UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

UNIVERSIDAD DISTRITAL FRANCISCO JOSÉ DE CALDAS
Faculty of Engineering
Systems Engineering Program

Technical report of the Desktop Note-Taking Application NoteHub

Technical Report

Authors:
Juan Carlos Córdoba Asprilla
Sebastián Camilo Sánchez Cárdenas

Supervisor:
Eng. Carlos Andrés Sierra

Bogotá D.C., Colombia
May 2025

Abstract

This report presents the design and development of a desktop note-taking application inspired by Notion (NoteHub), utilizing principles of object-oriented programming (OOP). The app supports note creation, task management, scheduling, and local reminders, offering a clean and intuitive interface. The project includes detailed requirements, user stories, class structures, CRC cards, and UML diagrams. The current implementation focuses on offline usage, local data persistence, and modular organization.

Keywords: Object-Oriented Programming, Notes App, Desktop Application, User Stories, CRC Cards, UML Diagrams, Local Storage, Technical Design, Educational Project.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This technical report presents the development of NoteHub, a desktop-based note-taking application inspired by Notion, designed with a focus on simplicity, offline capability, and core functionalities. Built using Java Swing and a layered architecture, the application enables users to manage notes, reminders, schedules, and calendar events without the complexities of cloud-based tools.

1.0.1 Background

Existing productivity tools often integrate extensive features that, while powerful, result in steep learning curves and excessive complexity for everyday users. Notion, one of the most prominent platforms, exemplifies this issue. NoteHub arises as a simplified alternative, focusing on usability, offline access, and essential organizational tools.

1.0.2 Problem Statement

There is a lack of lightweight, user-friendly desktop applications that consolidate basic productivity functionalities such as note-taking, reminders, schedules, and calendar annotations—without relying on internet connectivity or complex interfaces.

1.0.3 Aims and Objectives

The main aim is to design and implement a modular and maintainable desktop application that provides the following:

- Creation and management of notes and reminders.
- Customizable weekly schedules.
- A calendar interface for annotated events.
- Full offline functionality with local storage.

1.0.4 Solution Approach

NoteHub applies object-oriented programming principles and a three-layered software architecture:

- **Presentation Layer:** Java Swing-based GUI for user interaction.
- **Business Logic Layer:** Classes like `Note`, `Reminder`, and `Calendar` to encapsulate core logic.
- **Data Access Layer:** File-based storage system using plain text files (.txt).

Layer Separation for Maintainability

The separation of concerns ensures easier maintenance, scalability, and testing.

Offline-Centric Design

All functionalities are supported without internet connectivity, providing privacy and reliability.

1.0.5 Summary of Contributions and Achievements

This report outlines the architectural design, implementation steps, and core modules of NoteHub. It details how Java Swing and OOP were applied to create a functional, extensible desktop application with persistent local data management.

1.0.6 Organization of the Report

The report is structured as follows:

- **Chapter 1** introduces the context, problem, and proposed solution.
- **Chapter 2** presents the literature review and related work.
- **Chapter 3** outlines the methodology including requirements, architecture, and implementation process.
- **Chapter 4** describes the results of the development and implementation.
- **Chapter 5** provides discussion and analysis of findings.
- **Chapter 6** presents the conclusions and suggests future work.
- **Chapter 7** offers a personal reflection on the development experience.
- **Appendices** include supplementary materials such as user stories and CRC cards.

Chapter 2

Literature Review

The development of NoteHub is grounded in established principles and tools from object-oriented programming (OOP), user interface design, and software architecture. This section presents an overview of the foundational concepts and tools consulted during the design and implementation of the application.

2.0.1 Object-Oriented Programming and SOLID Principles

Object-Oriented Programming (OOP) is central to NoteHub's structure, promoting modularity and reusability. The five SOLID principles—Single Responsibility, Open/Closed, Liskov Substitution, Interface Segregation, and Dependency Inversion—were essential in designing maintainable and scalable components. Each class in NoteHub, such as `Note`, `Reminder`, and `Calendar`, adheres to one clear responsibility, allowing for easier extension and minimal modification of existing codebases.

