Final Project Documentation

Engineering and Design Decisions

Our project was meant to be an easier way for users to use Zillow. The user could input what they would like their house to be, and our code would export every listing that met their standards. This is meant to streamline the users' experience and save time and effort when searching for houses. We first decided that we would need a Zillow API and a map to place the houses on. We decided that this project would not only help us with our grades but could also help the real world as well.

Preliminary Findings

We found that Zillow API was a private service, so we had to research different APIs. This led us to find scrapers. These scrapers would function as a middleman to find the users inputted information. We found that this service worked very well and produced many different outputs. This led us to want to try and do more with it. We pushed to allow the user to select greater than or equal to on the number of bedrooms and bathrooms as well as allowing the user to have error in their square footage input. This allows the user to not only find exact matches but also ones that may be better for them as well.

Challenges

We found out that we are unable to do a heatmap within Vaadin as it was depreciated more than 6 years ago. So, we had to find a different solution to this. Our goal was to map the cost to square foot ratio within a heatmap. If it was greater than the national average per square foot, it would be red. If it were less than, it would be green. This would have been a great visual for the user to understand. We decided that we were going to just pinpoint the location on the map instead. This would be a lot simpler and would work. While it may not be as visually appealing, it would be functional. We also found out that our API can be rate limited if it has too many inputs as once. This led to a lot of issues. We troubleshooted it for a while and were confused on why we were getting outputs. It turns out that the troubleshooting took long enough that the cooldown would have passed. To counteract this, we implemented a time buffer to allow the cooldown to pass. We never had this issue before. We found out that after a certain number of inputs; the API would rate limit itself. So, while it worked previously, this would change once the user used the

API too many times. We also found out that the map won't take address inputs. This meant that we had to implement a Geocoding API as well. We wanted to make sure that the map would work properly, and we felt that this would be the best choice to get it to work.

Code Review

Our code currently has four java classes. API.java, MainView.java, State.java, MapView.java. API.java includes both our Zillow API and our Geocoding API. We first started out having separate classes for this but decided to merge them as they are too reliant on one another to work. This merger would streamline the process and mitigate having to call each class multiple times within separate classes. State.java handles the getters and setters for our City and State input, as well as gets the coordinates to recenter our map. We have a CSV dataset called uscities.CSV. This includes 31,000 data inputs. Each data input is a different city within the United States. This is used to populate our dropdowns on textboxes. This could have been done with another API, but we decided to go on the easier route and just include a CSV for it all, which was free. MainView.java handles all the user inputs, filtering, grids, and the average cost. There are a lot of different filters within this java class. With a lot of filters, the users will never actually see unless they see the code. The filters are meant to make sure the APIs don't run repeated inputs, and make sure the received data has all the needed outputs the user wants. We will soon include our map on this page as well. We have separated them so we can work independently on it all without redoing someone else's work. As said previously, MapView.java includes the code for our map. The map takes input from the first page, stores it within cache, and when the user reloads the webpage, the map will be reloaded to the users' selected city.

Added since Mid Progress Review

Since our mid progress review, we have gotten a lot more accomplished. Our heatmap now works properly with some help from Node.js and scripts. We also now have our housing price estimator tool. This tool is trained on a Random Forest model to predict what the house will cost based on the city, State, number of beds, number of baths, and square footage. This then outputs a price based on 21000 data entries from a 2023 Zillow data scrape. This satisfies our second technical component by implementing the actual housing price estimator tool we had previously touched within our housing outline. Our map view has gone from nothing to amazing within this time. There is now a heat overlay on the open street map. This heatmap spans the whole united states. It takes the average cost

per square foot from the Zillow scrape data previously talked about and maps it out onto the map using geocoding. The map required everything to be in coordinates, so we had to implement a geocoding API to get this to work as well.

Where do we stand?

We feel like our project has taken a huge leap forward with the new additions. This project went from just a project to an actual tool that could be used by people across the country. We are genuinely proud of the progress we have made and feel as if it has done a great job showcasing our skills and our willingness to learn more to grow within the coding world.