Final Report

- **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).**
    - **Discuss if you changed the schema or source of the data for your application**
        - We updated the schema to additionally include the latitude and longitude for apartments, streetlights, and crashes so that we are able to plot the apartments on the map in their exact locations, and have the option of plotting streetlights and crashes in the future. The source of the data did not change.
    - **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 made each table in the diagram include more data from the original data sources so that we had more columns to work with in the next stages.
    - **Discuss what functionalities you added or removed. Why?**
        - We did not remove any functionalities defined in our proposal. We added some functionality in the filtering feature. Originally, we proposed that users can filter apartments based on their SafeStay score and community score. In our final projects, in addition to this, we allow users to filter apartments based on how many neighboring street lights they have, and how many pedestrian crashes have occurred near them. This allows users to prioritize safety factors that are important to them when searching for apartments, and grants users better access to the different types of data sources we have.
    - **Are there other things that changed comparing the final application with the original proposal?**
        - Originally, we had some ideas of an algorithm to calculate safestay score for each apartment but nothing set in stone. Through a bunch of experimenting, and a lot of fine tuning, we came up with an algorithm that utilizes 4 metrics: the number of streetlights within a 500 meter radius to that apartment, the average wattage of those streetlights, the number of crashes that happened in the block the apartment belongs to, and whether or not there is a police station in

the block the apartment belongs to. Each of these metrics is given a weight and the score is calculated through adding these weighted scores together. Our goal was to create as much spread as possible as our safestay score is a range from 1-10.

        - In our project proposal, we originally had planned to not only display where apartments are located, but also where streetlights, police stations, and pedestrian crashes happened. We ended up only plotting apartments because plotting streetlights, crashes, and police stations would have made the map super super cluttered. Given our application's purpose of helping people find apartments around the area based on safety, it makes the most sense to plot what is the most important - the apartments themselves.
- **Discuss what you think your application achieved or failed to achieve regarding its usefulness.**
    - Achieved:
        - Addresses a critical need by helping people prioritize safety in their decision-making process when searching for apartments in the Champaign area.
        - Community Aspect was achieved as well as users are able to share and add their own safety ratings based on their own personal experience with living at a particular apartment
        - The map-based interface provides great visualizations and allows users to easily locate where apartments are and quickly find their corresponding community and safestay scores
        - Enables users to filter apartments based on specific factors, enhancing customizability and user experience.
        - Raising awareness of potential factors that might influence safety, such as street lights, crashes, and proximity to police stations.
    - Failed:
        - Because our data has come from public sources and is a little outdated, we would have liked to include features to add new information, namely streetlights, police stations, crash reports directly through the frontend
        - Our data is coming namely from Champaign, we would have liked to include data from Urbana, because that represents a huge portion of potential apartments as well
- **Explain how you think your advanced database programs complement your application.**

- Procedure: We wrote a stored procedure to calculate community scores for all apartments by averaging all user ratings for specific apartments. If an apartment was rated by 0 or 1 persons, we set the community score to -1 and display to the user that the community score for that apartment is not available since there is not enough data to show a meaningful value. The procedure creates a table with the community score for each apartment. The community score is displayed in the application when an apartment marker is clicked on, and used for filtering, so having this table avoids us from having to calculate the community score many times even if it has not changed. The procedure is especially useful for batch updates, so we used it on the ratings data that we initially generated. It can also be used for scheduled maintenance, such as once a day. Although we have triggers to update this community score when a user adds or updates a rating (described below), there could be inconsistencies in the data due to simultaneous updates by multiple users, so running the procedure routinely will correct those issues.
- Trigger: We wrote two triggers to update the community score originally calculated by a procedure (described above). One is triggered after an insert to the Rating table, and the other is triggered after an update to the Rating table. For the same reason as above, we only update the community score for the rated apartment if there are 2 or more ratings for that apartment. The trigger only updates the community score for the apartment that was just rated, rather than all of the apartments (as the procedure does), making it an efficient way to have an updated list of community scores.
- Transaction: We use a transaction for getting filtered apartment data, which comes from multiple queries and intersecting their results. The isolation level, repeatable read, ensures that the data being queried from remains stable throughout the transaction, even if another transaction updates the underlying data. This is especially important because the filters may relate to each other, such as if a user is looking for apartments with a SafeStay score in a certain range–if the data changes between such queries, it can lead to inconsistent results. Additionally, the transaction involves multiple steps that depend on each other, so rolling back ensures no partially executed queries remain in the session.
- Constraints: We used primary and foreign keys to clearly show the relationships between each table, making the database an interconnected system that supports the application's functionality. Primary keys uniquely identify records in each table, like username in the User table or

block_number in the Location table, while foreign keys link related data across tables. For example, the username in the User table connects to the Rating table, associating user ratings with specific apartments. Similarly, the address in the Apartment table ties to the Rating table to link ratings directly to apartments. The block_number in the Location table connects geographic areas to apartments, streetlights, and pedestrian crashes, helping to analyze safety metrics comprehensively. This design highlights how the tables relate to each other, ensuring accuracy, consistency, and the ability to easily trace connections across the database.

