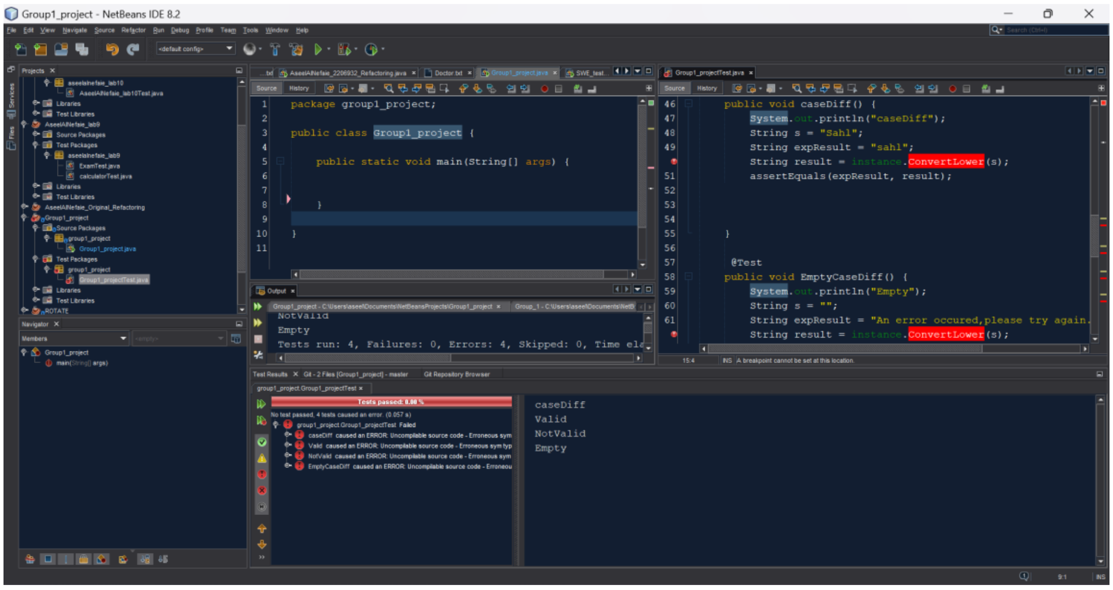
- Store the information in a file named information.txt.
- Ensure information was structured with clearly marked headers and associated content.
- Highlight keywords for efficient searching by enclosing them in &&.

A notable challenge was handling Arabic content from the guide provided by Jeddah Municipality. Since Java does not natively handle Arabic, we translated the guide into clear, searchable text in English.

User workflow:

When an Employee initiates a search, they are prompted to enter a keyword. The program searches the stored information, identifying and displaying the relevant content under its corresponding section, along with the matched keyword highlighted for clarity.

Test Driven Development:



```
package group1_project;

public class Group1_project {

    public static void main(String[] args) {


    }

}
```

```
    public void caseDiff() {
        System.out.println("caseDiff");
        String s = "Sahl";
        String expResult = "sahl";
        String result = instance.ConvertLower(s);
        assertEquals(expResult, result);




    }

     @Test
    public void EmptyCaseDiff() {
        System.out.println("Empty");
        String s = "";
        String expResult = "An error occured,please try again.
        String result = instance.ConvertLower(s);
```

```
NotValid
Empty
Tests run: 4, Failures: 0, Errors: 4, Skipped: 0, Time ela
```

```
Tests passed: 0.00 %
No test passed, 4 tests caused an error. (0.057 s)
group1_project.Group1_projectTest Failed
    caseDiff caused an ERROR: Uncompilable source code - Erroneous sym
    Valid caused an ERROR: Uncompilable source code - Erroneous sym typ
    NotValid caused an ERROR: Uncompilable source code - Erroneous sym
    EmptyCaseDiff caused an ERROR: Uncompilable source code - Erroneou
```

```
caseDiff
Valid
NotValid
Empty
```

Figure 10 - Release 1 (test cases before implementation)

Following the principles of **test-driven development**, we defined five test cases to guide implementation:

1. **File Non-Empty Test:**
    - Ensures information.txt is not empty, as an empty file would be critical for functionality.
2. **Keyword Match Not Found:**
    - Tests how the application handles cases where no information matches the entered keyword.
3. **Empty Keyword Input:**
    - Ensures an appropriate error message is displayed if the user inputs an empty keyword.
4. **Valid Keyword Input:**
    - Verifies that valid keywords are processed correctly, resulting in accurate searches and outputs.
5. **Invalid Keyword Input:**
    - Ensures non-string inputs are handled gracefully with error messages and a second chance to enter the keyword.

Implementation :

Initial Implementation (a day):

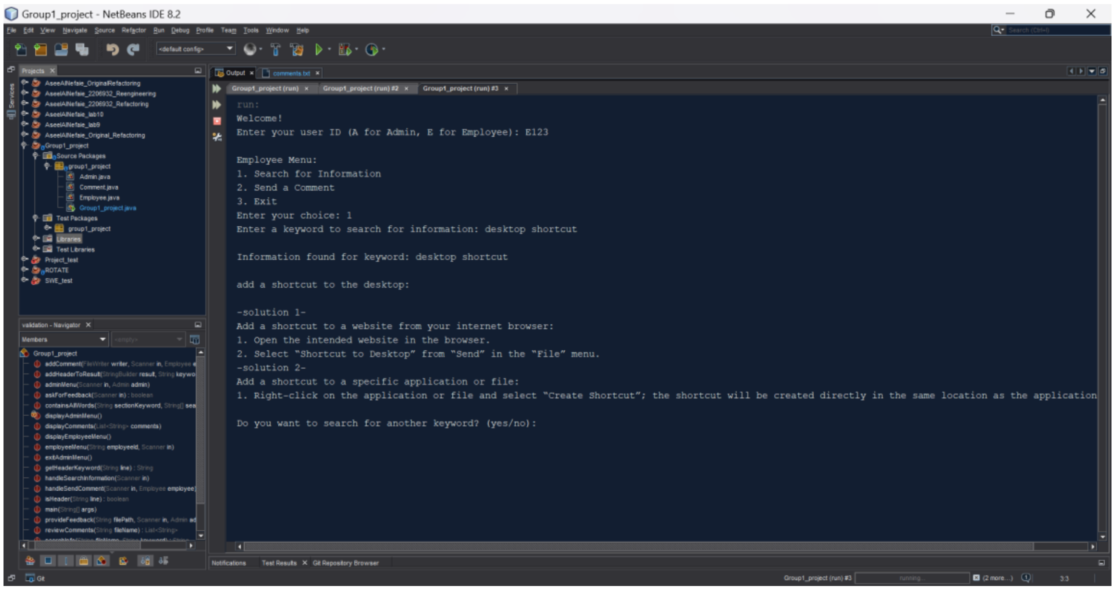
We collaboratively translated the Arabic guide into English, dividing the document into sections for efficiency. Raya has translated the whole file guide in English and stored the results in information.txt.

Searching Functionality(3 days):

- **Raya:** Took responsibility for structuring the file, ensuring that keywords were clearly enclosed in && for simplified searching.
- **Layan:** Focused on implementing the search logic. This included:
    - Reading the entered keyword.
    - Comparing it against the headers in information.txt.
    - Applying conditions to ensure a match between the entered keyword and the file content.
- **Aseel:** Implemented the display functionality. Key aspects included:
    - Displaying “Searching for information...” to keep users informed of progress.
    - Showing the found information clearly and asking if the user wanted to perform another search.

Examining (a day):

- After completing the initial implementation, we faced issues with invalid keyword entries. To address this, we:
    - Made additional test cases for keywords entries(test cases 3, 4 and 5).
    - Added a method for empty keywords, ensuring users were notified and given a second chance.
    - Handled cases where the input was not a string.
    - Tested the application with valid inputs to confirm correct functionality.



```
run:
Welcome!
Enter your user ID (A for Admin, E for Employee): E123

Employee Menu:
1. Search for Information
2. Send a Comment
3. Exit
Enter your choice: 1
Enter a keyword to search for information: desktop shortcut

Information found for keyword: desktop shortcut

add a shortcut to the desktop:

-solution 1-
Add a shortcut to a website from your internet browser:
1. Open the intended website in the browser.
2. Select "Shortcut to Desktop" from "Send" in the "File" menu.
-solution 2-
Add a shortcut to a specific application or file:
1. Right-click on the application or file and select "Create Shortcut"; the shortcut will be created directly in the same location as the application

Do you want to search for another keyword? (yes/no):
```

Figure 11 - Searching

Final Integration and Release :

We held a final meeting to:

- Combine our code into a single implementation.
- Ensure all five test cases passed successfully.

This collaborative effort resulted in a functional first release, delivering the ability to search for and display information efficiently.

Release 2 :

In this release, we adhered to Extreme Programming (XP) principles, for implementation of the use case "Add comment", emphasizing test-driven development and collaborative coding practices. Our primary objective was to implement functionality allowing employees to add comments and ensure these comments are properly stored with relevant details for future use.

Planning Phase:

The planning phase lasted approximately one hour, during which Layan and Aseel discussed the following key points:

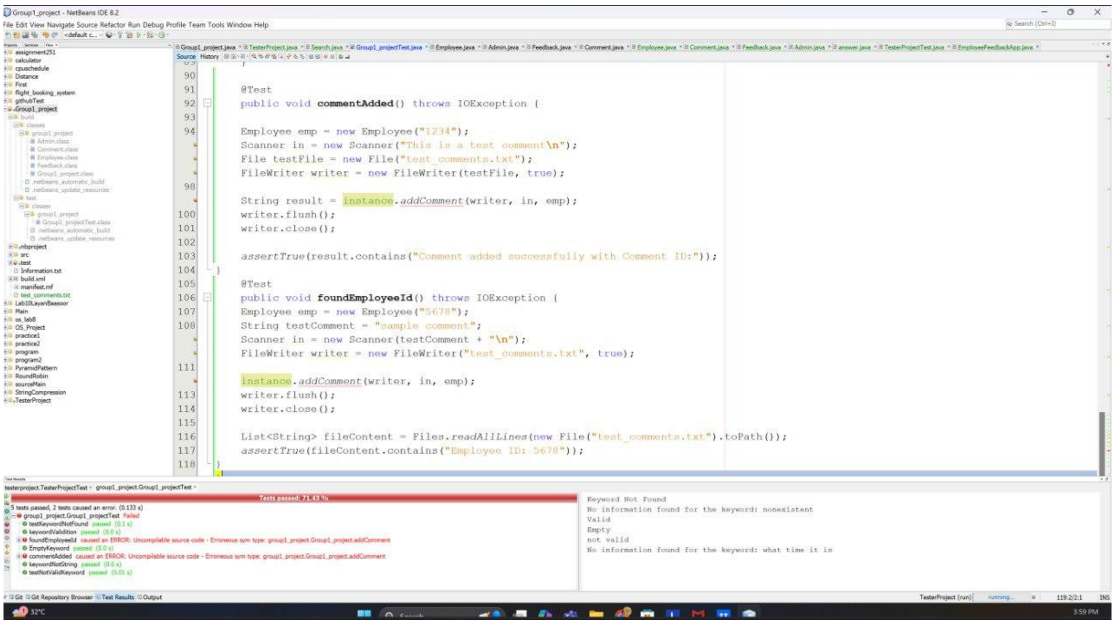
Functionality:
Employees need the ability to add comments, which will be stored in a file (comments.txt) that they will create in this release, with associated metadata. Each comment entry includes:

- Employee ID
- Comment text
- Comment ID

User Workflow:

Employees can input their comments through the console. The system generates a unique Comment ID and associates it with the Employee ID. The comment, along with its metadata, then stored in comments.txt. Once the process is completed, the application confirms successful addition by displaying the generated Comment ID to the user.

