Team049 PT1 Stage4 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).

The overall direction and core objectives outlined in our proposal have remained consistent. The primary features and creative components we planned have been implemented successfully and align well with our initial vision. We made some minor adjustments based on practicality and feasibility: adding the personalized recommendation feed when logged in using the "AdvancedPropertyInsights" stored procedure, and reducing items such as not developing a data crawler, removing the AI chatbot integration, and omitting community forums.

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

Our application successfully achieved its usefulness by transforming the often stressful process of finding the right housing into a smooth easy process, particularly for students and young professionals. All features on the website function well and handle errors appropriately and correctly. The website achieved smooth user interaction as expected. With a great amount of rental listings and user-friendly tools like map-based search, real-time updates, and personalized recommendations, our platform ensures that our service adapts to the growing needs and tastes of our users, keeping the platform both relevant and deeply useful. However, given the ambitious scope of our initial proposal, some features such as the community forums and AI-driven chatbot were omitted because we decided to prioritize core database functionalities.

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

To improve the granularity and usability of our data, we decided to modify our schema by treating floor plans as weak entities. We also modified the "Favorites" schema: while the original design used only FavoriteID as the primary key, the final design used both FavoriteID and PropertyID as primary key. Additionally, we also added the foreign key constraints (PropertyID) to the Favorites, Listing table so as to refer to the Property table.These help ensure data integrity and enable cascading updates and deletes within the database. It provides a more solid framework for future enhancements and helps maintain data integrity and sets clear hierarchical relationships within the database.

For the source of data, originally, we planned to write a custom crawler to gather data from real estate websites along with the existing dataset. Due to the varied formats and the complexity of handling web crawling legally and efficiently, we opted to use an existing dataset from Kaggle. This decision reduced our development time and allowed us to focus on other critical database features of our application.

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 ER diagram has stayed the same as the original design, and our table structure remains mostly as planned, with only minor modifications. For instance, in the Favorites table, while the original design used only FavoriteID as the primary key, the final design extends the primary key to include both FavoriteID and PropertyID. This modification helps keep data consistent and improves query performance. Additionally, the final design specifies foreign key constraints, adding them to the Favorites table for the Listing, Property, and User tables. These help in maintaining data integrity and enable cascading updates and deletes within the database. We also introduced a PriceChanged field in the FloorPlan table to monitor price fluctuations, supporting our feature for email alerts.

The current design is more suitable because it has been implemented based on these database designs, and we have already completed the development of the entire website using this structure.

Discuss what functionalities you added or removed. Why?

We added a personalized recommendation feed feature that greatly improves user engagement. This feature was achieved using the "AdvancedPropertyInsights" stored procedure, which provides customized property recommendations each time a user logs in. The system automatically displays tailored property suggestions on the user's dashboard based on data from user favorites and ratings, as well as geographical borders. This feature aims to give a more personalized customer experience, which is critical for standing out in the competitive real estate platform marketplace.

On the other hand, some planned functionalities were removed after careful consideration. We dropped the idea of building a data crawler to collect real estate listings because it was too complex and could cause legal issues. Instead, we used existing data from Kaggle, which let us focus on improving the main parts of our app.

We also decided against integrating an AI-driven chatbot, which was initially intended to help users by providing interactive responses and property suggestions. After some tests, we realized it would take too much effort to make sure it worked well and didn't give wrong information, which could frustrate users instead of helping them. Therefore, we prioritized direct user interactions through our recommendation systems over automated chat functionalities.

Lastly, we removed the plan to have community forums. While it could have been a nice way for users to connect, it would have taken a lot of effort to set up and manage, which we thought could be better spent on features that help users find homes directly.

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

Our "AdvancedPropertyInsights" stored procedure directly enhances our app by providing personalized property recommendations each time when a user logs in. It dynamically generates suggestions based on the user's favorites, ratings, and location by processing data through advanced SQL queries. This ensures that recommendations are always fresh, improving user engagement and therefore makes our application more responsive.

Our “AddToFavorites” transaction allows users to add specific properties to their favorites. It verifies whether the property has a valid listing record and whether the listing date is before the current date. If both conditions are met, it creates a new favorite entry and updates the price at the time. The transaction is created to ensure that a consistent snapshot of the data can be read repeatedly throughout the transaction. It prevents scenarios where a transaction might read the same data twice and get different results each time due to changes made by other concurrent transactions. This improves user experience and the usefulness of the data.

