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).

In our final DwellX project, several changes occurred compared to our original proposal. While we initially planned to incorporate metrics such as political affiliation and employment rates, we ultimately narrowed our focus to housing prices, school ratings, crime rates, hospital proximity, and recreation access. Additionally, although our proposal outlined an interactive dynamic map using MapBox and data visualizations with Chart.js or Tableau, these features were not implemented due to time constraints; instead, users received neighborhood recommendations displayed through static cards. We also intended to develop a community forum for user discussions, but this component was omitted to prioritize core functionality. Our preference ranking feature, originally envisioned with adjustable sliders, was simplified into dropdown selections. On the technical side, we successfully implemented user account creation, login, and preference saving functionalities, with strong backend support through stored procedures, transactions, and triggers. Overall, while the scope and some features were adjusted, DwellX still fulfilled its core mission of providing a personalized, data-driven platform to help users evaluate Chicago neighborhoods.

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

Our DwellX application successfully achieved its primary goal of helping users find suitable neighborhoods in Chicago based on personalized preferences. By allowing users to create accounts, save preferences, and view recommended neighborhoods tied to important factors like housing affordability, school quality, safety, hospital proximity, and recreation, we provided a functional and user-friendly tool to streamline the decision-making process for newcomers. The backend was robust, supporting account management, preference storage, and CRUD operations with proper database integrity through transactions, triggers, and stored procedures. However, the application did fall short of some aspects of our original vision for usefulness. We were unable to implement dynamic visualizations such as interactive maps or side-by-side chart comparisons, which would have enhanced users' ability to visually explore and compare districts. We also did not integrate a community forum, which would have added depth and

qualitative insights beyond the available datasets. Overall, despite some missed opportunities for richer user interaction and visualization, DwellX remains a useful tool for newcomers seeking an efficient, data-driven way to evaluate Chicago neighborhoods.

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

Over the course of implementing DwellX, we largely adhered to the normalized schema outlined in our original proposal, with only a few targeted extensions introduced to support additional functionality. The data model continued to center around the seven core entities: Housing, Schools, Hospitals, Crimes, Parks, Users, and Preferences, and four many-to-many join tables (School_District, Hospitals_Houses, Parks_Houses, and Crime_Level), all populated from the same Chicago open-data CSV sources we initially specified. No new data sources were introduced, and none of the original entities, relationships, or foreign key constraints were sacrificed. However, we did make practical adjustments: due to the large size of the crimes dataset, we imported approximately the first 15,000 rows rather than the full set. To support a user "favorites" feature, we added more attributes to the Favorites table, and wrote database triggers to maintain this count accurately across inserts and deletes, ensuring database-level consistency without requiring expensive aggregate queries. Lastly, to meet project fulfillment criteria requiring a sufficient volume of housing data, we auto-generated additional housing entries.

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?

Our final DwellX implementation introduced a few important changes to the original ER diagram and table designs. While the core structure comprising entities like Housing, Schools, Hospitals, Parks, Crimes, Users, and Preferences remained intact, we made specific updates around user favorites. Originally, we planned a simple join table linking users and housing IDs. In the final design, we restructured this into a dedicated `favorites` table with a `favorite_id` primary key, `user_id`, `neighborhood_name`, and `zipcode` fields, along with a `created_at` timestamp. We also added a composite unique constraint on `(user_id, neighborhood_name, zipcode)` to prevent duplicate favorites at

the database level. This design provided greater flexibility, allowing us to track when a user favorited a neighborhood.

Discuss what functionalities you added or removed. Why?

During the development of DwellX, we both added and removed certain functionalities compared to our original plan. One major addition was the favorites system, which allows users to bookmark neighborhoods they are interested in. This feature was not originally emphasized in our proposal but was added to enhance user engagement and personalization. To support it, we created a dedicated `favorites` table with timestamps and uniqueness constraints, and designed supporting backend endpoints. On the other hand, several functionalities were removed to prioritize core features and ensure a stable product. Notably, we removed the interactive map (originally intended to be built with MapBox), dynamic data visualizations with Chart.js or Tableau, and the community forum for user discussions. We also dropped real-time updates from external data sources, instead opting for a static dataset approach to simplify backend complexity. These removals were necessary due to time constraints, the technical difficulty of integrating third-party APIs, and our decision to focus on ensuring that core CRUD operations, personalized recommendations, and stable user account management were robust and complete. Overall, the functionality trade-offs allowed us to deliver a clean, working product with meaningful core features, even if some originally planned enhancements were postponed