Test-Driven Development:



```
    @Test
    public void commentAdded() throws IOException {

        Employee emp = new Employee("1234");
        Scanner in = new Scanner("This is a test comment\n");
        File testFile = new File("test_comments.txt");
        FileWriter writer = new FileWriter(testFile, true);

        String result = instance.addComment(writer, in, emp);
        writer.flush();
        writer.close();

        assertTrue(result.contains("Comment added successfully with Comment ID:"));
    }

    @Test
    public void foundEmployeeId() throws IOException {
        Employee emp = new Employee("5678");
        String testComment = "sample comment";
        Scanner in = new Scanner(testComment + "\n");
        FileWriter writer = new FileWriter("test_comments.txt", true);

        instance.addComment(writer, in, emp);
        writer.flush();
        writer.close();

        List<String> fileContent = Files.readAllLines(new File("test_comments.txt").toPath());
        assertTrue(fileContent.contains("Employee ID: 5678"));
    }
```

```
Tests passed: 71.43 %
5 tests passed, 2 tests caused an error. (0.133 s)
group1_project.Group1_projectTest Failed
    testKeywordNotFound passed (0.1 s)
    keywordValidition passed (0.0 s)
    foundEmployeeId caused an ERROR: Uncompilable source code - Erroneous sym type: group1_project.Group1_project.addComment
    EmptyKeyword passed (0.0 s)
    commentAdded caused an ERROR: Uncompilable source code - Erroneous sym type: group1_project.Group1_project.addComment
    keywordNotString passed (0.0 s)
    testNotValidKeyword passed (0.01 s)
```

```
Keyword Not Found
No information found for the keyword: nonexistent
Valid
Empty
not valid
No information found for the keyword: what time it is
```

Figure 12 - Release 2 (test cases before implementation)

Following XP principles, test cases were written to guide the implementation. The test cases addressed the following scenarios:

1. **Comment Addition Test:** Ensures that a comment is added successfully, and a Comment ID is generated and displayed.
2. **Employee ID Storage Test:** Verifies that the Employee ID is stored correctly alongside the corresponding comment in the file.

Both test cases were reviewed and finalized by Layan and Aseel to ensure they covered the functionality comprehensively.

Implementation:

The development process was structured and collaborative, with clear division of tasks between Layan and Aseel.

1. **Initial Implementation (1.5 hours):**
   Layan implemented the core functionality for adding comments:
   - Accepting the comment input from the user.
   - Generating a unique Comment ID for each new comment.

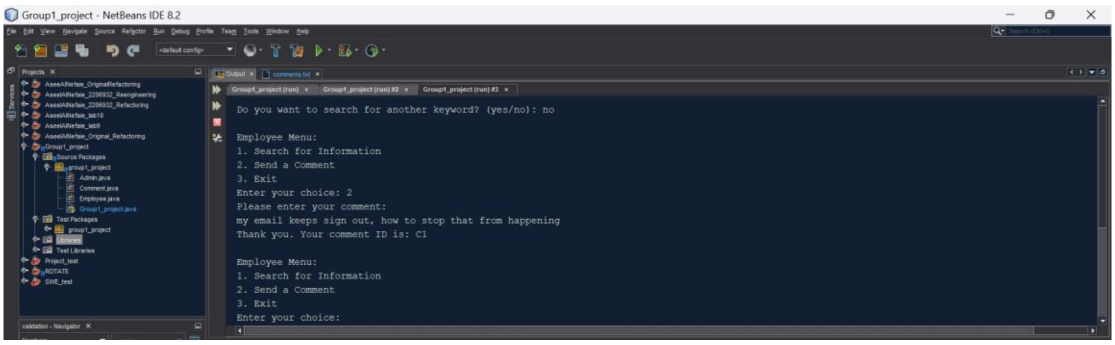
   - Storing the comment along with the Employee ID and Comment ID in the comments.txt file.

After completing this part, Layan explained the logic and structure to Aseel to ensure seamless collaboration.

2. **Validation and Testing (1 hour):**
   Aseel worked on writing test cases to validate the functionality.
   - Handle any I/O exceptions.
   - Ensuring that a comment is successfully added to the file.
   - Verifying that the associated Employee ID is stored correctly.

Aseel also reviewed the edge cases, such as handling input that could potentially cause errors (e.g., empty comments), to ensure consistency.

3. **Integration and Review (Online Meeting):**
   Layan and Aseel held an online meeting to integrate the implementation and validate its functionality against the test cases. Together, they ensured that the feature met the requirements and refined the code for readability and efficiency.

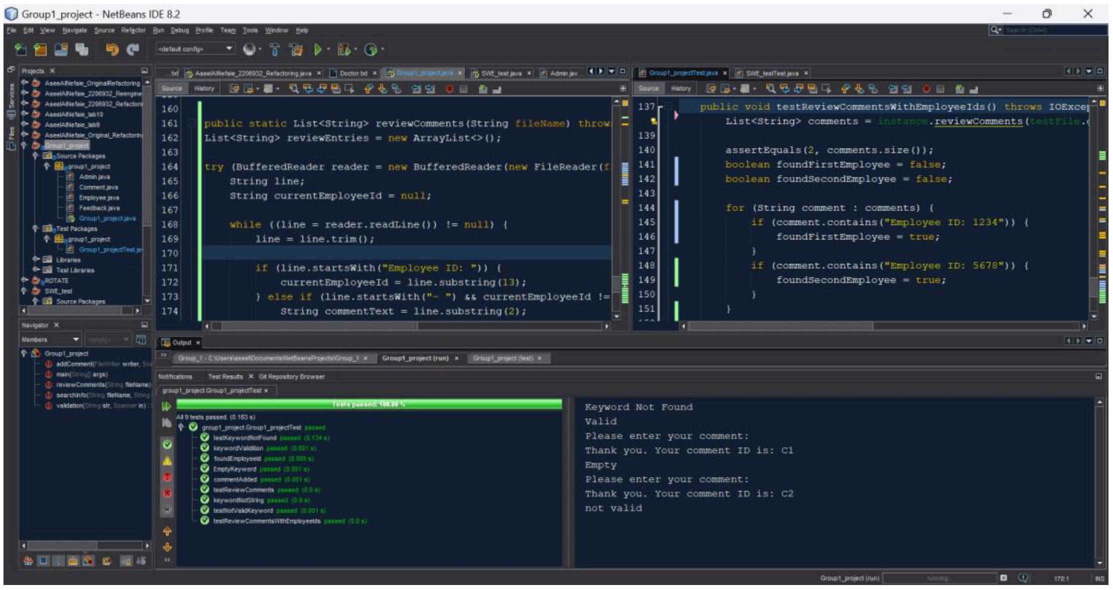


```
Do you want to search for another keyword? (yes/no): no

Employee Menu:
1. Search for Information
2. Send a Comment
3. Exit
Enter your choice: 2
Please enter your comment:
my email keeps sign out, how to stop that from happening
Thank you. Your comment ID is: C1

Employee Menu:
1. Search for Information
2. Send a Comment
3. Exit
Enter your choice:
```

Figure 13 - Add comment

Final Integration and Release :

The following test cases were executed successfully:



```java
public static List<String> reviewComments(String fileName) throw
List<String> reviewEntries = new ArrayList<>();

try (BufferedReader reader = new BufferedReader(new FileReader(f
    String line;
    String currentEmployeeId = null;

    while ((line = reader.readLine()) != null) {
        line = line.trim();

        if (line.startsWith("Employee ID: ")) {
            currentEmployeeId = line.substring(13);
        } else if (line.startsWith("- ") && currentEmployeeId !=
            String commentText = line.substring(2);
```

```java
public void testReviewCommentsWithEmployeeIds() throws IOExce
    List<String> comments = instance.reviewComments(testFile.

    assertEquals(2, comments.size());
    boolean foundFirstEmployee = false;
    boolean foundSecondEmployee = false;

    for (String comment : comments) {
        if (comment.contains("Employee ID: 1234")) {
            foundFirstEmployee = true;
        }
        if (comment.contains("Employee ID: 5678")) {
            foundSecondEmployee = true;
        }
    }
```

```
Tests passed: 100.00 %
All 9 tests passed. (0.183 s)
group1_project.Group1_projectTest passed
    testKeywordNotFound passed (0.134 s)
    keywordValidtion passed (0.001 s)
    foundEmployeeId passed (0.005 s)
    EmptyKeyword passed (0.001 s)
    commentAdded passed (0.001 s)
    testReviewComments passed (0.0 s)
    keywordNotString passed (0.0 s)
    testNotValidKeyword passed (0.001 s)
    testReviewCommentsWithEmployeeIds passed (0.0 s)
```

```
Keyword Not Found
Valid
Please enter your comment:
Thank you. Your comment ID is: C1
Empty
Please enter your comment:
Thank you. Your comment ID is: C2
not valid
```

Figure 7 - Release 2 (test cases after implementation)

- **Comment Addition:** Confirmed that comments were added to the file with a unique Comment ID.
- **Employee ID Storage:** Verified that Employee IDs matched those stored in the file alongside their respective comments.

All test cases passed, demonstrating the correctness and reliability of the feature.

Release 3 :

In this release, we followed Extreme Programming (XP) principles, including test-driven development, pair programming, and iterative collaboration. For implementing the use case “Displayed received comments”, Our goal was to implement functionality for admins to view employee comments without feedback.

The application allows admins to view comments without feedback, displaying all necessary details associated with the comment.

The implementation was completed collaboratively between Aseel and Raya, using tests driven development process.

Timing and pair programming arrangements ensured efficient task distribution.

This release demonstrates the effectiveness of XP principles, particularly test-driven development and collaborative coding, in delivering a consistent and functional feature.

Planning Phase:

The planning phase lasted approximately one hour, during which Aseel and Raya discussed the following key aspects:

Functionality:

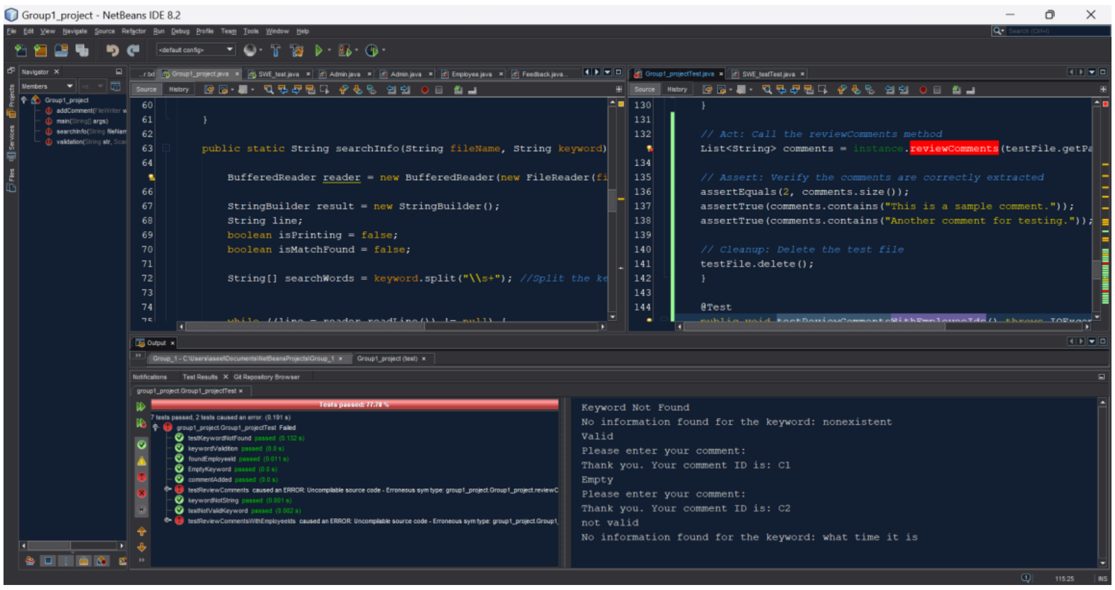
The application already stores employee comments, which might include comments requiring admin responses, or feedback on existing information (e.g., updates, corrections, or clarifications).

