1. Please list out changes in the directions of your project if the final project is different from your original proposal (based on your stage 1 proposal submission).

From the overall perspective of our project, due to time constraints, we made a conscious decision at the beginning of the implementation phase to prioritize and categorize the features we planned to develop. We divided these features into three groups: **must-have** features, **nice-to-have** features, and **extra features** that could be implemented if time permitted. By focusing on the core functionalities, we successfully implemented all the must-have and part of nice-to-have features, such as Transaction Management, Spending Tracking, and the ability to Create/Join Groups. However, we had to omit some features, like data visualization, which were not critical to the project's core purpose.

**UI Change:** One of the significant changes we made was in the design and complexity of the UI. During the early stages of the project, a significant portion of our time was spent on database design and implementation, as we wanted to ensure a robust and efficient backend system. This resulted in less time being available for UI development. As a consequence, the final UI is much simpler compared to what we initially proposed. Many of the planned data visualization components, such as interactive charts or graphs, were removed to prioritize functionality over aesthetics.

**Inflation Rate Calculation:** Another notable change was the decision to exclude the calculation of inflation rates. In our original proposal, we considered incorporating inflation adjustment as an advanced feature to enhance the usability of the accounting book. However, as the project progressed, we recognized that our system primarily serves as a simple real-time bookkeeping tool. Given this focus, features like inflation rate calculations, which apply to long-term financial analysis, were deemed unnecessary. Real-time transaction tracking and spending management were our core goals, so we decided to omit this feature to better allocate our time and resources.

2. Discuss what you think your application achieved or failed to achieve regarding its usefulness.

**What we did:** Our application successfully achieved its core objective of providing a simple, functional, and user-friendly accounting tool that facilitates basic financial tracking and group-based expense management. The application effectively supports essential bookkeeping features such as Transaction Management, Spending Tracking, and the ability to Create/Join Groups. These features enable users to record, view, and manage their expenses in a clear and organized manner. The group functionality allows users to collaborate on expense tracking, which is particularly useful for shared expenses among friends, roommates, or teams. This

simplifies the process of managing group-related finances, such as splitting bills or tracking group spending. By streamlining features and focusing on core functionalities, the application remains simple to use without overwhelming users. It fulfills the needs of users who require a straightforward accounting book for real-time expense tracking. Most Important thing is the robust design and implementation of the database ensures that data storage and retrieval are consistent and efficient. This reliability contributes to the application's overall usefulness by preventing data loss or corruption.

Limitations:

(1) **Lack of Data Visualization:** One of the most significant limitations is the absence of data visualization features such as graphs, charts, or summaries that would allow users to quickly understand their spending patterns. Visualizing data would greatly improve the application's ability to provide insights into a user's financial behavior, making it not just a record-keeping tool but also an analytical one.

(2) **No Advanced Financial Features**:While the application works well for simple, real-time expense tracking, it lacks advanced features like budget planning, spending limits, or financial goal setting. These features would have significantly improved its long-term usefulness for users looking to manage their finances more comprehensively.

3. Discuss if you changed the schema or source of the data for your application

We didn't make any major changes to the overall schema or data source, but we did optimize some of the fields. Transaction IDs and Spending IDs were manually assigned, but now they are auto-incremented. This adjustment simplifies data insertion operations, ensures the uniqueness of each record, reduces the complexity of manually managing IDs, and improves the reliability and efficiency of the database.

4. Discuss what you change to your ER diagram and/or your table implementations. What are some differences between the original design and the final design? Why? What do you think is a more suitable design?

We have not modified the original ER diagram, but we have made some minor adjustments to some of the table structures in order to add trigger functionality. For example, to support the function, we ensured the consistency and integrity of certain fields. For example, when we insert or delete a record in Party(Group) will also be reflected on the partyMember table to maintain consistency of the database. These adjustments ensure automation of database operations and data accuracy without affecting the core structure of the original design.