According to [1], these principles reduce code fragility and coupling, fostering robustness in evolving systems.

2.0.2 Concurrency and Persistence in OOP Systems

Effective data persistence is a key requirement for desktop applications without cloud synchronization. As noted by [2], persistent storage mechanisms in OOP systems should abstract away file-handling complexity, enabling the rest of the application to focus on business logic. NoteHub achieves this separation through a dedicated data access layer, where content is stored and retrieved from plain-text files.

2.0.3 GUI Development with Java Swing

The graphical user interface (GUI) of NoteHub is built using Java Swing, a mature library for developing desktop applications. It supports components like `JFrame`, `JTabbedPane`, `JTextArea`, and `JButton`, which are used to manage user interactions. Oracle's official Java Swing [3] was instrumental in designing the interface, ensuring responsiveness and usability.

2.0.4 UML for System Design

Unified Modeling Language (UML) diagrams were used during the initial planning phase to visualize class structures and module interactions. Tools such as [4] facilitated the generation of class and package diagrams that provided a clear overview of system architecture. These diagrams supported the alignment of implementation with the design and enhanced communication among team members.

2.0.5 Summary

The literature consulted ensured that NoteHub was built upon a solid theoretical and practical foundation. From OOP principles to GUI toolkits and persistence mechanisms, each source informed a specific part of the system's architecture or implementation strategy.

Chapter 3

Methodology

The development of NoteHub followed an lineal and structured methodology, combining object-oriented analysis and design (OOAD), agile principles, and layered software architecture. The process was divided into four stages, each with a specific focus and deliverables.

3.0.1 Requirements Analysis

The initial stage focused on identifying user needs through **user stories**, defining clear **functional and non-functional requirements**, and setting the **project objectives**.

The user stories were defined based on typical personas, such as students, event planners, and managers. Each story included acceptance criteria to ensure the desired functionality was clear and testable. For example, the story *'As a student, I want to create a note to remember university tasks"* resulted in the note creation module. Similarly, *'As a runner, I want to create reminders to check my leg"* led to the implementation of scheduled reminders.

3.0.2 Object-Oriented Design with CRC Cards and UML

Once requirements were gathered, the system was modeled using **CRC Cards (Class–Responsibility–Collab** to define the primary classes:

- **Application**: Manages the graphical interface and overall control flow.
- **Notes / Reminder**: Represent content types and implement shared interface.
- **Content Interface**: Defines common behavior for all content-based classes.
- **Calendar**: Handles event storage and updates.

These responsibilities were used to design a **UML Class Diagram** showing inheritance and relationships.

