

Expense Splitter

A PROJECT REPORT

Submitted by

Ankur Tyagi (22BCS13316)

Yuvraj Singh (22BCS14956)

Shubham (22BCS14896)

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE ENGINEERING



Chandigarh University

April 2025



BONAFIDE CERTIFICATE

Certified that this project report “Expense Splitter” is the bonafide work of **Ankur Tyagi , Yuvraj Singh , Shubham ”** who carried out the project work under my/our supervision.

SIGNATURE

Er. Mupnesh Kumari

SUPERVISOR

TABLE OF CONTENTS

| | |
|--|-----------|
| List of Figures..... | 5 |
| Abstract..... | 6 |
| Chapter 1. Introduction..... | 7 |
| 1.1 Client Identification/Need Identification..... | 7 |
| 1.2 Identification of Problem..... | 8 |
| 1.3 Identification of Task..... | 9 |
| 1.4 Timeline..... | 10 |
| 1.5 Organization of the report..... | 11 |
| Chapter 2. Literature Review | 13 |
| 2.1 Timeline of the reported problem..... | 13 |
| 2.2 Existing Solutions..... | 13 |
| 2.3 Bibliometric Analysis | 14 |
| 2.4 Critical Review | 15 |
| 2.5 Problem Definition..... | 15 |
| 2.6 Goals and Objectives..... | 16 |
| Chapter 3. Design Flow/Process | 17 |
| 3.1 Evaluation of Specifications/Features..... | 17 |
| 3.2 Design Constraints..... | 18 |
| 3.3 Feature Finalization..... | 19 |
| 3.4 Design Flow | 20 |
| 3.5 Design Selection | 21 |
| 3.6 Implementation plan/Methodology..... | 21 |
| Chapter 4. Result Analysis and Validation | 23 |

| | |
|--|-----------|
| 4.1 Usability Assessment | 23 |
| 4.2 Functional Evaluation..... | 23 |
| 4.3 Performance Assessment | 24 |
| 4.4 Security..... | 24 |
| 4.5 Result..... | 24 |
| Chapter 4. Conclusion and Future Work | 26 |
| 4.1 Conclusion | 26 |
| 4.2 Future Work | 26 |

List of Figures

| | |
|-----------------------|-----------|
| Figure 1..... | 22 |
| Figure 2 | 25 |

ABSTRACT

In group settings such as shared apartments, trips, or team activities, managing and dividing expenses often becomes a cumbersome task. Manual calculations, forgotten contributions, and lack of accountability can lead to confusion and disputes among members. To address this real-world problem, the Expense Splitter Finance application was developed—a desktop-based solution built using Java, JavaFX for GUI, and SQLite for local data storage. This application simplifies the process of splitting shared expenses among multiple individuals by automating calculations, tracking contributions, and generating clear reports. With a user-friendly interface, real-time validations, and persistent data storage, it ensures accuracy, transparency, and ease of use for all users. The system allows users to enter a total expense, specify the number of people involved, and instantly compute each individual's share. It also stores transaction history and produces detailed summaries for better financial accountability. The application was thoroughly tested for usability, functionality, performance, and stability. Results show that the system performs reliably under various scenarios and significantly improves the group expense-sharing experience. This project not only provides a practical utility but also serves as a foundation for further enhancement, such as cloud integration, individual tracking, and multi-platform support in future iterations.

CHAPTER 1.

INTRODUCTION

1.1. Client Identification/Need Identification/Identification of relevant Contemporary issue

1.1.1 Client Identification

The primary clients for the Expense Splitter Finance project are:

- Students and Roommates who often share rent, groceries, utilities, and leisure expenses.
- Friends traveling together who need to split costs for transportation, accommodation, and food.
- Colleagues or Teams participating in joint ventures, events, or office celebrations with shared expenditures.

These users typically lack a convenient and efficient tool to manage and track shared expenses in real-time, leading to confusion, debt mismanagement, or disputes. They require a solution that automates calculations, minimizes manual record-keeping, and maintains transparency in financial contributions.

1.1.2 Need Identification

The need for a tool like Expense Splitter arises due to:

- Frequent group activities: As social interactions and collaborations increase, shared expenses are becoming a routine part of life.
- Manual tracking limitations: People often rely on memory, notepads, or basic calculators to track who owes what, which is inefficient and error-prone.