5. Discuss what functionalities you added or removed. Why?
Data visualization: including charts and statistical analysis, due to limited development time, priority was given to ensure core functionality. Inflation Rate Calculation: Considering that our application is a real-time ledger tool, this feature is not part of the core requirements and was therefore removed Beautiful UI design: Because the team's front-end development capabilities are limited, we opted for a simple and functional interface rather than a highly aesthetic design.

6. Explain how you think your advanced database programs complement your application.
Our advanced database design, including procedures, ensures the application's core functionalities, like Transaction Management and Spending Tracking, operate reliably and efficiently. Stored procedures streamline complex operations, reduce redundancy, and enhance data consistency. Features like auto-incremented IDs and triggers further simplify operations and maintain integrity, while the robust schema supports future enhancements like data visualization for long-term adaptability.

7. Each team member should describe one technical challenge that the team encountered.  This should be sufficiently detailed such that another future team could use this as helpful advice if they were to start a similar project or where to maintain your project.

- Integrating the front end with the backend posed challenges due to limited team experience in designing functional user interfaces. Ensuring smooth communication between the front end and backend APIs required thorough testing of request-response cycles. Future teams should prioritize early integration testing and maintain clear documentation for API endpoints to streamline the development process.

- One major challenge was implementing triggers to maintain data consistency across tables. For instance, when inserting or deleting a record in the `Party` table, corresponding changes needed to reflect in the `partyMember` table automatically. Ensuring this without creating infinite loops or redundant updates required a deep understanding of trigger logic and careful debugging. **Advice**: Future teams should design trigger logic thoroughly before implementation, ensuring a clear understanding of which fields require synchronization. Writing comprehensive test cases can help validate functionality step-by-step. Additionally, avoid overusing triggers as they can negatively impact database performance.

- Integrating the backend API with the frontend UI was challenging due to differences in data formats and limited experience in handling API responses effectively. For example, inconsistencies in JSON formatting or errors in handling null values caused delays and required significant debugging.
  **Advice**: Future teams should establish a clear API contract early in the project and ensure consistent data structures between the frontend and backend. Using tools like Postman or Swagger for API testing before frontend integration can save a lot of time and frustration.

- While implementing advanced database queries, ensuring low query costs for large datasets proved challenging. Some queries, especially those involving multiple joins, initially had high execution times due to suboptimal indexing and schema design. Optimizing indexes and restructuring queries took several iterations to achieve acceptable performance.
  **Advice**: Future teams should use tools like `EXPLAIN ANALYZE` to evaluate query performance early and frequently. Design indexes based on expected query patterns, and consider query optimization a continuous process rather than a one-time task during development.

8. Are there other things that changed comparing the final application with the original proposal?
Yes, several aspects of the final application changed compared to the original proposal:

- UI Design: The final UI is simpler than proposed, as more development time was allocated to backend robustness. Features like interactive visual elements were sacrificed for functionality.

- Database Optimizations: While the overall schema remained consistent with the proposal, we added auto-incremented IDs and triggers to improve efficiency and ensure data consistency, which were not explicitly planned initially.

9. Describe future work that you think, other than the interface, that the application can improve on.
Future improvements for the application, beyond the interface, could focus on adding advanced features like data visualization, including interactive charts and graphs, to provide users with better insights into their spending patterns. Enhancements could also include budget planning tools, financial goal tracking, and spending limit notifications to make the application more comprehensive for long-term financial

management. Additionally, optimizing database queries for even faster performance and incorporating features like automated data backups.

10. Describe the final division of labor and how well you managed teamwork.
Our team of four divided tasks based on individual strengths to maximize efficiency. One member focused on database design, including schema optimization, stored procedures, and triggers. Another handled core application logic, such as transaction management and group functionalities. The third team member concentrated on front-end development, creating a simple but functional UI and integrating it with the backend. The fourth member was responsible for testing, debugging, and ensuring overall system reliability. We managed teamwork effectively through regular meetings, clear communication, and collaborative tools to track progress. This structured approach allowed us to address challenges and complete essential features on time, even with adjustments to the original plan.