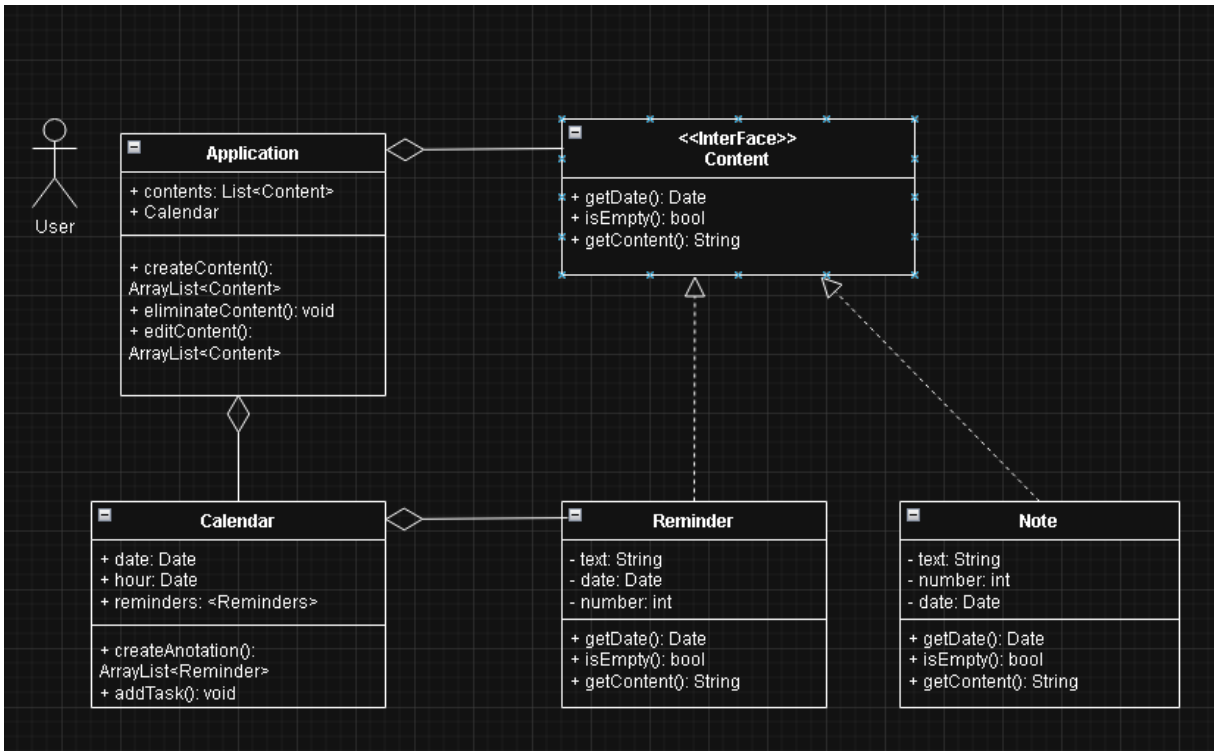


Figure 3.1: UML CLASS DIAGRAM

Sequence diagrams were created to demonstrate user interactions over time—such as creating a note or saving reminders.