The new feature allows admins to review comments without feedback (where the feedback flag is false). These comments are read from a file (comments.txt) and displayed with details such as Employee ID, Comment ID and The comment text.

User Workflow:

When an admin logs in, the first menu choice will display these comments with no feedback. Showing the comment with its ID and the employee ID.

Test driven development:



```
    }

    public static String searchInfo(String fileName, String keyword)

        BufferedReader reader = new BufferedReader(new FileReader(fi

        StringBuilder result = new StringBuilder();
        String line;
        boolean isPrinting = false;
        boolean isMatchFound = false;

        String[] searchWords = keyword.split("\\s+"); //Split the ke
```

```
    }

    // Act: Call the reviewComments method
    List<String> comments = instance.reviewComments(testFile.getPa

    // Assert: Verify the comments are correctly extracted
    assertEquals(2, comments.size());
    assertTrue(comments.contains("This is a sample comment."));
    assertTrue(comments.contains("Another comment for testing."));

    // Cleanup: Delete the test file
    testFile.delete();
    }

    @Test
```

```
Tests passed: 77.78 %
7 tests passed, 2 tests caused an error. (0.191 s)
group1_project.Group1_projectTest Failed
    testKeywordNotFound passed (0.132 s)
    keywordValidtion passed (0.0 s)
    foundEmployeeId passed (0.011 s)
    EmptyKeyword passed (0.0 s)
    commentAdded passed (0.0 s)
    testReviewComments caused an ERROR: Uncompilable source code - Erroneous sym type: group1_project.Group1_project.reviewC
    keywordNotString passed (0.001 s)
    testNotValidKeyword passed (0.002 s)
    testReviewCommentsWithEmployeeIds caused an ERROR: Uncompilable source code - Erroneous sym type: group1_project.Group1_
```

```
Keyword Not Found
No information found for the keyword: nonexistent
Valid
Please enter your comment:
Thank you. Your comment ID is: C1
Empty
Please enter your comment:
Thank you. Your comment ID is: C2
not valid
No information found for the keyword: what time it is
```

Figure 14 - Release 3 (test cases before implementation)

Following XP's test-driven development principle, we first wrote test cases to define the behavior of the feature. These test cases helped guide the implementation:

1. Review Comments:

   Verifies that all comments without feedback are correctly displayed with full details, including Employee ID, Comment ID, and the comment text.

2. Review Comments With Employee Ids:

   Ensures that the displayed Employee IDs match those stored with the corresponding comments in the comments.txt file.

Aseel and Raya worked together on writing these test cases, ensuring they were clear and covered edge cases.

Implementation:

The implementation followed a structured workflow with a division of tasks:

Initial Implementation (1 hour):

Aseel implemented the functionality to read comments from the file.

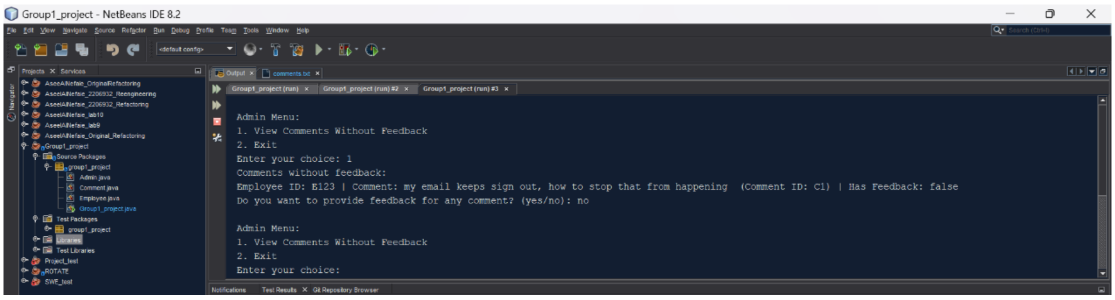
1. Reading the file line by line.
2. Filtering comments based on the feedback flag (false).
3. Storing relevant details for display (Employee ID, Comment ID, and comment text).

After completing this section, Aseel explained the logic, variable names, and data flow to Raya to ensure a smooth handoff.

Display Functionality (1.5 hours):

Raya took over and implemented the display functionality.

1. Handling potential I/O exceptions appropriately.
2. She ensuring the filtered comments were displayed with all required details, formatted clearly for the admin.



```
Admin Menu:
1. View Comments Without Feedback
2. Exit
Enter your choice: 1
Comments without feedback:
Employee ID: E123 | Comment: my email keeps sign out, how to stop that from happening  (Comment ID: C1) | Has Feedback: false
Do you want to provide feedback for any comment? (yes/no): no

Admin Menu:
1. View Comments Without Feedback
2. Exit
Enter your choice:
```

Figure 15- View received comments(for admin)

Final Integration and Release :

We combined our code during an online meeting. We tested the feature using various input files and scenarios, ensuring the comments were displayed as expected.

Both test cases passed successfully, verifying the correctness of the implementation.

Release 4 :

In this release, we focused on implementing a functionality that allows admins to provide feedback on employee comments. For the use case “add feedback”, By following Extreme Programming (XP) principles, particularly test-driven development and collaborative coding, the feature ensures comments are updated with feedback efficiently and accurately. Raya and Layan jointly worked on this release, dividing tasks strategically to maximize productivity.

Planning Phase:

The planning phase lasted less than an hour, during which Raya and Layan collaborated to outline the requirements and approach.

Functionality:
Admins can add feedback to specific employee comments stored in a file (comments.txt). The feature ensures:

- Feedback is linked to a specific Comment ID.
- Admin metadata (Admin ID) is included with the feedback.
- The feedback is appended below the corresponding comment in the file.

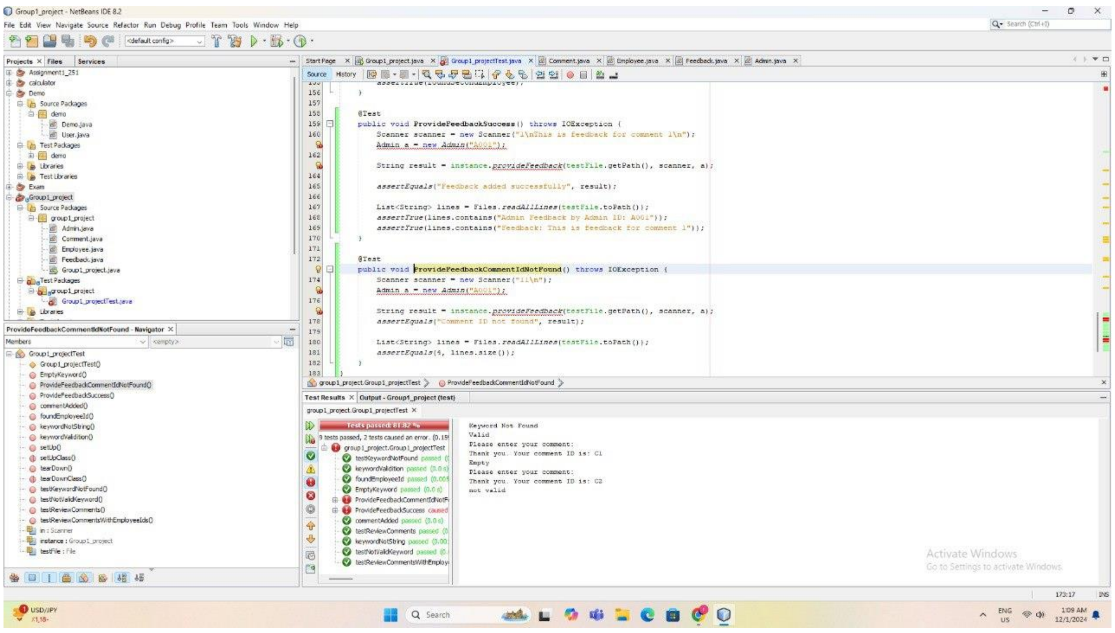
User Workflow:

Admins are prompted to enter the Comment ID they wish to provide feedback for.

If the Comment ID exists, the admin can input their feedback. The program appends the feedback along with the Admin ID to the file.

If the Comment ID does not exist, an error message is displayed.

Test-Driven Development:



```java
    @Test
    public void ProvideFeedbackSuccess() throws IOException {
        Scanner scanner = new Scanner("1\nThis is feedback for comment 1\n");
        Admin a = new Admin("A001");

        String result = instance.provideFeedback(testFile.getPath(), scanner, a);

        assertEquals("Feedback added successfully", result);

        List<String> lines = Files.readAllLines(testFile.toPath());
        assertTrue(lines.contains("Admin Feedback by Admin ID: A001"));
        assertTrue(lines.contains("Feedback: This is feedback for comment 1"));
    }

    @Test
    public void ProvideFeedbackCommentIdNotFound() throws IOException {
        Scanner scanner = new Scanner("11\n");
        Admin a = new Admin("A001");

        String result = instance.provideFeedback(testFile.getPath(), scanner, a);
        assertEquals("Comment ID not found", result);

        List<String> lines = Files.readAllLines(testFile.toPath());
        assertEquals(4, lines.size());
    }
}
```

```
Tests passed: 81.82 %
9 tests passed, 2 tests caused an error. (0.15
group1_project.Group1_projectTest
  testKeywordNotFound passed
  keywordValidition passed (0.0 s)
  foundEmployeeId passed (0.005
  EmptyKeyword passed (0.0 s)
  ProvideFeedbackCommentIdNotF
  ProvideFeedbackSuccess caused
  commentAdded passed (0.0 s)
  testReviewComments passed
  keywordNotString passed (0.00
  testNotValidKeyword passed
  testReviewCommentsWithEmploy
```

```
Keyword Not Found
Valid
Please enter your comment:
Thank you. Your comment ID is: C1
Empty
Please enter your comment:
Thank you. Your comment ID is: C2
not valid
```

Figure 16 - Release 4 (testcases before implementation)

To ensure robust functionality, Raya and Layan wrote the following test cases:

- **Provide Feedback Success:** Validates that feedback is successfully added to the file for a given Comment ID.
- **Provide Feedback Comment ID Not Found:** Ensures the application handles scenarios where the provided Comment ID does not exist.