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

Our advanced database programs significantly complemented and strengthened the DwellX application by improving both functionality and system reliability. Stored procedures were used to handle complex queries, such as matching users to neighborhoods based on their preferences, which allowed us to centralize important business logic inside the database and reduce redundancy in the application code. Triggers played a key role in automatically setting default preferences when the Generate page was loaded, ensuring baseline values were established before users entered their specific inputs. This automation aligned perfectly with the intended purpose of a trigger functionality. A Transaction was used to create a ‘Popular’ tab that displayed houses that were preferred by most

users. These are particularly useful when dealing with databases that are being modified in real time. Since the preferences table is not static and is constantly being inserted into or updated, we chose a repeatable read isolation level to prevent dirty reads and only view committed data. This guaranteed that the database state remained consistent. We allowed for phantom reads, as a few new insertions into the table would not cause a huge difference in the most recommended houses. Together, these advanced programs not only optimized performance by reducing the number of database queries needed but also enforced integrity constraints that would have been difficult to maintain solely in the application layer.

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.

Manavi – Authentication + Search
Manavi worked on building the Authentication system and implementing the Search functionality. She focused on ensuring that users could securely log in, stay authenticated across sessions, and search for neighborhoods based on user-specified criteria. A technical challenge she encountered was ensuring that user information was successfully updated and stored in the MySQL `Users` table after authentication events such as registration and login. Sometimes the updates would not persist properly due to misconfigured queries or missing backend validation. She always double-checks backend logic to verify that database mutations (INSERT, UPDATE) are committed properly, test each authentication endpoint thoroughly with both valid and invalid inputs, and to carefully configure CORS and cookie policies between frontend and backend cause that was another issue.

Khushi – Frontend + CRUD

Khushi led the Frontend Development, focusing on building a clean user experience where users could search neighborhoods, view matches, and save favorites. She was responsible for designing the logical flow of the application, making decisions about which React components should display specific information and how data should be passed between them cleanly. A key technical challenge was ensuring that frontend components displayed the correct state based on complex backend responses, especially when dealing with

nested data returned from stored procedures. She thinks spending time early on mapping out the data flow clearly between components, to normalize backend responses before setting frontend states is safer.

Annapoorna – Backend + CRUD

Annapoorna focused on implementing the CRUD operations for the backend, creating endpoints for managing user preferences and favorites. A major technical challenge she faced was getting asynchronous database functions to work seamlessly with Express.js, ensuring that data updates actually reflected correctly inside MySQL. Integrating them properly into the frontend required reshaping responses, adjusting field names, and handling errors carefully. She worked extensively to make sure that Create, Read, Update, and Delete operations were comprehensive, robust, and fully working with frontend functionality. She would advise to test endpoints independently before frontend integration, handle asynchronous behavior carefully with try-catch blocks, and standardize API response formats early to avoid frontend confusion.

Advita – Stored Procedures, Transactions & Triggers

Advita focused on implementing the Transaction, Trigger, and Stored Procedures components of the project. For the transaction, she spent time considering the different isolation levels to balance consistency of the database with performance. By working with the data and checking what errors cause the most difference in the output, she found that dirty reads would be the worst kind and that phantom reads would not cause much of a difference. While working with the Trigger, an additional check in the code to see whether the preferences table had a row right after a user registered on the website was causing an issue with the user login. Since it interfered with the logic of adding a new row to the users table, she used sources like StackOverflow to figure out the error. This issue was resolved by removing the redundant check and ensuring that the trigger ran after a user was added to the table. Stored procedure, although it did not have any major issues, took quite some time to get right. It had to be modified repeatedly throughout the development cycle, whether to change what outputs were needed to display in the frontend, or to fix rounding issues with floating-point numbers. She recommends future teams to be deliberate about picking appropriate isolation levels for transactions, to carefully test triggers in isolation with both expected and unexpected inputs, and to modularize stored procedures for easier testing and maintenance.

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

We think we have covered all of what changed from our original proposal to our final application in the answer to the first question.

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

Looking ahead, we could plan to embed an interactive Google Maps layer so users can visualize housing locations, school zones, parks, and hospitals in real time, clicking each marker for detailed metrics. Beyond Chicago, we'll expand our data pipeline to open-data feeds across Illinois and then scale to every U.S. state so the same analytics is nationally available. Rather than disjoint endpoints, we'll unify all our advanced queries behind a single "Insights" dashboard with multi-select filters (ZIP, crime threshold, school rating, amenity proximity) so users can combine criteria on the fly.

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

As a team we worked until Stage 3 together as we did 2 queries each and based on the complexity and results we chose the best 5. For loading the datasets we started out together and loaded various datasets on GCP. For the final stage since it was relatively a bigger stage than the rest we divided work where Manavi worked on Authentication and Search, Advita worked on the Transaction, Trigger, and Stored Procedures, Khushi worked on the bulk of frontend and Annapoorna the CRUD operations on backend. Everyone worked on loading datasets and creating the DDL, writing queries, cleaning up endpoints and general debugging. Overall, as a team we feel we worked well together and that we managed the work well.