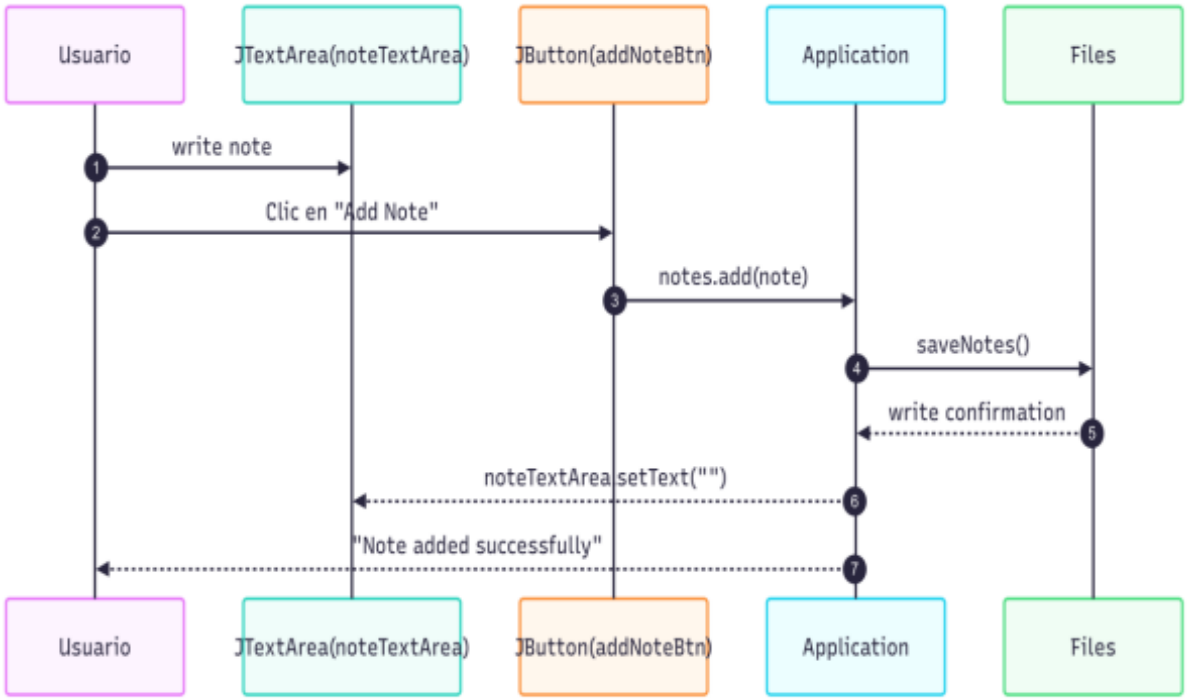


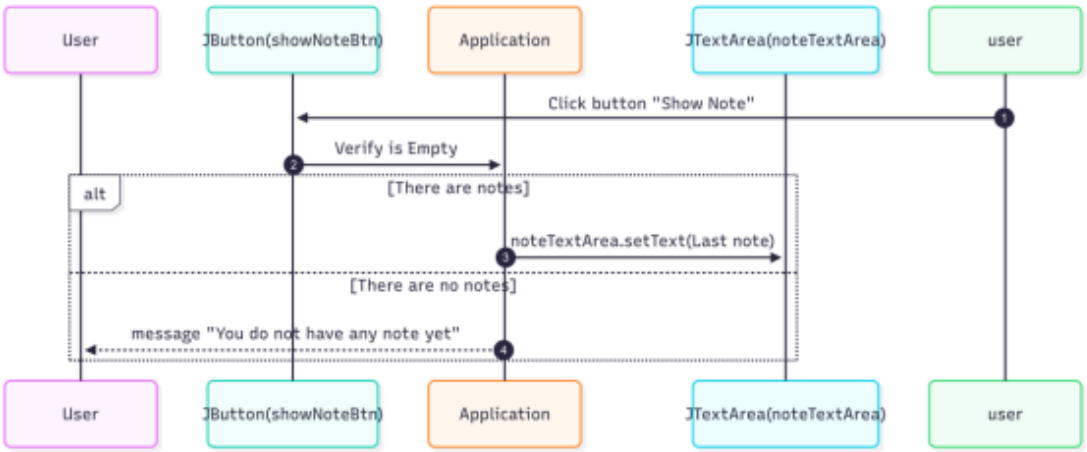
Figure 3.2: Sequence Diagram Create note

Sequence

1. The user writes a note in the text area.

2. The user clicks the `"Add Note"` button.
3. The note is added to the notes list in the `Application`.
4. The `saveNotes()` function is called to write the updated list to the `notes.txt` file.
5. If saved successfully:
   (a) The text area is cleared using `noteTextArea.setText("")`.
   (b) A message is displayed: `"Note added successfully"`.

Sequence diagram Show Notes



Sequence:

1. The user clicks the **"Show Note"** button.
2. The system checks if the `notes` list is not empty.
   If there are notes:
   The last note is displayed in the text area.
   If not:
   A message appears: **"You do not have any note yet"**.