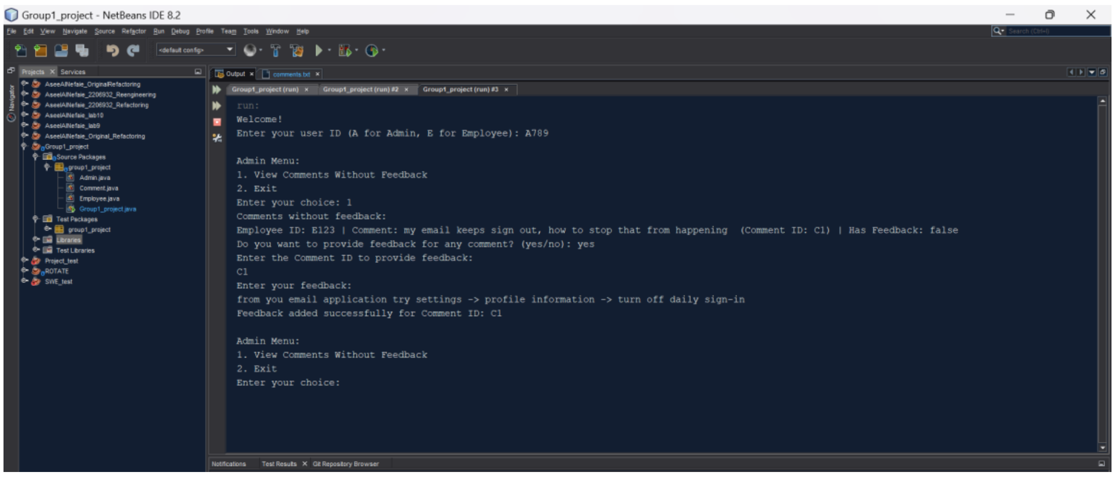
Both test cases were reviewed and refined collaboratively.

Implementation:

1. **Initial Implementation (2 hours):**
   Layan worked on implementing the core functionality for locating and updating comments:
   - Reading and parsing the file (comments.txt).
   - Validating the Comment ID format and existence in the file.
   - Appending feedback with the Admin ID below the corresponding comment.
   - Handling file-related errors gracefully.

After completing this, Layan explained the workflow and logic to Raya for further enhancements.

2. **Feedback Input and Error Handling (1.5 hours):**
   Raya extended the functionality by:
   - Improving input validation (e.g., ensuring proper Comment ID format).
   - Adding error messages for scenarios like missing or invalid Comment IDs.
   - Ensuring all updates to the file maintain its original structure and formatting.
3. **Integration and Testing (Online Meeting):**
   Raya and Layan met online to integrate their work and validate it against the defined test cases. Together, they ensured the feature handled both typical and edge cases effectively.



```
run:
Welcome!
Enter your user ID (A for Admin, E for Employee): A789

Admin Menu:
1. View Comments Without Feedback
2. Exit
Enter your choice: 1
Comments without feedback:
Employee ID: E123 | Comment: my email keeps sign out, how to stop that from happening  (Comment ID: C1) | Has Feedback: false
Do you want to provide feedback for any comment? (yes/no): yes
Enter the Comment ID to provide feedback:
C1
Enter your feedback:
from you email application try settings -> profile information -> turn off daily sign-in
Feedback added successfully for Comment ID: C1

Admin Menu:
1. View Comments Without Feedback
2. Exit
Enter your choice:
```

Figure 17 - Provide Feedback

Final Integration and Release :

The following test cases were executed successfully:

1. **Provide Feedback Success:**
   - Verified that the feedback, along with the Admin ID, was appended to the correct comment in the file.
   - Ensured the feedback format matched the requirements.
2. **Provide Feedback Comment ID Not Found:**
   - Confirmed the application displayed an appropriate error message when an invalid Comment ID was entered.
   - Ensured the file remained unchanged in such scenarios.

Refactoring

Main method

Layan refactored the main method to improve readability and maintainability. Instead of handling all functionalities within a single large main method, Layan splits the program into smaller, reusable methods organized by functionality and user role (employee and admin). This includes:

A new userAccess method was introduced to separate user type validation and role-based access. It checks the prefix of the user ID (A for Admin, E for Employee) and directs users to their respective menus.

Layan created employeeMenu and adminMenu methods to handle functionality specific to employees and admins. Each menu displays options and processes user input in a loop, allowing repeated access until the user chooses to exit.

Also, Layan added displayEmployeeMenu to show the options available to employees, such as searching for information and sending comments. Then created handleSearchInformation and handleSendComment methods to manage specific tasks related to employees. These methods encapsulate the logic for searching and sending comments, respectively, making them easier to understand and maintain. Layan added displayAdminMenu to show options for the admin, such as reviewing comments without feedback and viewCommentsWithoutFeedback to displaying comments without feedback. The method interacts with the admin user and keeps the functionality clean and encapsulated. Created utility methods such as displayComments and askForFeedback to manage specific sub-tasks, improving code reuse and clarity.

Repeated logic such as handling file operations and displaying menus, was encapsulated in methods. The main method now only initializes the scanner and user session (userAccess), delegating all role-specific logic to dedicated methods. This makes the main method easy to follow.

searchInfo method

The searchInfo method was used to search a file for information based on a keyword, but it had many problems, like repeated code, being too long, confusing. logic, and unnecessary complexity. To fix this, Raya made several improvements .

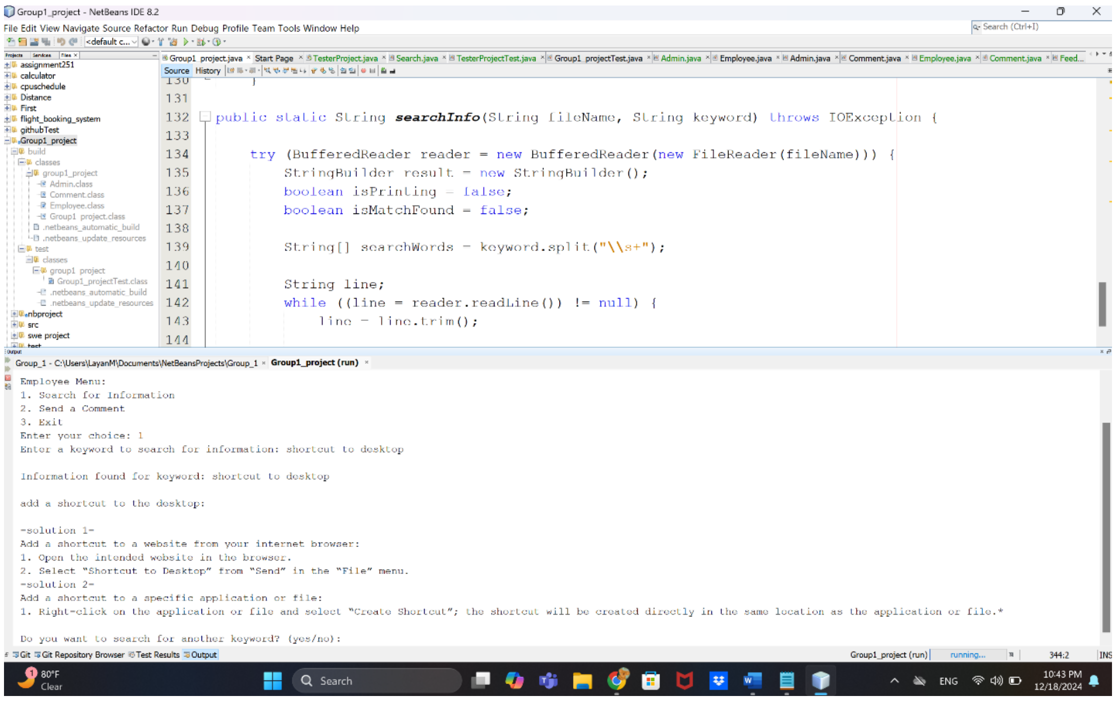
First, removed repeated tasks like checking headers or keywords by creating small helper methods. Then, split the long method into smaller, focused parts, each handling one job. also simplified the logic by replacing complicated `if` statements with clear, straightforward checks. Instead of handling the same data multiple times, Raya grouped related tasks into one helper method and removed unnecessary parts to focus only on what was needed .

Now, the searchInfo method is much cleaner. The main method is short and only handles the main steps, while helper methods like isHeader and containsAllWords handle the details. This makes the code easier to read, reuse, and update.

provideFeedback method

Aseel adds a null check for the Admin object and its ID to prevent potential NullPointerException errors, ensuring valid admin details are provided before proceeding. The file existence check using File.exists() ensures the method fails gracefully if the specified file path is invalid. Instead of reading all lines at once with Files.readAllLines(), the method now uses a BufferedReader to read the file line by line, optimizing memory usage for larger files. The input for the Comment ID and feedback is trimmed to remove unnecessary whitespace, enhancing user input validation. The code still iterates through the file to find the comment and appends the feedback below it but adds further clarity and structure. For writing back to the file, a BufferedWriter is used to ensure efficient writing and proper handling of line breaks. Overall, these changes enhance the method's performance, error handling, and readability while keeping the logic consistent with the original intent.

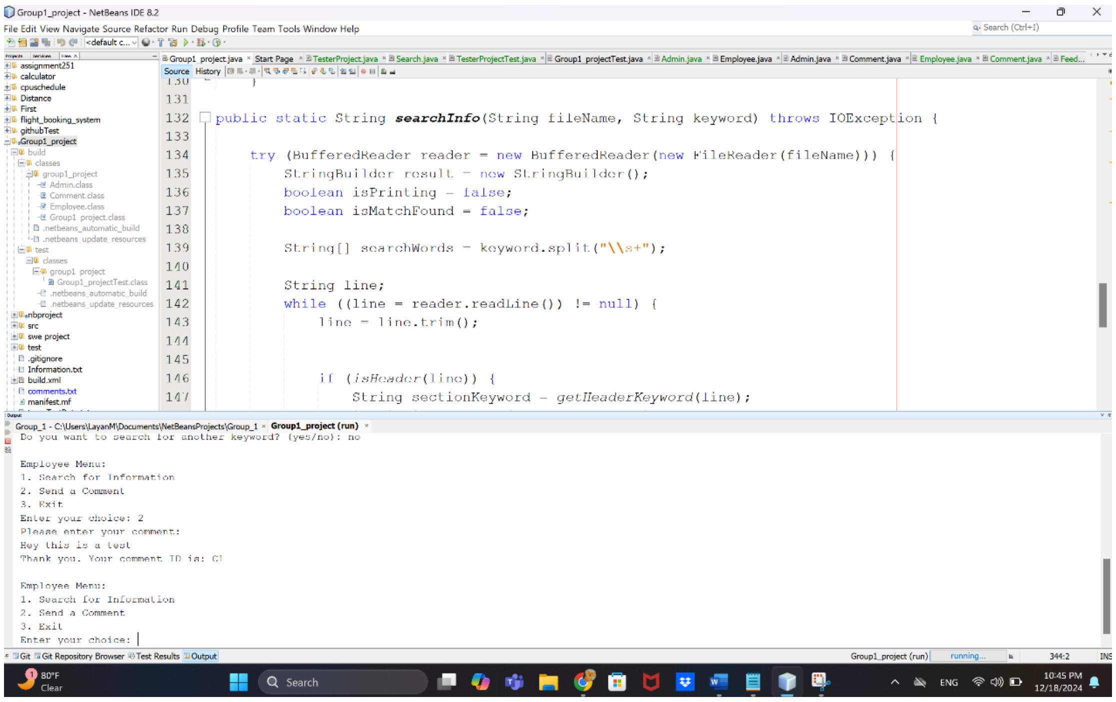
Implementation



```java
public static String searchInfo(String fileName, String keyword) throws IOException {

    try (BufferedReader reader = new BufferedReader(new FileReader(fileName))) {
        StringBuilder result = new StringBuilder();
        boolean isPrinting = false;
        boolean isMatchFound = false;

        String[] searchWords = keyword.split("\\s+");

        String line;
        while ((line = reader.readLine()) != null) {
            line = line.trim();
```

```
Employee Menu:
1. Search for Information
2. Send a Comment
3. Exit
Enter your choice: 1
Enter a keyword to search for information: shortcut to desktop

Information found for keyword: shortcut to desktop

add a shortcut to the desktop:

-solution 1-
Add a shortcut to a website from your internet browser:
1. Open the intended website in the browser.
2. Select "Shortcut to Desktop" from "Send" in the "File" menu.
-solution 2-
Add a shortcut to a specific application or file:
1. Right-click on the application or file and select "Create Shortcut"; the shortcut will be created directly in the same location as the application or file.*

Do you want to search for another keyword? (yes/no):
```