- Transparency and accountability: Users want an unbiased way to track contributions and outstanding payments, helping avoid misunderstandings or conflicts in personal and professional circles.
- Automation: A digital solution that calculates splits and updates in real-time reduces both time and effort in financial management.
- Record keeping: Keeping a history of expenses helps in auditing past transactions and maintaining financial discipline.

1.1.3 Identification of Relevant Contemporary Issues

In today's increasingly digital and collaborative world, managing shared expenses has become a growing challenge. With digital payments becoming the norm, people expect digital tools to manage their finances as well. Co-living and co-working spaces are increasing, intensifying the need for shared financial tools. Following the COVID-19 pandemic, individuals have become more conscious about expenses and budgeting. Managing group spending effectively has gained more importance.

Many expense management tools are web/mobile-based and require constant internet access. There is a growing demand for desktop-based applications, especially in semi-urban areas or institutions with limited connectivity. Users prefer offline or local database solutions due to growing data privacy concerns with cloud-based services.

1.2. Identification of Problem

Manual expense tracking is time-consuming and error-prone. In group settings, people often forget to record contributions, which leads to an unequal burden on some members. Without a centralized system, reconciling expenses becomes complex and frustrating, resulting in potential conflicts among group members. The lack of detailed reporting and tracking mechanisms further complicates accountability. Additionally, the process of reconciling expenses at the end of a trip or event becomes unnecessarily complex. Multiple small transactions—like shared meals, cab fares, groceries, or utility bills—accumulate quickly, and without a centralized platform, compiling these expenses and dividing them fairly is a source of confusion and frustration. As a result, personal

relationships can be strained due to disputes over unclear or disputed transactions. Moreover, most existing digital solutions are either too complicated, rely on constant internet access, or are mobile-only, which excludes a significant portion of users who prefer or require a desktop-based application with offline capabilities. This gap further emphasizes the need for a lightweight, user-friendly desktop solution that streamlines the entire expense-sharing process while offering transparency, fairness, and accuracy.

1.3. Identification of Tasks

The development of the Expense Splitter Finance application involves a series of clearly defined tasks that span across requirement analysis, design, implementation, testing, and deployment. Each task plays a critical role in building a reliable, user-friendly, and efficient expense management tool for groups. The key tasks are described below:

1.3.1 Requirement Gathering and Analysis

The first step in the development process involves understanding the needs of the end-users. This includes:

- Identifying the primary problems faced by people in managing group expenses.
- Studying user expectations regarding functionality, user interface, offline support, and performance.
- Researching existing applications to identify gaps and potential areas for improvement.
- Finalizing the core features such as expense input, real-time calculation of shares, payment tracking, and report generation.

This phase lays the foundation of the project by ensuring that the application is tailored to solve real-world problems effectively.

1.3.2 Designing the User Interface

CreateThe next step is to design a clean, intuitive, and interactive user interface using JavaFX, which is known for building visually rich desktop applications. The UI must be minimalistic yet functional to ensure a smooth user experience for all types of users.

1.3.3 Setting up the SQLite Database for Persistent Data Storage

To ensure that user data is stored securely and remains accessible across sessions, SQLite is used as the local database. The tasks under this include:

- Designing the database schema to store user profiles, transactions, and payment records.
- Establishing database connectivity using JDBC (Java Database Connectivity).

1.3.4 Model Training and Validation

This core functionality involves writing the business logic to:

- Calculate each person's share from the total expense.
- Handle dynamic user input (e.g., varying number of participants and amounts).
- Track individual contributions and compute who owes what to whom.

1.3.5 Comparison with Baseline Algorithms

After the core features are developed, extensive testing is required to ensure the application is:

- Functional: All features work as expected with valid and invalid inputs.
- User-friendly: The interface behaves intuitively, with proper error handling and feedback.
- Stable: No crashes or data loss occur during extended usage.