Figure 3.3: Sequence Diagram Show Notes

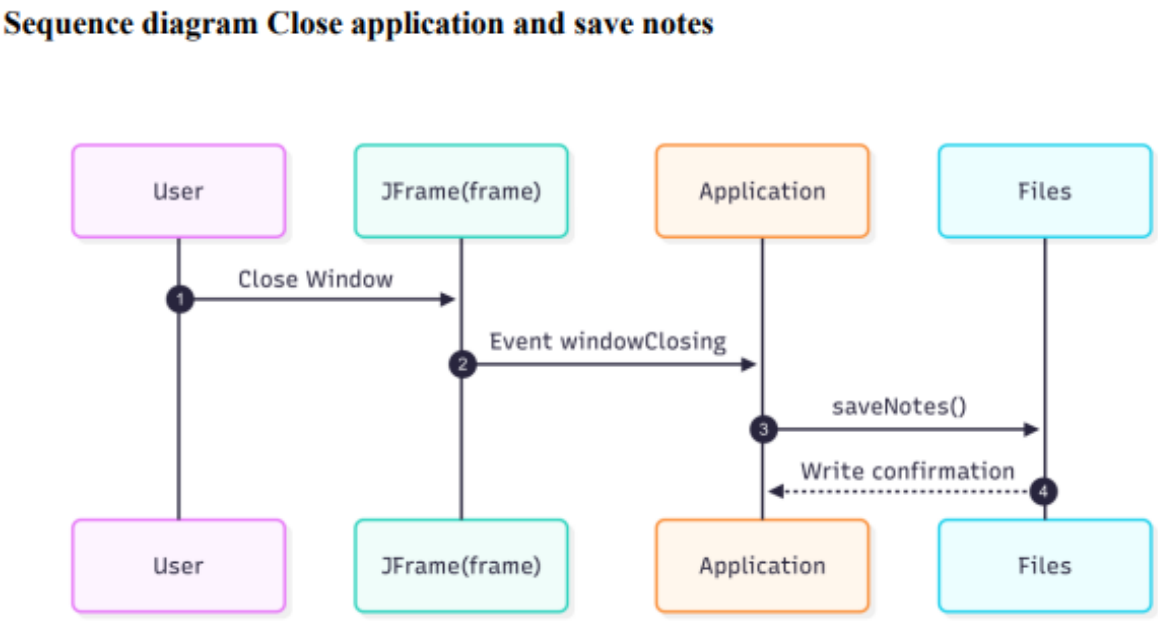


Figure 3.4: Sequence Diagram Close Application

3.0.3 Layered Architecture Implementation

The software was developed with a **three-layer architecture** to support scalability and maintainability:

- **Presentation Layer**: Built with Java Swing. The `Application` class manages a GUI using `JFrame`, `JTabbedPane`, `JTextArea`, and buttons. Users interact through tabs labeled "Notes" , "Calendar", Schedule" "Reminders".
- **Business Logic Layer**: Classes `Notes`, `Reminder`, and `Calendar` encapsulate logic for handling user content. All implement the `Content` interface.
- **Data Access Layer**: Manages persistent storage using plain text files (`notes.txt`, `reminders.txt`).

3.0.4 Tooling and Development Environment

The project was developed using:

- **Language**: Java
- **IDE**: Visual Studio Code
- **Libraries**: Java Swing (GUI), java.util (Dates), java.nio.file (Persistence)

- **Version Control**: Git and GitHub (repository: BreezeTask)
- **Documentation Tools**: Drawio (UML), PlantUML, Canva, Word

3.0.5 Use of SOLID Principles

The system was designed around the SOLID principles to ensure clean architecture:

- **SRP**: Each class handles only one responsibility (e.g., `Note` stores notes).
- **OCP**: New classes (like `Task`) can be added without modifying existing code.
- **LSP**: `Notes` and `Reminder` replace `Content` seamlessly.
- **ISP**: The `Content` interface only includes essential methods.
- **DIP**: `Application` depends on the abstraction `Content`, not concrete types.

3.0.6 Coding Practices and Persistence

Each content object is stored in memory and persisted locally:

- `saveNotes()` writes the notes list to `notes.txt`.
- `loadNotes()` initializes the app with saved content.
- Similar logic exists for reminders, stored in `reminders.txt`.

GUI actions are connected to logic through event listeners. For example, clicking `Add Note` triggers a listener that appends the content to the list, saves it, and displays a confirmation message.