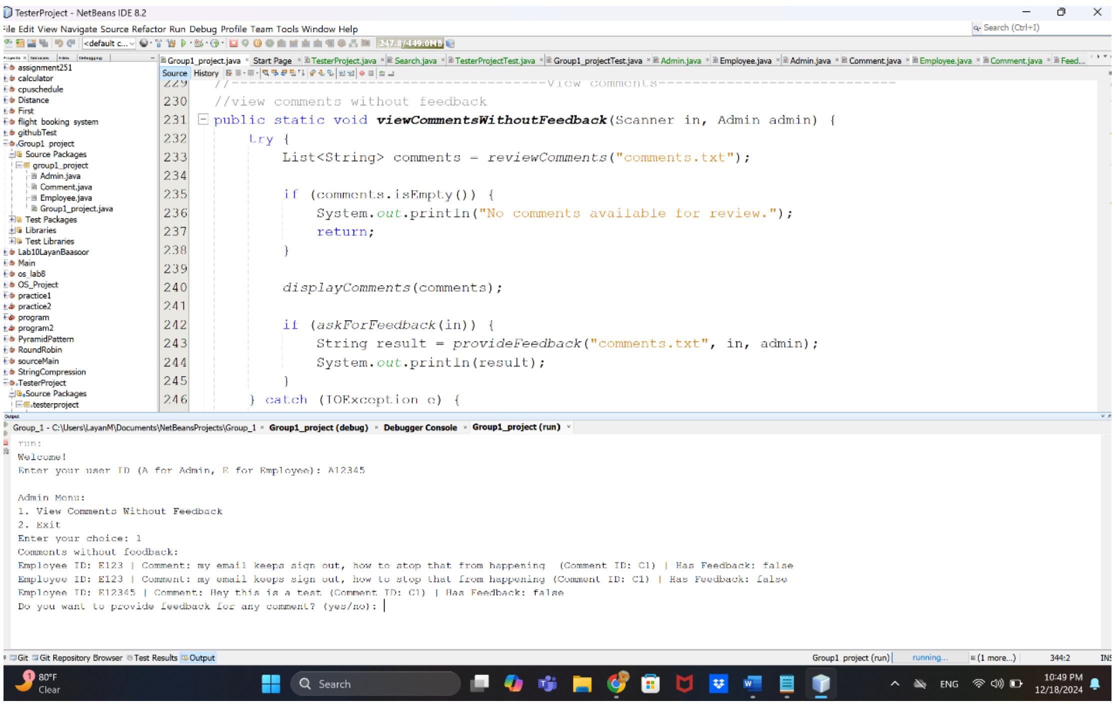
Figure 18- searchInfo method



```java
public static String searchInfo(String fileName, String keyword) throws IOException {

    try (BufferedReader reader = new BufferedReader(new FileReader(fileName))) {
        StringBuilder result = new StringBuilder();
        boolean isPrinting = false;
        boolean isMatchFound = false;

        String[] searchWords = keyword.split("\\s+");

        String line;
        while ((line = reader.readLine()) != null) {
            line = line.trim();


            if (isHeader(line)) {
                String sectionKeyword = getHeaderKeyword(line);
```

```
Do you want to search for another keyword? (yes/no): no

Employee Menu:
1. Search for Information
2. Send a Comment
3. Exit
Enter your choice: 2
Please enter your comment:
Hey this is a test
Thank you. Your comment ID is: C1

Employee Menu:
1. Search for Information
2. Send a Comment
3. Exit
Enter your choice:
```

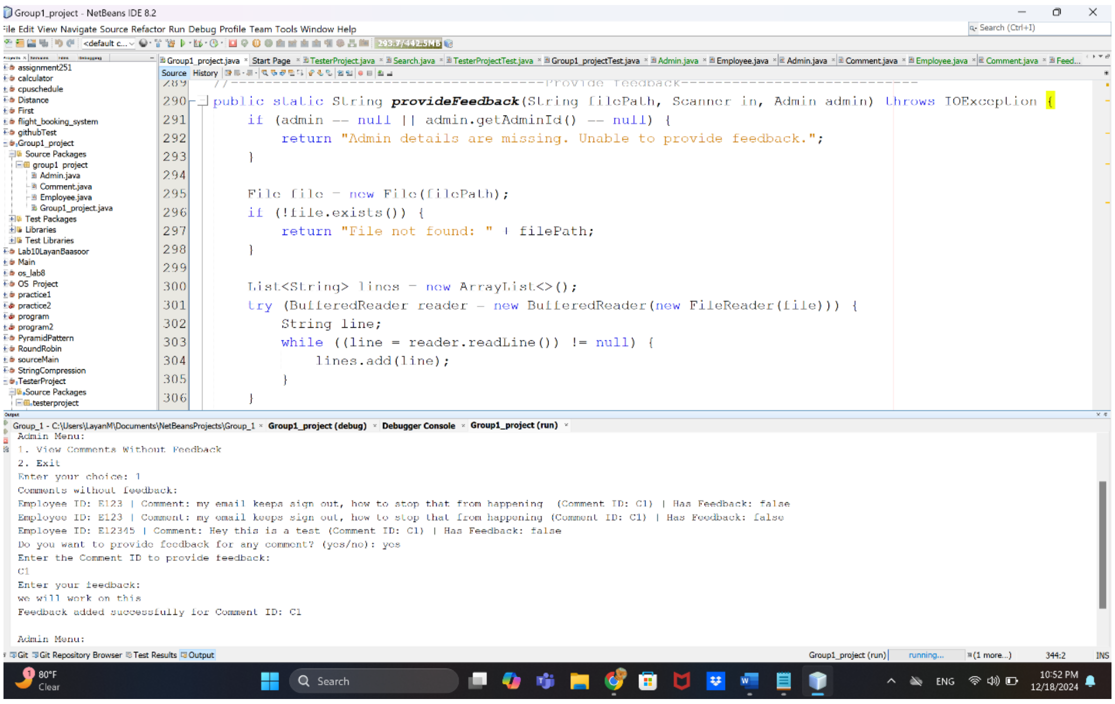
Figure 19- addComment method



```
//------------------------------------View comments-----------------------------
//view comments without feedback
public static void viewCommentsWithoutFeedback(Scanner in, Admin admin) {
    try {
        List<String> comments = reviewComments("comments.txt");

        if (comments.isEmpty()) {
            System.out.println("No comments available for review.");
            return;
        }

        displayComments(comments);

        if (askForFeedback(in)) {
            String result = provideFeedback("comments.txt", in, admin);
            System.out.println(result);
        }
    } catch (IOException e) {
```

```
run:
Welcome!
Enter your user ID (A for Admin, E for Employee): A12345

Admin Menu:
1. View Comments Without Feedback
2. Exit
Enter your choice: 1
Comments without feedback:
Employee ID: E123 | Comment: my email keeps sign out, how to stop that from happening  (Comment ID: C1) | Has Feedback: false
Employee ID: E123 | Comment: my email keeps sign out, how to stop that from happening (Comment ID: C1) | Has Feedback: false
Employee ID: E12345 | Comment: Hey this is a test (Comment ID: C1) | Has Feedback: false
Do you want to provide feedback for any comment? (yes/no): 
```

Figure 20- viewCommentsWithoutFeedback method

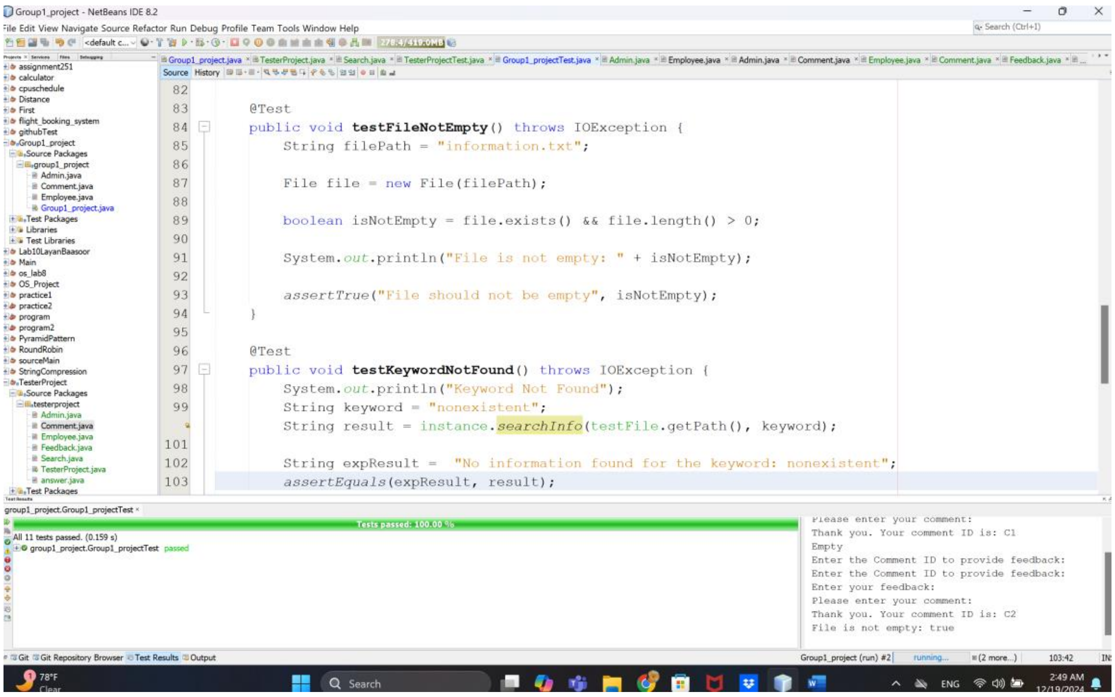


```
//------------------------------------Provide feedback-----------------------------
public static String provideFeedback(String filePath, Scanner in, Admin admin) throws IOException {
    if (admin == null || admin.getAdminId() == null) {
        return "Admin details are missing. Unable to provide feedback.";
    }

    File file = new File(filePath);
    if (!file.exists()) {
        return "File not found: " + filePath;
    }

    List<String> lines = new ArrayList<>();
    try (BufferedReader reader = new BufferedReader(new FileReader(file))) {
        String line;
        while ((line = reader.readLine()) != null) {
            lines.add(line);
        }
    }
```

```
Admin Menu:
1. View Comments Without Feedback
2. Exit
Enter your choice: 1
Comments without feedback:
Employee ID: E123 | Comment: my email keeps sign out, how to stop that from happening  (Comment ID: C1) | Has Feedback: false
Employee ID: E123 | Comment: my email keeps sign out, how to stop that from happening (Comment ID: C1) | Has Feedback: false
Employee ID: E12345 | Comment: Hey this is a test (Comment ID: C1) | Has Feedback: false
Do you want to provide feedback for any comment? (yes/no): yes
Enter the Comment ID to provide feedback:
C1
Enter your feedback:
we will work on this
Feedback added successfully for Comment ID: C1

Admin Menu:
```