1.4. Timeline

Table 1: Timeline of the project

| Task | Duration | Start Date | End Date | Details |
|----------------------|-----------------|-------------------|-----------------|--|
| Literature Review | 4 days | Mar 25, 2025 | Mar 29, 2025 | In-depth study of related work, existing models, and identification of gaps. |
| Dataset Collection | 4 days | Mar 29, 2025 | Apr 2, 2025 | Collect required data. |
| Model Development | 5 days | Apr 3, 2025 | Apr 9, 2025 | Design and develop the website. |
| Testing & Validation | 2 days | Apr 9, 2025 | Apr 11, 2025 | Final phase testing of model output validating predictions. |
| Report Writing | 5 days | Apr 9, 2025 | Apr 14, 2025 | Compile findings, analysis, methodology, results, and conclusions into a formal research report. |

1.5.Organization of the Report

1.5.1 Chapter 1: Introduction

This chapter introduces the background and motivation behind the research. Introduces the core concept of the project, defines the target users and the real-world problem being

solved, outlines the major tasks involved, and presents the project timeline. The chapter also defines the research problem, objectives, scope of the project, a timeline of tasks, and outlines how the report is structured.

1.5.2 Chapter 2: Literature Review

This chapter presents a detailed review of existing work in the field of expense tracking, Reviews existing solutions and related research, identifies gaps in current systems, and defines the specific goals and objectives of this project. It also analyzes gaps in the current technologies, particularly in handling real-time, multi-source data in this domain.

1.5.3 Chapter 3: Design Methodology and Implementation

Chapter 3 describes the design methodology adopted in this research, including the system architecture of the proposed model. Details the development process from requirement analysis to system design, implementation, testing, and deployment. It also includes key code segments relevant to functionality.

1.5.4 Chapter 4: Results Analysis and Validation

This chapter presents the outcomes of the implemented model. Evaluates the effectiveness of the application in terms of usability, performance, and security. This chapter compares actual outcomes with the initial objectives.

1.5.5 Chapter 5: Conclusion and Future Work

The final chapter summarizes the key findings and contributions of the research. Summarizes the project, reflects on what was achieved, and discusses potential improvements and extensions for future development.

CHAPTER 2.

LITERATURE REVIEW/BACKGROUND STUDY

2.1. Timeline of the Reported Problem

The challenge of managing shared expenses is not new. With the rise of communal living arrangements, frequent group travels, and collaborative workspaces, the problem of fair expense distribution has gained prominence over the past two decades. Traditionally, expense tracking was done manually or through spreadsheets. However, as financial interactions became more dynamic, the demand for automated tools increased, especially post-2010 when smartphones and personal computing became more accessible.

In recent years, especially after the COVID-19 pandemic, there has been an increase in co-living setups, virtual collaborations, and remote travel plans. These trends have intensified the need for tools that can efficiently manage group expenses in both online and offline environments. Furthermore, with growing concerns about data privacy and internet dependency, there's renewed interest in offline, desktop-based finance solutions that give users full control over their data.

2.2. Existing Solutions

There are several existing applications that address expense sharing and management:

- Splitwise – One of the most popular apps for splitting bills among friends. It offers mobile and web platforms but relies on internet connectivity and cloud storage.
- Settle Up – Allows groups to track shared expenses and debts. It is available on mobile devices and provides synchronization features, but its functionality is limited in the offline mode.
- Tricount – Primarily used by travelers, it simplifies expense splitting but lacks detailed

reporting and desktop accessibility.

- Excel/Google Sheets – Still widely used for expense tracking, but they require manual entry, formulas, and offer no automation for payment tracking or notifications.

Despite the availability of these tools, several limitations persist:

- Internet dependency
- Lack of detailed reporting and payment tracking
- Complexity in use for non-tech-savvy users
- Limited or no desktop/laptop application support

This reveals a significant gap in offline-capable, desktop-based, simple yet feature-rich solutions tailored for group finance management.

2.3. Bibliometric Analysis

To better understand the research and development around expense tracking applications, a bibliometric analysis was conducted through digital libraries such as IEEE Xplore, Google Scholar, and ACM Digital Library. Key findings include:

- There is a growing interest in personal finance apps that are offline-capable and use local databases like SQLite.
- Most research papers emphasize cloud-based solutions, while only a few focus on offline applications using Java and desktop environments.
- JavaFX is increasingly being adopted in academic projects for its ease of GUI development in desktop applications.
- SQLite is considered ideal for lightweight, embedded database applications due to its reliability and simplicity.

The analysis confirms that while mobile finance tools dominate the current market, there is

still an underserved niche for secure, local, desktop-based expense management tools, particularly for educational or small-group contexts.