3.0.7 Summary

This methodology ensured systematic development through requirements gathering, object-oriented modeling, SOLID-compliant class design, layered architecture implementation, and effective use of Java tooling. It allowed to focus on incremental improvements, verifiable features, and separation of responsibilities while ensuring long-term extensibility and code clarity.

Chapter 4

Results

The development of the NoteHub application has generated several visible results reflected in the project:

4.1 Functional Results

- A graphical user interface (GUI) with Java Swing that allows the creation of notes, reminders, a calendar, and a schedule builder.
- Tabs and panels that organize the UI into manageable components for notes, reminders, the calendar, and schedules.
- Buttons and input areas configured with listeners to allow user interaction with the system.
- Event-based control logic that triggers file saving and updates the UI accordingly.

4.2 Architecture and Design Results

- A layered architecture that separates the GUI (presentation), business logic (domain classes), and data persistence (file storage).

4.3 Data Persistence and Handling

- Storing Notes and Reminders
- Correct retrieval and display of saved data upon application startup.
- The archive files include: `notes.txt` and `reminders.txt`.

4.4 Summary

The current implementation validates the viability of NoteHub's architecture and use cases. While not all modules are graphically finalized as originally envisioned, the foundation has been laid for a functional and extensible desktop application.

Chapter 5

Discussion and Analysis

This chapter provides a reflective assessment of the results obtained during the development of the NoteHub desktop application. The evaluation considers architectural design choices, implementation challenges, significance of outcomes, and recognized limitations.

5.1 Evaluation of Architecture and Design

The chosen layered architecture enabled clear separation of responsibilities across the GUI, logic, and data layers. This approach simplified debugging, testing, and future scalability. The use of object-oriented principles enhanced the modularity and clarity of each component:

- The `Application` class focuses on GUI control and navigation.
- Logic classes such as `Note` and `Reminder` implement content behaviors through a shared interface.
- Data access is encapsulated in file-based methods, allowing persistence without coupling logic.

These decisions align with best practices in object-oriented development and reinforce maintainability.

5.2 Significance of the Findings

The implemented functionalities address the problem defined in the early stages—providing a lightweight, offline, and intuitive tool. The system meets the key goals:

- Supports note creation and reminders , calendar and schedules manage through a user-friendly interface.
- Ensures persistence using local text files for data continuity.
- Demonstrates SOLID principles in practice, showing extensibility and clarity.

The use of CRC cards, user stories, and UML diagrams contributed to a structured development process. These tools allowed better collaboration, modular thinking, and clearer planning of the system architecture.

5.3 Limitations

While the system successfully meets core goals, there are areas where improvements are required:

- **Limited functionality:** The current version lacks calendar advanced task features (e.g., priorities, deadlines).
- **GUI aesthetics:** Java Swing, although functional, does not provide modern UI components without extensive customization.
- **Persistence layer:** File-based text storage works for basic needs, but lacks support for structured querying or complex data relationships. Migration to JSON or SQLite could be considered.
- **Scalability:** Current architecture supports desktop use only. For mobile compatibility, significant redesign would be necessary.

5.4 Summary

This chapter highlighted how architectural decisions, design methodology, and programming principles shaped the results of the project. The implemented system demonstrates the effectiveness of applying object-oriented principles and layered architecture to a real-world productivity tool. The recognized limitations offer opportunities for future development and refinement. The approach used serves as a foundation for building more complex, user-centered desktop applications.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

The project achieved its primary objectives:

- Developed a functional GUI using Java Swing.
- Implemented content classes (`Note`, `Reminder`) adhering to the `Content` interface.
- Created a persistence mechanism using local text files.
- Applied SOLID principles to promote clean architecture.

The design and implementation process was supported by thorough documentation including UML diagrams, CRC cards, and user stories. These tools facilitated consistent planning, communication, and problem-solving throughout the development cycle.

