Project Video Link: https://www.youtube.com/watch?v=TnnN9NnAK_M

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)

The overall vision of our project has remained the same since stage 1, but we struggled to implement features to the detail that we had originally hoped for given the time constraint. In particular, we had originally hoped for our web application to store all trip details (flights, hotels, activities) in one place, but our final project includes a database of less detailed information. We keep track of attributes such as the title of the trip, a description, the location, the dates. As we worked on the project, we realized that keeping track of flights and hotels and individual activities includes more details than the information our dataset provides. Also, we had originally planned to include more features related to the budgeting and expenses part of the project, but given the time constraints, we mainly focused on achieving full functionality with our basic trip information.

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

When we wrote our first stage proposal, we had two goals in mind in terms of the usefulness of our product: 1) Allow users to be able to keep track of their trips 2) Allow users to also budget their expenses for each specific trip. Overall, our application achieved the usefulness for the first goal, but we did not have enough time to do the same for the second goal. Our app currently has all the features for users to create, read, update, and delete all trips, which is extremely useful for users. It allows them to have a travel log over time. For the future, we hope to have the same features with the budgeting attributes of each trip.

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

Yes, the original datasets that we had picked did not align much with the schema that we had created since it did not have much details about the trips, but rather information about different places around the world. Since we needed specific data about each trip specifically, we chose to auto generate our data for the time being. As an extension of this project, we would like to eventually transition to using real-world data from a more suitable source. However, because we were unable to find a dataset that matched our needs during this project, we decided to move forward with auto-generated data for now.

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?

One major change we made from the original design to the final design was updating how the relationship between Users and Trips was handled. Originally, in our database schema, the Trip table had a foreign key UserId referencing the User table. However, in the ER diagram, we had modeled the relationship between Users and Trips as many-to-many so a single user could join multiple trips, and a single trip could have multiple users collaborating. To properly implement this many-to-many relationship, we created a join table called TripUser, which had its own composite primary key (UserId, TripId) and foreign keys to both User and Trip. We removed the UserId column from the Trip table to avoid redundancy and potential inconsistency. Now, all associations between users and trips are handled through the TripUser table. We also removed the foreign key we originally had on UserId in the trip table that referenced User's UserId since it became redundant. We think our final design with the TripUser table is more suitable because it allows a trip to have many users instead of being tied to one, it eliminates any confusion on who "owns" the trip since that becomes a new attribute in TripUser, and we can add more features to TripUser in a clean and flexible manner.

Discuss what functionalities you added or removed. Why?

Our final project includes all the CRUD features as we had originally planned. These features have been tested and are functioning properly. Besides these features, we had originally planned to include the creative component of being able to convert all the expenses to the home currency. We were not able to get to this functionality within the timeline of the project, but still would like to implement this in the future as an extension to the project. In the list of functionalities in stage 1, we also said that we wanted to allow users to be able to search and filter expenses based on categories, dates, or amounts. We implemented this using a simple search bar at the top of our application where you can search by trip name. Another functionality that we removed, but still would like to implement is allowing users to export trip reports for financial tracking. In general, we did not get to focus much on the financial and budgeting aspect of the project and functionalities related to that.

Explain how you think your advanced database programs complement your application.

Our advanced database programs have enhanced how TravelEase functions by making the app smarter and more efficient. Using stored procedures, transactions, and advanced SQL queries, we were able to automate certain tasks and provide useful insights to users without adding extra work on the backend or frontend. For example, the stored procedures help quickly find the top 3 most expensive trips and list frequent travelers. This keeps complex data processing within the

database, so our Flask backend and frontend stay fast and useful. We also used transactions to automatically apply a one-time cost to trips when needed, making sure that data stays accurate and consistent without manual updates. In addition, our SQL queries such as one that assigns an "Admin" label to active users based on their expenses, help personalize the user experience and automate management tasks. Overall, these database features support our application by automating processes, while reducing the workload on other parts of the system.

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

Manya struggled with connecting the frontend to the backend. When first trying to connect the frontend to the backend APIs, I ran into issues where the frontend could not successfully make requests. After troubleshooting, I realized that simply using localhost in the frontend did not work when trying to access the backend server from different devices or environments. To fix this, I had to replace localhost with my machine's local IP address so that other devices (or even my own browser under different settings) could correctly route requests. I also used Postman to test backend APIs directly and realized that the database was only accessible when the IP address of my laptop was added as an authorized source. Without this, API calls would fail silently. In addition, I learned that for better development and testing, it's important to bind the backend server to 0.0.0.0 so it can listen for requests across the network.