Our "FormatAndValidateContactNumber" triggers enhance our application by ensuring data consistency and reliability at the database level. We implement these triggers before inserting and updating operations on the Property table, and we validate and format contact numbers to meet specific criteria. Each trigger checks that the contact number is exactly 10 digits long and that the first digit falls within the acceptable range of 2 to 9, which helps avoid errors due to incorrect data entry. If these conditions meet our standards, we format the contact number into a more readable form (xxx-xxx-xxxx). This approach prevents invalid data from entering our system and enhances the user experience by providing consistently formatted and reliable contact information.

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.

Vincent Lyu:
One challenge I've faced is that approximately 60% of our property dataset only contains longitude and latitude information, and lacking street addresses. Street address is considered vital

to our application. To address this, I implemented Google's geocoding API, which translates latitude and longitude coordinates into street addresses. I integrated this functionality whenever users tried to get the detailed property information instead of querying it for all properties, thus saving costs while providing street information for our users.

Yu Tian:
I faced a challenge in managing the CRUD operations for the Property, which included a List of FloorPlan. This task was complicated because I needed to support modifications for any number of floor plans and handle relationships with other tables like Listing. To ensure data integrity and provide a rollback in case of errors, I ensured all database operations within these methods were conducted within a single transaction. I also crafted logic to manage related entities such as Listings and Ratings and the primary property records. This process required me to develop various test scenarios to fully explore all possible interactions between the Property entity and its associated FloorPlans and ensure proper functioning with related entities. Moreover, I implemented specific error handling for certain database access problems, enhancing our application's robustness and user experience.

Zilin Zhou:
One technical challenge was that in the implementation of the transaction, handling variable case sensitivity in backend API calls. SQL is case-insensitive for identifiers. However, our backend programming required case-sensitive handling of variables. This discrepancy led to inconsistencies when data passed from the front end did not exactly match the expected format in the backend. To resolve this, we standardized all variable names across the entire stack using a consistent naming convention.

Kathy Lo:
Integrating the stored procedure initially was a challenge, mainly because we had little idea of how to design effective queries that could interact with the frontend to dynamically generate personalized property recommendations. Initially, our approach used fixed thresholds to categorize properties based on how often they were favorited, which later we found not effective and inflexible when our user base and data volume grow. To overcome this limitation and let our website have scalability, we shifted to using percentiles to categorize property popularity. This adjustment allowed us to dynamically refine our recommendations based on real user interactions. Properties favored by more than 75% of users are now classified as 'Highly Recommended,' while those favored by more than 50% but less than 75% are labeled as 'Popular.' This shift from hard thresholds to a percentile-based system improved the adaptability and effectiveness of our recommendation engine.

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

We initially planned to write a crawler for data collection and integrate it with the existing database on Kaggle. However, each real estate website encrypts or renders its data differently before sending it to the client, which makes it difficult for us to develop scripts for each housing provider, thus we use the Kaggle dataset instead.

In our original proposal, we designed an interactive AI chatbot to provide housing information by combining data from our database. However, after experimenting with the API, we found it challenging to integrate the database with the chatbot due to the large amount of data each query requires, consuming an excessive number of tokens. Additionally, the AI may sometimes generate properties that do not exist. Which is not ideal for our application.

After some discussions, we decided not to include the user profile function, as its functionality can be achieved through the various filtering functions.

In the project proposal, we mentioned that users could 'engage in community forums, discussing topics related to housing.' However, implementing this function alone could be a separate project, which is not feasible and would deviate from our primary focus.

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

For future work, we can improve our housing web app by integrating MongoDB due to its flexible schema, scalability, and other NoSQL benefits. It can handle diverse and large datasets efficiently, making it ideal for real estate applications where data types and structures can vary widely. Then we can easily incorporate new features like user comments and dynamic ratings without extensive database restructuring. Furthermore, NoSQL has distribution capabilities, meaning we can manage higher traffic volumes by spreading data across multiple servers, enhancing our website's performance globally.

On the other hand, we can make better use of the session management mechanism. Currently, the user authentication states are stored locally using local storage. On login, user data is stored locally and used to fetch personalized insights. On logout, this data is cleared from local storage and the page is refreshed to ensure the user state is reset. We can use Redis to handle user sessions better. Since redis is a fast storage system that can handle many users at once, we can make the site smooth and quick, especially when users return.

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

We divided our work between frontend and backend tasks, split it into different phases. For each phase, we set specific requirements and set a clear deadline. Each team member contributed a significant amount of work and engaged in extensive research and discussions weekly to ensure we met these goals. For each feature, we tested a lot and provided feedback and comments to each other's pull requests, continuously iterating to make sure each feature does not break when the user interacts. On the completion of these deadlines, we meet on Zoom to review our progress, discuss the outcomes, and reflect on our work.