Figure 21- provideFeedback method

Testing

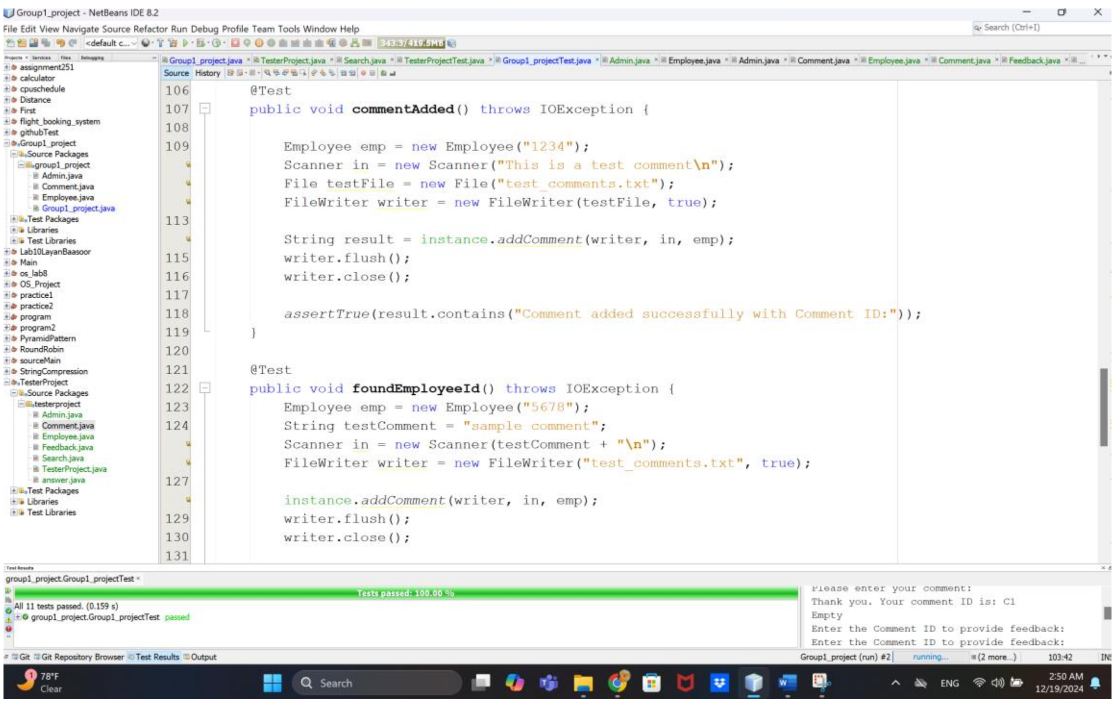


```java
    @Test
    public void testFileNotEmpty() throws IOException {
        String filePath = "information.txt";

        File file = new File(filePath);

        boolean isNotEmpty = file.exists() && file.length() > 0;

        System.out.println("File is not empty: " + isNotEmpty);

        assertTrue("File should not be empty", isNotEmpty);
    }

    @Test
    public void testKeywordNotFound() throws IOException {
        System.out.println("Keyword Not Found");
        String keyword = "nonexistent";
        String result = instance.searchInfo(testFile.getPath(), keyword);

        String expResult =  "No information found for the keyword: nonexistent";
        assertEquals(expResult, result);
```

All 11 tests passed. (0.159 s)
group1_project.Group1_projectTest passed

```
Please enter your comment:
Thank you. Your comment ID is: C1
Empty
Enter the Comment ID to provide feedback:
Enter the Comment ID to provide feedback:
Enter your feedback:
Please enter your comment:
Thank you. Your comment ID is: C2
File is not empty: true
```

Figure 22- test case searchInfo

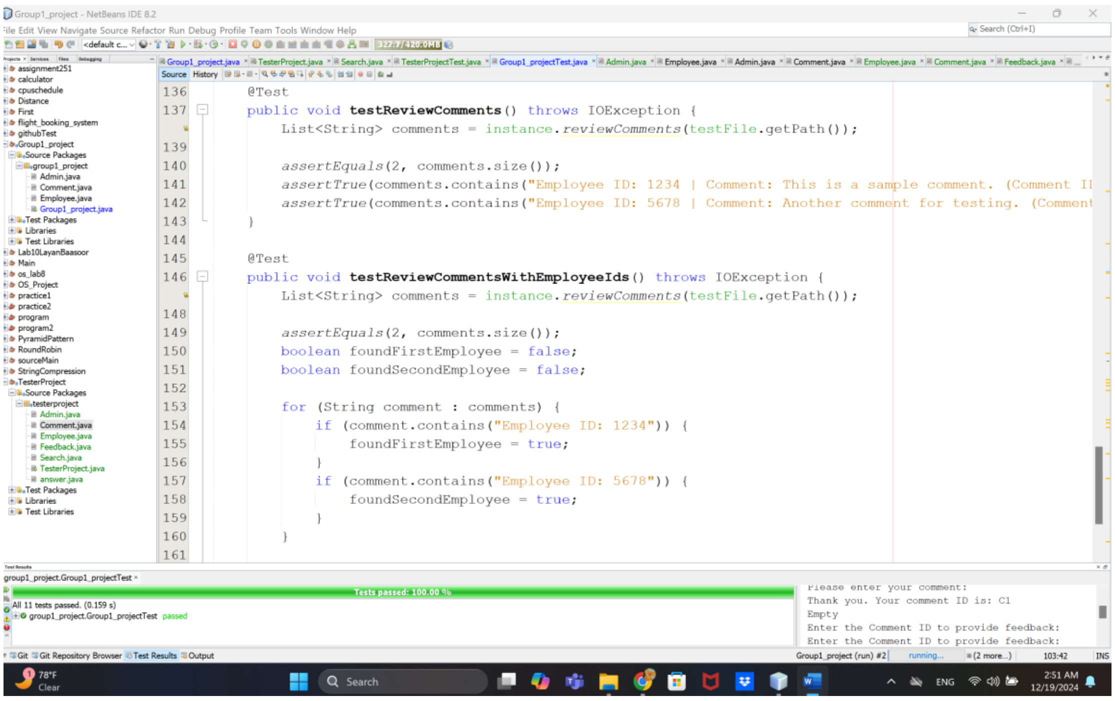


```java
    @Test
    public void commentAdded() throws IOException {

        Employee emp = new Employee("1234");
        Scanner in = new Scanner("This is a test comment\n");
        File testFile = new File("test_comments.txt");
        FileWriter writer = new FileWriter(testFile, true);

        String result = instance.addComment(writer, in, emp);
        writer.flush();
        writer.close();

        assertTrue(result.contains("Comment added successfully with Comment ID:"));
    }

    @Test
    public void foundEmployeeId() throws IOException {
        Employee emp = new Employee("5678");
        String testComment = "sample comment";
        Scanner in = new Scanner(testComment + "\n");
        FileWriter writer = new FileWriter("test_comments.txt", true);

        instance.addComment(writer, in, emp);
        writer.flush();
        writer.close();
```

All 11 tests passed. (0.159 s)
group1_project.Group1_projectTest passed

```
Please enter your comment:
Thank you. Your comment ID is: C1
Empty
Enter the Comment ID to provide feedback:
Enter the Comment ID to provide feedback:
```

Figure 23- test case addComment

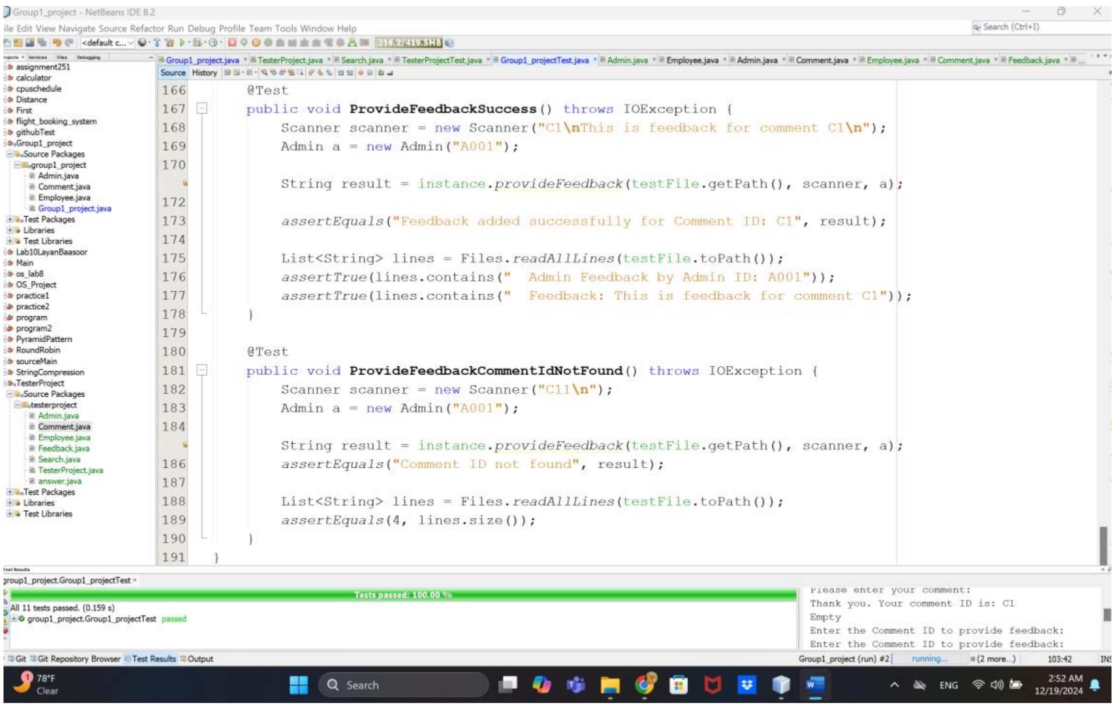


```java
    @Test
    public void testReviewComments() throws IOException {
        List<String> comments = instance.reviewComments(testFile.getPath());

        assertEquals(2, comments.size());
        assertTrue(comments.contains("Employee ID: 1234 | Comment: This is a sample comment. (Comment II
        assertTrue(comments.contains("Employee ID: 5678 | Comment: Another comment for testing. (Comment
    }

    @Test
    public void testReviewCommentsWithEmployeeIds() throws IOException {
        List<String> comments = instance.reviewComments(testFile.getPath());

        assertEquals(2, comments.size());
        boolean foundFirstEmployee = false;
        boolean foundSecondEmployee = false;

        for (String comment : comments) {
            if (comment.contains("Employee ID: 1234")) {
                foundFirstEmployee = true;
            }
            if (comment.contains("Employee ID: 5678")) {
                foundSecondEmployee = true;
            }
        }
```

```
Tests passed: 100.00 %
All 11 tests passed. (0.159 s)
group1_project.Group1_projectTest passed
```

```
Please enter your comment:
Thank you. Your comment ID is: C1
Empty
Enter the Comment ID to provide feedback:
Enter the Comment ID to provide feedback:
```

Figure 24- test case reviewComments



```java
    @Test
    public void ProvideFeedbackSuccess() throws IOException {
        Scanner scanner = new Scanner("C1\nThis is feedback for comment C1\n");
        Admin a = new Admin("A001");

        String result = instance.provideFeedback(testFile.getPath(), scanner, a);

        assertEquals("Feedback added successfully for Comment ID: C1", result);

        List<String> lines = Files.readAllLines(testFile.toPath());
        assertTrue(lines.contains("  Admin Feedback by Admin ID: A001"));
        assertTrue(lines.contains("  Feedback: This is feedback for comment C1"));
    }

    @Test
    public void ProvideFeedbackCommentIdNotFound() throws IOException {
        Scanner scanner = new Scanner("C11\n");
        Admin a = new Admin("A001");

        String result = instance.provideFeedback(testFile.getPath(), scanner, a);
        assertEquals("Comment ID not found", result);

        List<String> lines = Files.readAllLines(testFile.toPath());
        assertEquals(4, lines.size());
    }
}
```

```
Tests passed: 100.00 %
All 11 tests passed. (0.159 s)
group1_project.Group1_projectTest passed
```

```
Please enter your comment:
Thank you. Your comment ID is: C1
Empty
Enter the Comment ID to provide feedback:
Enter the Comment ID to provide feedback:
```

Figure 25- test case provideFeedback

Task That Have Been Done By Each Student

At the beginning of the project, all team members—Aseel Alnefaie, Layan Baasoor, and Raya Alyoubi—collaborated to establish a shared understanding of the overall project vision. Together, we outlined the primary functionality of the application, prioritized features based on importance and planned the development process accordingly.