Japjeev struggled with the delete and update functionalities in CRUD. When implementing the delete and update functionalities, a major challenge that I ran into was accidentally trying to define multiple separate routes for the same /trips/<trip_id>endpoint. I originally had used the same endpoint for the different types of request (GET, UPDATE, and DELETE). However, since Python flask does not allow for there to be multiple definitions of the same route, the behavior was unpredictable and I was not able to pinpoint the issue for some time. I eventually realized that it is better to use a single route and differentiate between request types using if request.method == 'GET', elif request.method == 'DELETE', and so on. Also, from a UI perspective, I learned that it is useful to have a confirmation prompt before a delete action to prevent accidental deletion of data. As a result, I added this feature.

Ranjana struggled with creating the advanced queries inside the stored procedures. At the beginning of the project, we designed a stored procedure that required user input parameters (such as TripID or UserID) to execute specific queries. However, I realized that during frontend development, requiring dynamic user input made the API calls much more complicated, since every time we wanted to run the stored procedure, the frontend would have needed to get user input, validate it, and send it as a parameter, and also handle user errors such as typos and more.

This added extra overhead and over complicated the process. To solve this, I redesigned the stored procedures without any input parameters and instead created stored procedures to fetch generalized results that were in line with the point of the application. This way, the frontend was much cleaner while remaining useful.

Sneha had issues with indexing in Stage 3. This was a challenge because I couldn't find indexes that improved the efficiency of the queries. Most of the attributes that I was querying were already automatically indexed by the database because they were primary keys, which I did not realize was happening. This is why the efficiency of the queries was not being improved. To address this, I found non-primary keys to index so that they were not already being used by the database to potentially improve the performance of the queries, and I learned that you cannot expect every index to improve the efficiency of the query, which I was trying to achieve.

Are there other things that changed comparing the final application with the original proposal?

Yes, several changes were made between our original proposal and the final application. One major difference is that we decided not to implement the dynamic currency conversion feature that was part of our initial plan. We chose to focus more on core functionalities like trip and expense management. In the final version, we added a dashboard where users can view all their trips and create new ones, which was a refinement from the more general interface we originally described. We also introduced new features that weren't part of the proposal, such as applying a one-time cost automatically to trips and assigning a special "Admin" label to active users based on their number of expenses. We also added two buttons on the dashboard such as "Show Frequent Travelers" and "Show Expensive Trips," to display the top 3 most expensive trips across all users and another to identify frequent travelers who have taken more than one trip. We have added two database triggers. The first trigger ensures that any trip inserted without a description is automatically given a default description. The second trigger archives deleted trips into a separate table with a deletion timestamp, ensuring that records are kept for previously deleted data.

Describe future work that you think, other than the interface, that the application can improve on

Other than enhancing the interface, TravelEase can improve by integrating smart budgeting features through data analysis or machine learning. This would allow the app to provide personalized spending insights and also alert users when they are crossing their budget limits. Another area for future development could be collaborative expense tracking/management, where users traveling in groups can track shared expenses and automate cost-splitting within the app. This would eliminate the need for external tools like Splitwise as well. Lastly, connecting

TravelEase with other services like flight and hotel booking sites or calendar apps could make trip planning easier by automatically adding reservations and schedules. This can help users keep all details in one place for a smoother travel experience.

Describe the final division of labor and how well you managed teamwork

Throughout the project, we maintained strong collaboration and communication, which allowed us to support each other at every stage. In Stages 1 to 3, we worked closely together on the bigger tasks, such as brainstorming ideas for the creative component, designing advanced queries, developing the UML diagram, and writing the DDL commands. For Stage 4, we divided responsibilities based on our strengths and project needs. Ranjana focused on developing the frontend. Meanwhile, Japjeev, Sneha, and Manya handled the backend development and integration, working to connect the Flask backend with the frontend. Within this, Ranjana also took ownership of implementing the stored procedures, while Sneha and Manya focused on the transaction logic and constraints. Japjeev was responsible for developing the triggers.

Our teamwork was very effective and we maintained communication throughout and regularly checked in to address challenges as they came up. Whether it was debugging integration issues or refining database logic, we were always ready to support one another. This not only helped us stay on track but also made the experience enjoyable. Overall, we’re proud of how well we worked together, combining our individual contributions into a cohesive and successful final product.