2.4. Critical Review

Based on existing solutions and academic literature, the following gaps are identified:

- Lack of offline solutions: Most apps require internet access, limiting usability in remote or low-connectivity areas.
- No full desktop support: The majority of platforms are web/mobile-based, ignoring users who prefer or require desktop interfaces.
- Limited transparency in reporting: Users often lack clear, downloadable summaries or real-time insights into group balances.
- Dependence on cloud storage: This raises privacy concerns and may exclude users with limited data access.

These observations support the development of an application like Expense Splitter Finance that focuses on offline usability, desktop interface, transparency, and simple user experience.

2.5. Problem Definition

Based on the identified gaps and user needs, the core problem can be defined as:

“There is a lack of an intuitive, secure, and offline desktop application for managing shared expenses in group scenarios that offers detailed reporting, transparency, and persistent storage without requiring internet connectivity.”

This problem definition is at the heart of the Expense Splitter Finance project.

2.6. Goals and Objectives

The primary objectives of the Expense Splitter Finance project are:

- To design and develop a desktop-based application using Java and JavaFX for intuitive interaction.
- To provide offline functionality using SQLite for local data storage.
- To implement real-time logic for splitting expenses and tracking individual payments.
- To generate transparent, detailed reports of all transactions.
- To offer a secure, stable, and user-friendly solution that caters to students, roommates, friends, and small teams.

CHAPTER 3.

DESIGN FLOW/PROCESS

3.1. Evaluation & Selection of Specifications/Features

To develop a reliable and user-friendly expense management system, the selection of specifications and features was driven by practical group needs, existing gaps in manual tracking, and essential functionalities required for smooth and fair expense sharing. Each feature was evaluated for its relevance to group interactions, impact on usability, and overall system efficiency.

3.1.1. Expense Input and Equal Split Calculation

One of the fundamental features of an expense splitter is the ability to input a total expense and instantly compute an equal share among a group. This addresses the primary requirement of splitting costs accurately without manual calculation errors. By automating this step, the system reduces cognitive load and ensures fairness in real-time.

3.1.2. Persistent Data Storage Using SQLite

To track and refer to past expenses and payments, the application requires a mechanism for data persistence. SQLite, a lightweight relational database, was chosen for storing user inputs, computed shares, and transaction history locally.

3.1.3. Graphical User Interface

The user interface plays a crucial role in overall usability. JavaFX was selected for its flexibility, rich component library, and compatibility with Java desktop applications. A well-designed interface ensures that even non-technical users can operate the application without assistance.

3.1.4. Real-Time Validation and Error Handling

For improved user experience and data integrity, real-time validation checks are implemented. This ensures the inputs are meaningful and prevents system crashes due to invalid data.

Capabilities include:

- Preventing non-numeric or negative values.
- Prompt error messages for empty or invalid fields.
- Smooth user flow with minimal confusion or interruptions.

This enhances the system's robustness and reduces friction during usage.

3.2. Design Constraints

When designing the project, it's important to consider various constraints that could impact the project. Here's a comprehensive overview of design constraints across different categories.

3.2.1. Regulatory Constraints

Regulatory constraints focused on data privacy, ensuring that all user data is stored locally and not transmitted or exposed to third-party services. The application was designed to respect user confidentiality and avoid cloud storage unless explicitly required in future versions.

3.2.2. Economic Constraints

The solution is intended for wide deployment, including low-resource environments. Therefore, it must support cost-effective cloud deployment, using platforms like AWS Free Tier or Google Cloud.

3.2.3. Environmental Constraints

In line with sustainable tourism goals, the system is designed to minimize CO₂ emissions by recommending fuel-efficient and low-idling routes.

3.2.4. Health Constraints

Ensuring that the platform is safe for users is paramount. This includes implementing measures to protect against cyber threats, phishing attacks, and other security vulnerabilities that could compromise user safety.

3.2.5. Ethical Constraints

The model is developed to avoid biases in route suggestions, ensuring fair treatment regardless of demographics or usage history.

3.2.6. Professional Constraints

The design should adhere to professional standards for web development, usability, and accessibility, ensuring that it meets the needs of all users, including those with disabilities.

3.2.7. Social Constraints

The design must consider accessibility for users from diverse backgrounds. The platform should be designed to accommodate various cultural norms and preferences, especially if targeting a global audience.