Overall, NoteHub represents a successful demonstration of building a modular desktop application from scratch using OOP and structured methodologies. Although limited in certain aspects, it establishes a robust foundation for future enhancements and potential cross-platform expansion.

6.2 Future Work

Based on the current implementation and the limitations identified in earlier chapters, several areas for future development have been outlined:

- **Enhanced Calendar UI:** Replace the current event-handling structure with a visual calendar grid allowing users to navigate between days and months interactively.
- **Task Prioritization and Deadlines:** Introduce new features for managing task deadlines, priorities, and recurring reminders.
- **Improved Aesthetics and Responsiveness:** Upgrade the GUI using more modern UI frameworks or integrate design libraries for better usability.
- **Data Migration to JSON or SQLite:** Transition from basic text files to more robust storage formats like JSON or embedded databases, enabling better data querying and reliability.
- **Cross-platform Compatibility:** Explore the feasibility of porting the application to other platforms such as Android or web-based environments.

These enhancements would significantly improve user experience and system performance, while maintaining the core philosophy of simplicity and offline independence.

In conclusion, NoteHub has laid the groundwork for a fully featured personal information manager and offers numerous opportunities for educational and technical expansion in future iterations.

Chapter 7

Reflection

The development of NoteHub has been a transformative learning experience, both technically and personally. Beyond simply implementing a note-taking application, the project offered us the opportunity to explore structured problem-solving and to apply object-oriented design in a meaningful way.

Throughout this process, we gained a deeper appreciation for software architecture, especially the role of layered design in separating responsibilities and enhancing maintainability. we learned how to define and adhere to class responsibilities using CRC cards, and how to capture user intent through clearly written user stories and acceptance criteria. These tools provided clarity during development and supported better code structure and planning.

One of the most valuable lessons was the experience of working with SOLID principles. Applying these principles allowed me to maintain cleaner, more modular code, and made it easier to introduce changes without breaking existing functionality. The decision-making process for defining class responsibilities, dependencies, and interfaces helped sharpen us analytical thinking in system design.

From a practical standpoint, we developed familiarity with Java Swing for desktop application development. While we had initial challenges understanding its layout management and event handling, overcoming them provided a greater sense of competence and flexibility with GUI development. we also improved us proficiency in LaTeX for report writing, Git for version control, and tools like Draw.io for diagramming.

Not all aspects were smooth. A notable challenge was maintaining UI flexibility while keeping the architecture decoupled. Also, the reliance on plain text files for data persistence proved limiting when attempting to expand the app's complexity. If we were to restart this project, we would consider integrating a lightweight database from the beginning and dedicating more time to refining the user experience.

Reflecting on us initial objectives, we realize some goals evolved than we understanding deepened. While the system started as a simple notes app, it matured into a platform that has a coplex internal desing that cost hours and hours of brainrot for us.

Overall, the NoteHub project has been instrumental in consolidating us understanding of object-oriented design, user-centered development, and structured software engineering processes. it helped us think more about the project design and what users want from us before sitting down to program without knowing what we're going to do. We realized that sometimes thinking things through before doing them is much better than acting quickly without a clear direction.

Acknowledgments

We thank Professor Carlos Andrés Sierra for his guidance throughout the development process.

Glossary

OOP (Object-Oriented Programming)
A programming paradigm based on the concept of "objects," which can contain data (attributes) and methods (functions).

Class
A blueprint for creating objects, defining attributes and methods that characterize a type of object.

Object
An instance of a class containing real values for the properties defined in the class.

Encapsulation
OOP principle that restricts direct access to an object's data, allowing access only through public methods.

Inheritance
OOP feature that allows a class to inherit attributes and methods from another class.

Polymorphism
OOP concept that allows different classes to be treated as instances of the same class through shared interfaces or methods.

UML (Unified Modeling Language)
A standardized modeling language used to visualize and design software systems.

CRC Card
A design tool used to represent the Responsibilities and Collaborators of software classes.