For Release One, the entire team worked closely to implement the foundational features of the application. Work distribution was organized effectively, with tasks allocated based on each member's strengths and areas of interest. Regular discussions and pair programming sessions ensured that everyone contributed to core functionality while maintaining a cohesive and collaborative workflow.

Aseel Alnefaie:

- Implemented the display functionality for information search in Release 1.
- Worked on writing test cases to validate the functionality in Release 2.
- Implemented the core file reading logic for admin comment reviews in Release 3.
- Refactored provideFeedback method.

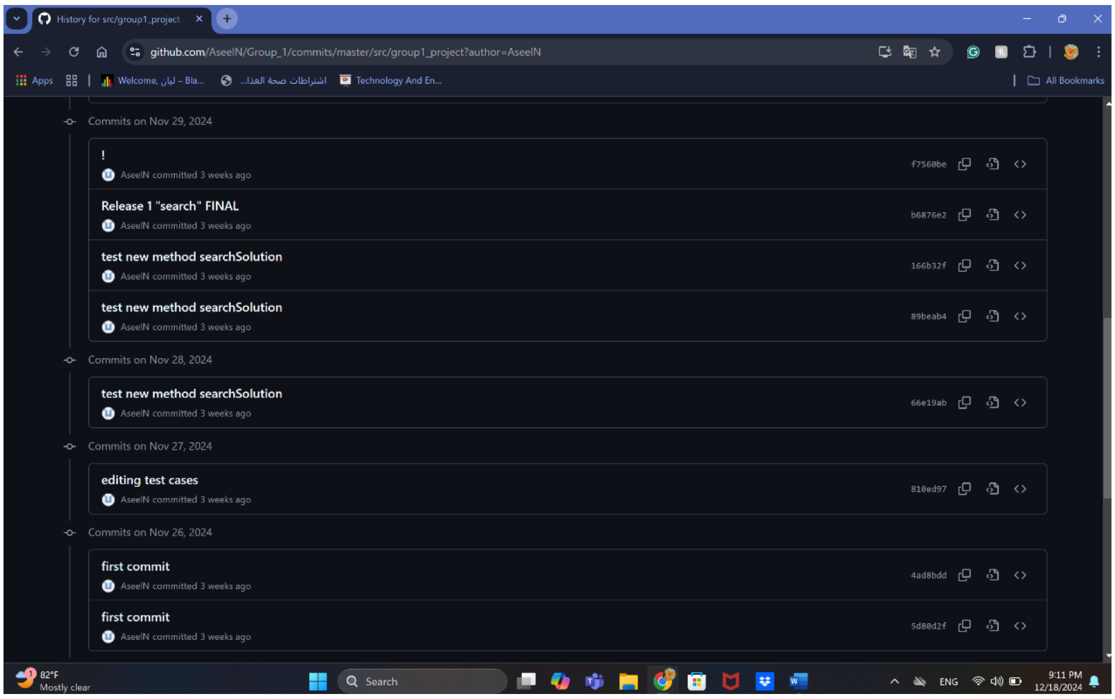
Layan Baasoor:

- Developed the search logic for keyword matching in Release 1.
- Implemented the functionality for employee comment addition in Release 2.
- Worked on validating and updating the comments file for admin feedback in Release 4.
- Refactored main method

Raya Alyoubi:

- Structured and formatted the information file in Release 1.
- Designed the display functionality for admin comment reviews in Release 3.
- Implemented provideFeedback method and enhanced feedback input handling and integrated the file update mechanism in Release 4.
- Refactored the searchInfo method

Distributed Tasks From GitHub



History for src/group1_project

github.com/AseelN/Group_1/commits/master/src/group1_project?author=AseelN

Commits on Nov 29, 2024

!
AseelN committed 3 weeks ago — f7560be

Release 1 "search" FINAL
AseelN committed 3 weeks ago — b6876e2

test new method searchSolution
AseelN committed 3 weeks ago — 166b32f

test new method searchSolution
AseelN committed 3 weeks ago — 89beab4

Commits on Nov 28, 2024

test new method searchSolution
AseelN committed 3 weeks ago — 66e19ab

Commits on Nov 27, 2024

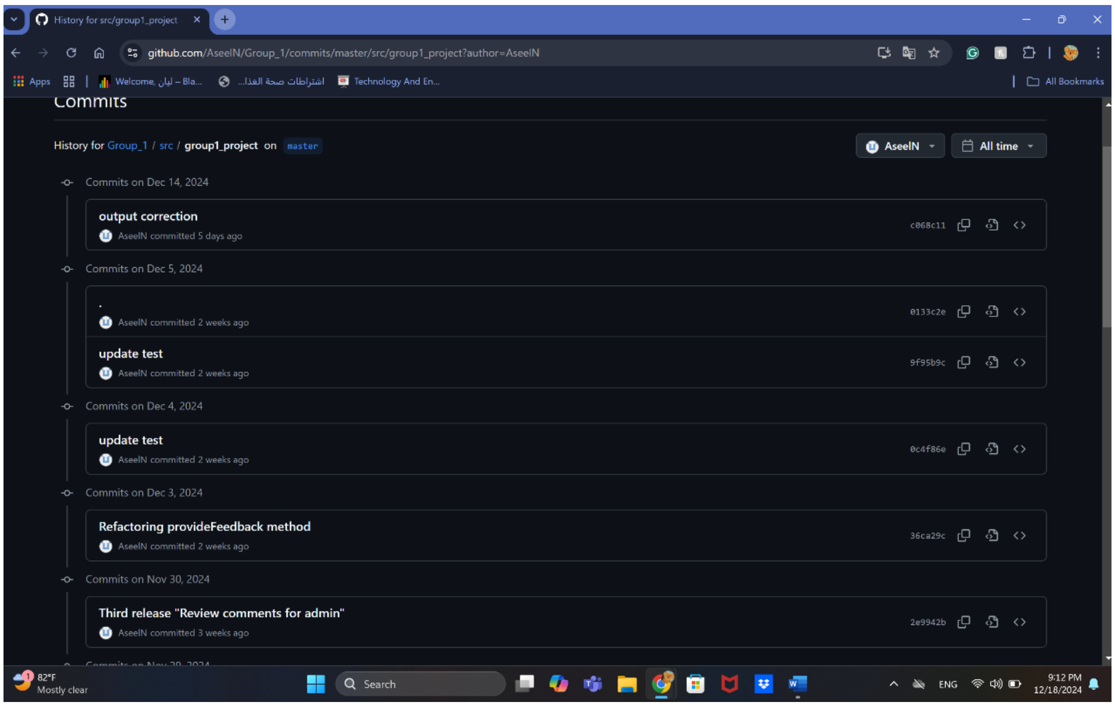
editing test cases
AseelN committed 3 weeks ago — 810ed97