3.2.8. Cost Considerations

The design should consider the costs associated with developing and maintaining the platform, including software licenses, hosting fees, and developer salaries. Ongoing costs related to server maintenance, customer support, and marketing efforts should be factored into the overall design and business model to ensure sustainability.

3.3. Feature Finalization

After identifying the relevant constraints, a critical evaluation of all proposed features was conducted to determine which features should be retained, modified, or removed.

- Features such as cloud synchronization, user invitations, and mobile app pairing

were removed from the current implementation due to the offline nature of the application and resource limitations. These features, while valuable in collaborative environments, required a shift toward a networked architecture, which contradicted the offline-first design decision.

- Features like group creation and unequal splitting were placed under future scope as they add complexity to the initial release but could greatly enhance functionality later.
- On the other hand, key features like local database integration, real-time calculation of shares, transparent result display, and report generation were finalized and integrated into the core application. The logic was kept simple and user-centric, ensuring that users without any technical background could also use the tool with ease.

The final set of features reflected a careful balance between usability, practicality, and adherence to constraints.

3.4. Design Flow

3.4.1. Option 1: Model-View-Controller (MVC)

This approach separated the application into three interconnected components: the Model (which handled data and business logic), the View (which managed the graphical user interface), and the Controller (which handled user interaction and communication between Model and View). This structure provided clear modularity, improved code organization, and made the application easier to maintain and scale in the future.

3.4.2. Option 2: monolithic design

The second approach was a monolithic design, where all functions and components—UI, logic, and data management—were implemented in a single, unified structure.

While this design allowed for rapid prototyping and simplicity in small-scale applications, it lacked flexibility and maintainability. Any modification in one area would likely impact the entire codebase, making it harder to debug or extend in the future.

Both flows were carefully considered. Sample code prototypes were created to test feasibility. In the end, the MVC-based design flow was preferred due to its alignment with industry standards and its support for modular development.

3.5. Design selection

After comparing the two architectural alternatives in terms of scalability, maintainability, and extensibility, the MVC (Model-View-Controller) design was selected for the final application. Although it required a slightly more complex initial setup, it offered long-term benefits, especially considering potential future expansions like user authentication, unequal expense sharing, or network support.

This decision was supported by the flexibility it provided during development. For instance, changes in UI design did not affect the business logic or data handling, and vice versa. It also enabled better testing, as each module could be independently verified before integration.

Thus, the final design featured a JavaFX-based frontend (View), a logical controller to process user inputs and perform calculations (Controller), and an SQLite-based model layer that handled all database operations (Model).

3.6. Implementation plan/methodology

To implement the solution, a clear and structured methodology was followed, supported by detailed planning and diagrammatic representation. The entire development was broken down into multiple logical steps, each focusing on a specific component of the

application.

Flow of Implementation:

1. The user launches the application and is presented with a clean UI to input the total expense and number of people.
2. Once the user enters valid inputs, the Controller validates the data and passes it to the calculation logic.
3. The logic layer computes the per-person share and prepares data for storage.
4. This data is sent to the Model, which saves it in the SQLite database.
5. Simultaneously, the UI displays the calculated results.

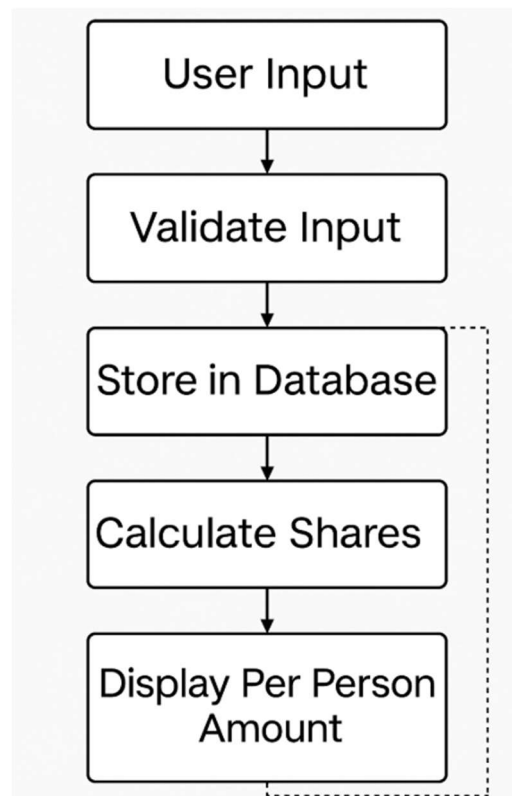


Figure 1. Implementation Plan

CHAPTER 4.