Mockup
A static visual prototype that illustrates the user interface design of an application.

Reminder
A timed notification that alerts the user of a scheduled event or task.

Local Storage
Mechanism for saving data directly on the device without requiring internet access.

Note
A digital record containing user-entered content such as text, lists, and labels.

Task
A type of note oriented toward actions with completion status and priority.

Schedule
The organization of activities and class hours based on a weekly calendar.

Bibliography

[1] freeCodeCamp. *Los principios SOLID explicados en español.* Disponible en: `https://www.freecodecamp.org/espanol/news/los-principios-solid-explicados-en-espanol/`

[2] Styde.net. *Concurrencia y persistencia en programación orientada a objetos.* Disponible en: `https://styde.net/concurrencia-y-persistencia-en-programacion-orientada-a-objetos/`

[3] Oracle. *The Java^{TM} Tutorials – Creating a GUI With Swing.* Disponible en: `https://docs.oracle.com/javase/tutorial/uiswing/`

[4] draw.io. *UML Class Diagrams – draw.io Blog.* Disponible en: `https://www.drawio.com/blog/uml-class-diagrams`

Appendix A

User Stories

The following user stories were designed to reflect real user interactions and system expectations. Each story includes a persona, a goal, and acceptance criteria.

Table A.1: User Story - Student Work

Actor	Student
Title	Student Work
Priority	High
I want to...	Create a note about an activity they left at my university
So that...	I can remember to do it
Acceptance Criteria	When I enter a title and content for the new note, the note must be saved correctly and must be visible in its respective place.

Table A.2: User Story - Remind

Actor	Runner
Title	Remind
Priority	High
I want to...	Create a reminder
So that...	I remember to check my leg after a marathon
Acceptance Criteria	When I check the reminder as complete, it should disappear.

Table A.3: User Story - Event

Actor	Event Planner
Title	Event
Priority	High
I want to...	Remove certain reminder note
So that...	I can organize my future events
Acceptance Criteria	When I select a note or reminder and confirm deletion, the item should no longer appear in the app.

Table A.4: User Story - Happy Birthday

Actor	Boss
Title	Happy Birthday
Priority	Low - Mid
I want to...	Put payment dates for employees on the annual calendar
So that...	I can prepare their payments on time
Acceptance Criteria	When I log out and back into the app, the dates I wrote down should still be visible.

Table A.5: User Story - Reminders

Actor	Event Planner
Title	Reminders
Priority	High
I want to...	Re-program a postponed event
So that...	I remember the new date I need to attend
Acceptance Criteria	When I update the reminder, it should reflect the last edited time.

Table A.6: User Story - Events

Actor	User
Title	Events
Priority	Mid - High
I want to...	Mark special dates or events on the calendar
So that...	I do not forget important dates
Acceptance Criteria	When I enter a date and description, the event should be saved and visible on the calendar.

Table A.7: User Story - Schedule

Actor	Student
Title	Schedule
Priority	High
I want to...	Create my class schedule
So that...	I can remember my class times
Acceptance Criteria	When I enter schedule data, it should be reflected in the weekly calendar and persist across sessions.

Appendix B

CRC Cards

Table B.1: CRC Card - Application

Responsibilities	Display graphical interface, manage note and reminders, handle user actions
Collaborators	Note, Reminder, Calendar ,Content

Table B.2: CRC Card - Note

Responsibilities	Store note content (text, ID, date), implement Content interface, validate fields
Collaborators	Application , Content

Table B.3: CRC Card - Reminder

Responsibilities	Store and update reminder events with timestamps, implement Content interface
Collaborators	Calendar, Application, content

Table B.4: CRC Card - Calendar

Responsibilities	Display monthly view of events, interact with reminders
Collaborators	Reminder, Application

Table B.5: CRC Card - Content Interface

Responsibilities	Define shared methods like getContent and getDate for note and reminders
Collaborators	Notes, Reminder