Commits on Nov 26, 2024

first commit
AseelN committed 3 weeks ago — 4ad8bdd

first commit
AseelN committed 3 weeks ago — 5d80d2f

Figure 26- Aseel GitHub



History for src/group1_project

github.com/AseelN/Group_1/commits/master/src/group1_project?author=AseelN

Commits

History for Group_1 / src / group1_project on master

AseelN — All time

Commits on Dec 14, 2024

output correction
AseelN committed 5 days ago — c068c11

Commits on Dec 5, 2024

.
AseelN committed 2 weeks ago — 0133c2e

update test
AseelN committed 2 weeks ago — 9f95b9c

Commits on Dec 4, 2024

update test
AseelN committed 2 weeks ago — 0c4f86e

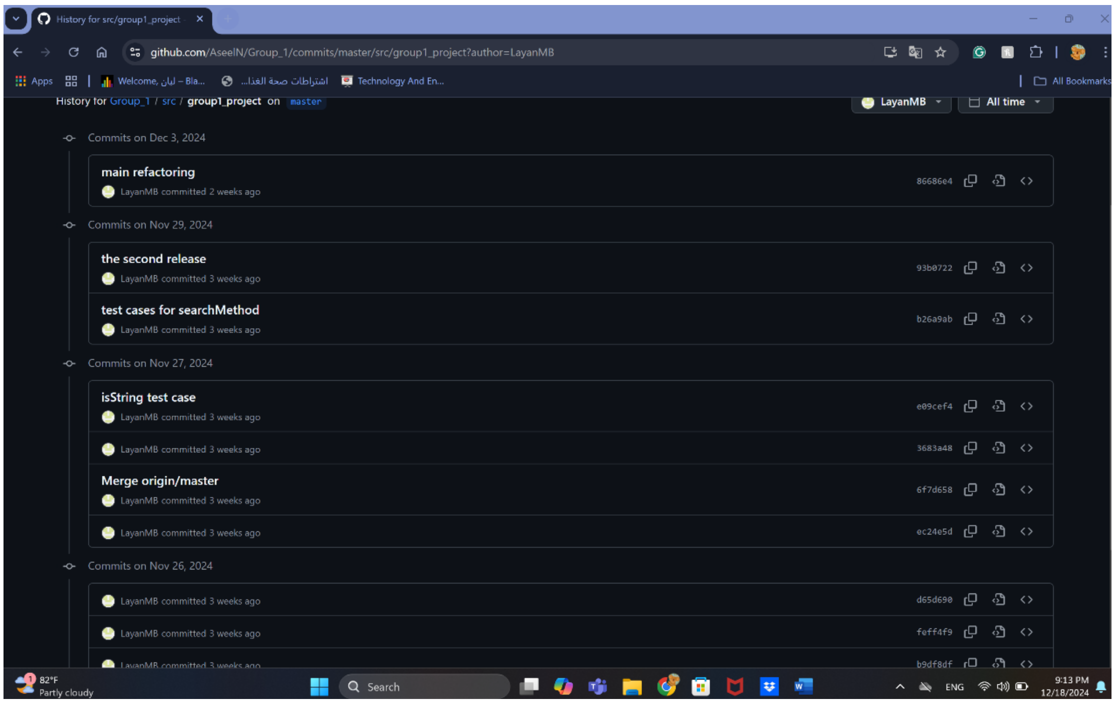
Commits on Dec 3, 2024

Refactoring provideFeedback method
AseelN committed 2 weeks ago — 36ca29c

Commits on Nov 30, 2024

Third release "Review comments for admin"
AseelN committed 3 weeks ago — 2e9942b

Figure 27- Aseel GitHub


History for src/group1_project

github.com/AseelN/Group_1/commits/master/src/group1_project?author=LayanMB

History for Group_1 / src / group1_project on master

LayanMB · All time

Commits on Dec 3, 2024

main refactoring
LayanMB committed 2 weeks ago · 8668 6e4

Commits on Nov 29, 2024

the second release
LayanMB committed 3 weeks ago · 93b0722

test cases for searchMethod
LayanMB committed 3 weeks ago · b26a9ab

Commits on Nov 27, 2024

isString test case
LayanMB committed 3 weeks ago · e09cef4

LayanMB committed 3 weeks ago · 3683a48

Merge origin/master
LayanMB committed 3 weeks ago · 6f7d658

LayanMB committed 3 weeks ago · ec24e5d

Commits on Nov 26, 2024

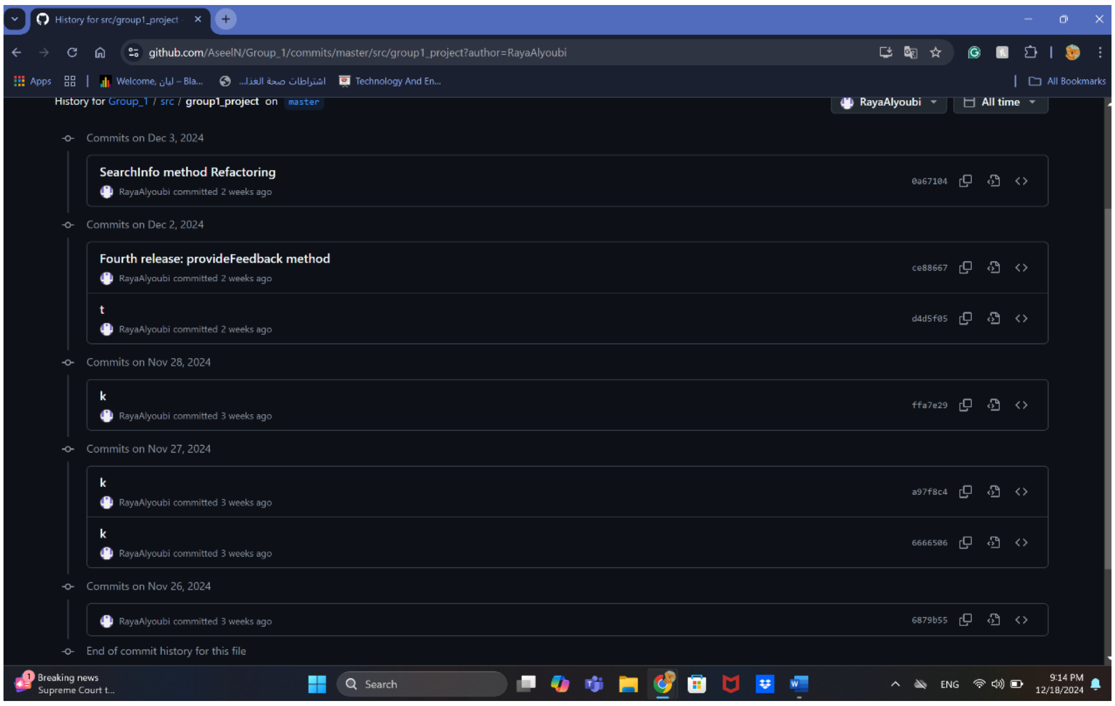
LayanMB committed 3 weeks ago · d65d690

LayanMB committed 3 weeks ago · feff4f9

LayanMB committed 3 weeks ago · b9df8df

82°F Partly cloudy · 9:13 PM 12/18/2024

Figure 28- Layan GitHub


History for src/group1_project

github.com/AseelN/Group_1/commits/master/src/group1_project?author=RayaAlyoubi

History for Group_1 / src / group1_project on master

RayaAlyoubi · All time

Commits on Dec 3, 2024

SearchInfo method Refactoring
RayaAlyoubi committed 2 weeks ago · 0a67104

Commits on Dec 2, 2024

Fourth release: provideFeedback method
RayaAlyoubi committed 2 weeks ago · ce88667

t
RayaAlyoubi committed 2 weeks ago · d4d5f05

Commits on Nov 28, 2024

k
RayaAlyoubi committed 3 weeks ago · ffa7e29

Commits on Nov 27, 2024

k
RayaAlyoubi committed 3 weeks ago · a97f8c4

k
RayaAlyoubi committed 3 weeks ago · 6666506

Commits on Nov 26, 2024

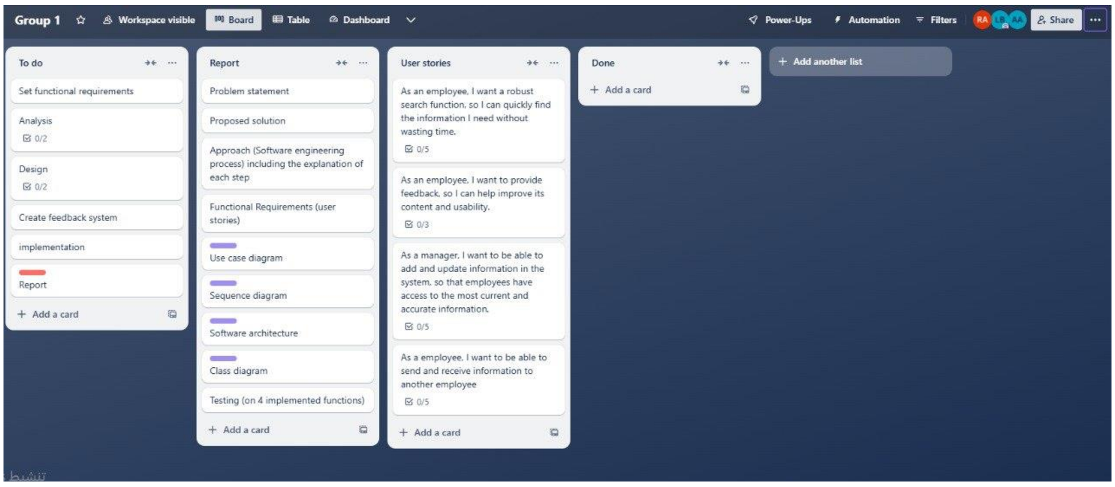
RayaAlyoubi committed 3 weeks ago · 6879b55

End of commit history for this file

Breaking news Supreme Court t... · 9:14 PM 12/18/2024

Figure 29- Raya GitHub

Trello



To do
- Set functional requirements
- Analysis ☑ 0/2
- Design ☑ 0/2
- Create feedback system
- implementation
- Report
- + Add a card

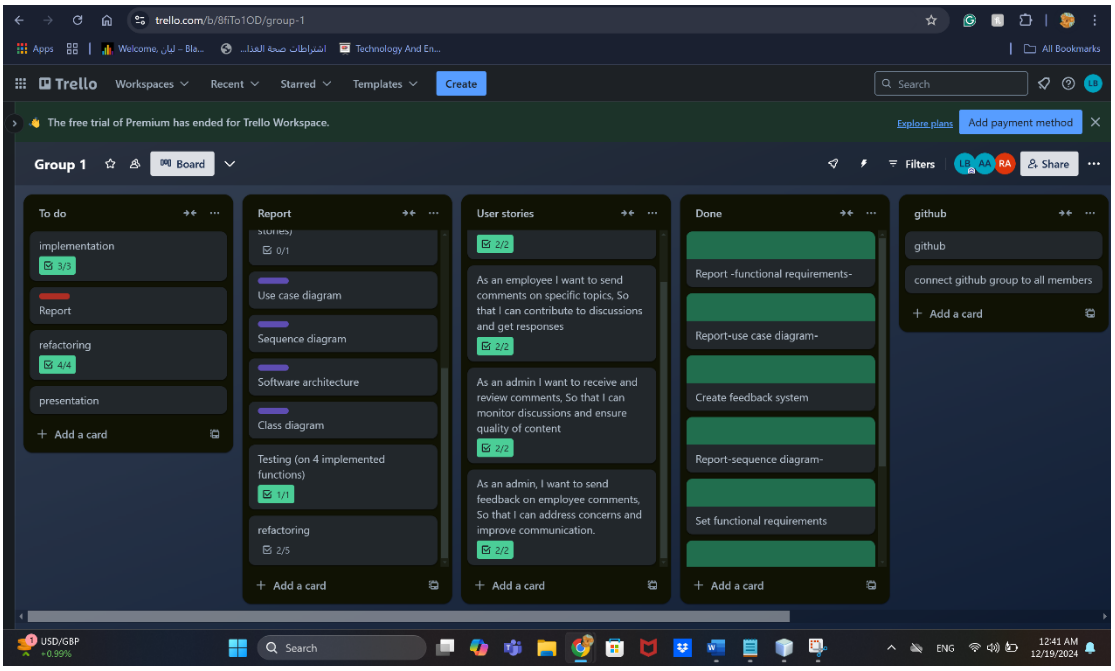
Report
- Problem statement
- Proposed solution
- Approach (Software engineering process) including the explanation of each step
- Functional Requirements (user stories)
- Use case diagram
- Sequence diagram
- Software architecture
- Class diagram
- Testing (on 4 implemented functions)
- + Add a card

User stories
- As an employee, I want a robust search function, so I can quickly find the information I need without wasting time. ☑ 0/5
- As an employee, I want to provide feedback, so I can help improve its content and usability. ☑ 0/3
- As a manager, I want to be able to add and update information in the system, so that employees have access to the most current and accurate information. ☑ 0/5
- As a employee, I want to be able to send and receive information to another employee ☑ 0/5
- + Add a card

Done
- + Add a card

+ Add another list

Figure 30- trello screenshot (1)



The free trial of Premium has ended for Trello Workspace.

To do
- implementation ☑ 3/3
- Report
- refactoring ☑ 4/4
- presentation
- + Add a card

Report
- stories) ☑ 0/1
- Use case diagram
- Sequence diagram
- Software architecture
- Class diagram
- Testing (on 4 implemented functions) ☑ 1/1
- refactoring ☑ 2/5
- + Add a card

User stories
- ☑ 2/2
- As an employee I want to send comments on specific topics, So that I can contribute to discussions and get responses ☑ 2/2
- As an admin I want to receive and review comments, So that I can monitor discussions and ensure quality of content ☑ 2/2
- As an admin, I want to send feedback on employee comments, So that I can address concerns and improve communication. ☑ 2/2
- + Add a card

Done
- Report -functional requirements-
- Report-use case diagram-
- Create feedback system
- Report-sequence diagram-
- Set functional requirements
- + Add a card

github
- github
- connect github group to all members
- + Add a card

Figure 31- trello screenshot (2)

Conclusion

The Information Management Program successfully resolves issues with inefficient information access and communication within the workplace. By implementing the project in multiple iterative releases, the team effectively addressed user needs while adhering to XP principles. The application's modular design and robust features provide a foundation for future enhancements.

Future Work

The application has potential for further development to enhance usability and scalability:

1. **Advanced Search Features:** Incorporate natural language processing (NLP) to allow users to search using conversational queries.
2. **Mobile Application:** Develop a mobile version of the application to increase accessibility for employees on the go.
3. **Analytics and Reporting:** Implement data analytics tools to generate insights from user activity and comments.
4. **Enhanced Security:** Integrate advanced authentication methods and encryption to ensure data confidentiality.
5. **AI-Powered Suggestions:** Leverage AI to recommend relevant information and prioritize comments requiring admin attention.

Links

Trello link:

https://trello.com/invite/b/671fd90bd39157836b7e9475/ATTI63c71dfa32724dc4405fc42cf20878573AB6095D/group-1

GitHub link:

https://github.com/AseelN/Group_1.git