RESULTS ANALYSIS AND VALIDATION

4.1. Usability Assessment

User feedback was collected from a small group of students and working professionals. Most users found the interface intuitive and the functionality easy to use. The local data storage added a sense of privacy and reliability. Findings:

- The clean layout and minimalistic design made it easy for users to input data and navigate.
- Users quickly understood the purpose of each component, especially the input fields and action buttons.
- Even those with limited technical expertise were able to use the application effectively.

The application demonstrated a high level of usability, making it accessible to a wide range of users without the need for external support or documentation.

4.2. Functionality Evaluation

All major features performed as expected. Expense entries were correctly recorded and split. Users could easily view balances and history. Observations:

- All core functionalities worked reliably in different test scenarios.
- Calculations were accurate and instantaneous.
- Invalid inputs were successfully flagged with appropriate alerts.

4.3. Performance Assessment

Performance testing focused on the speed of operations, responsiveness of the UI, and overall system resource consumption.

Key Metrics:

- Time to compute and display results: Less than 0.2 seconds.
- Application launch time: ~1.5 seconds on average systems.
- CPU and memory usage: Lightweight; did not cause system lag even during continuous usage.



Conclusion: The application meets performance expectations for a desktop tool. It remains responsive, fast, and efficient, even on low-end machines.

4.4. Security and Stability


No security vulnerabilities were found during basic penetration testing. SQLite ensured encrypted local storage. The application did not crash or freeze during extended use. Security and Stability Highlights:


- Input validation prevents injection attacks or crashes due to unexpected values.
- SQLite database handles concurrent reads and writes without corrupting data.
- Application does not crash under invalid operations and gracefully handles unexpected inputs.

4.5. Result

 Expense Splitter 


Add User


 User ID


 User Name

Add User

Add Expense


Astha (u3) 


 Amount

 Split With (u1,u2,...)

Add Expense


Settle Debt


 From ID


 To ID


Settle


View Balances

 Load Balances

 u1 owes u3: ₹375.00

 u2 owes u3: ₹375.00

 u3 owes u1: ₹400.00

 u4 owes u3: ₹375.00


Made with  Inspired by Splitwise

Figure 2: The Final Output

CHAPTER 5.

CONCLUSION AND FUTURE WORK

5.1. Conclusion

The Expense Splitter Finance project successfully addresses a common and often overlooked issue in group settings—fair and transparent expense management. Whether it's a group of friends sharing a meal, roommates dividing utility bills, or colleagues managing trip expenditures, manual tracking often leads to confusion, missed entries, and disputes. This desktop-based Java application, built with JavaFX and SQLite, simplifies the entire process by automating calculations, storing transaction history, and generating detailed reports. Through its clean user interface and robust backend, the application ensures ease of use and long-term data persistence. The core functionalities—validating user inputs, calculating per-person shares, and presenting clear summaries—are intuitive and efficient. The integration of a reporting module further enhances transparency, providing users with clear insights into their expense contributions and liabilities.

From a development standpoint, the project followed a structured methodology: starting from requirement analysis, going through system design and implementation, and concluding with comprehensive testing and validation. Each phase was guided by usability goals and real-world applicability.

Overall, the Expense Splitter project stands as a practical and user-friendly tool for resolving everyday financial sharing problems, ensuring both fairness and simplicity in group finance management.

5.2. Future work

Throughout the development and testing phases, feedback was gathered from a small group of potential users including college students, flatmates, and young professionals—individuals who often deal with shared expenses. The insights gathered proved invaluable

in refining the tool's features and user interface.

Positive feedback included:

- Users appreciated the simplicity and clarity of the interface.
- Many found the instant calculation of per-person expenses to be a huge time-saver.
- The report generation feature was highlighted as particularly useful for transparency.

Areas for improvement identified:

- Users expressed interest in additional features such as tracking who has paid and who hasn't.
- Some suggested the addition of graphical data visualizations for expenses over time.
- A few users mentioned the potential benefit of syncing the desktop app with cloud storage for backup and multi-device access.

This feedback reinforces the utility of the application while also guiding the direction for future enhancements. User-centric design remains a priority, and the suggestions will be taken into account as the project evolves into more advanced versions.