- **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.**
    - Neha: The data we found on the Champaign County Regional Data Portal regarding apartments, streetlights, and pedestrian crashes was in GeoJSON format. There was an option to download the data as .csv files, however the .csv files were missing critical information regarding the locations of these entities. So, in order to be able to import the data into our database, we had to convert the GeoJSON files into .csv files. Luckily, there is a library called GeoPandas that helped us with this, but we ran into some technical challenges. For the location of the entities, the geometry object associated with them was a polygon, so we had to extract the center point and convert it into latitude and longitude so that it is usable in our application. However, when doing this conversion, we had the wrong coordinate reference system, which we noticed while trying to plot the apartments on the map. As a result, we had to delete all of our existing data and reinsert the data with the correct latitude and longitude values.
    - Steven: Setting up Google Cloud, creating a VM and a MySQL instance. This was pretty challenging because there are a lot of intricacies and options that one can choose when creating and going through these steps. Additionally, small but critical details, such as columns being out of order, could easily be overlooked and lead to issues. It was imperative that we all worked together on this step because only one person could access the SQL instance at once. Another difficulty was that our original safestay algorithm ended up only producing 3 unique safestay scores. We had to employ different strategies to come up with a wider range of unique safestay scores.

    - Vy: In our project, we wanted to make sure that the application had a cohesive design. At first, we styled components manually using CSS, but this approach became harder to manage when we started adding brand new components and pages. There were some inconsistencies with designs and it was hard to maintain an entire theme throughout the project. We decided to use a component library such as Chakra UI because it integrated really easily with React, has a consistent design system, and was easily customizable to fit our design needs. Using the built-in style props, we could adjust the padding, margins, and colors. This helped us be more efficient because we didn't need to build buttons or modals from scratch. It also helped us maintain consistency throughout the design when we made theme.ts file, which defined our design tokens such as colors and fonts. We also applied it globally by wrapping it in our chakra provider and passing in the custom theme. For example, if we needed to update a color or adjust a font size, we could make the change in theme.ts, and it would automatically propagate throughout the site.
    - Mahima: While working on a project to plot markers on a Mapbox map, we encountered a perplexing issue where the markers simply wouldn't appear. After some debugging, we discovered that the problem lay in the longitude and latitude values, which were far outside their valid ranges—latitude exceeding 90. Initially, we thought the data might have been incorrectly scaled, so we attempted to normalize the coordinates by dividing them by 10,000. This adjustment brought the values within acceptable ranges, but the markers still plotted in completely wrong locations. It became clear that the original dataset's coordinates were fundamentally incorrect due to errors in the data source or faulty calculations. To resolve the issue, we had to reevaluate the data source and recalculate the coordinates entirely. Once we corrected the dataset, we were finally able to plot the markers accurately on the map. This challenge highlighted the importance of validating and cleaning data before using it in any visualization or mapping tool.
- **Describe future work that you think, other than the interface, that the application can improve on**
    - Security features especially to the signin/signup pages like hashing passwords and preventing SQL injection attacks
    - Nice to have: Adding profile pictures to user accounts, would need to store these pictures in something like a s3 bucket
    - Have users upload their past leases to verify that they lived at the apartment they are rating

    - Features and Options to display police stations, streetlights, and places where crashes happened
    - Expand to not only the Champaign area, but Urbana and other college towns potentially
    - Search Functionality that can filter and find a specific apartment based on a certain address
- **Describe the final division of labor and how well you managed teamwork.**
    - To build this application, we identified five major components of the project: authentication, developing an algorithm to generate the Safestay Scores, integrating the map into the website, implementing user functionality (such as viewing Safestay and Community scores and adding/updating user ratings), and creating advanced filtering queries. We started by collaboratively setting up the repository, database, and Google Cloud Platform (GCP) infrastructure. This initial step ensured that everyone had an understanding of the project's structure and setup. Once the groundwork was in place, we worked together to implement authentication, focusing on the sign-in/sign-up workflow. After completing authentication, we divided the remaining core functionalities among the team to work in parallel. Neha focused on implementing advanced filtering queries, Vy handled user functionalities like adding and updating user ratings, Mahima worked on map integration and querying addresses using markers, and Steven developed the algorithm to calculate the Safestay Scores. Once we came back from break, we worked all together on combining all our respective parts into our website and making finishing touches and QOL changes to the frontend and backend.