Technical Report Phase 3

- User Stories

  We were able to satisfy all the user stories that were provided by our customer for searching, sorting, and filtering. User story #10 described the user wanting to filter nursing homes by ratings to find the best ones in which we were able to successfully implement that feature. For user story #9, it describes wanting to be able to locate entertainment options that are free, in which this was implemented to where it filters entertainment options by free or paid. For user story #7, it describes wanting to be able to view nursing homes instances side-by-side, so that the user can decide which option is the best for them. This was successfully implemented in which we have many instances of nursing homes around Texas that can be filtered based on specific attributes. For user story #5, it describes that the user wants to filter nursing homes by ratings so that they can find the best option located near them. This was successfully implemented in which we have a ratings attribute that is filterable. The last user story that was implemented was described as wanting to discover entertainment options nearby in which we were able to add filtering options in which users can locate options based on the city.

- Tools

  The different tools that were utilized were similar to the previous phase in which flask was utilized again to update our endpoints for the api in order to incorporate searching, sorting, and filtering. On the frontend, react was used again to be able to display the different ways to filter the model page based on what the user is looking for, as well as a search bar.

- Backend
  - We were able to successfully update our endpoints in our api in order to implement searching and sorting. This was completed through adding another parameter for it where the specific search that the user types is added to the route. An example of a user typing in the city of Dallas for health centers would output this link: https://api.senioruplift.me/api/healthcenters/?search=Dallas.This link would output all of the instances that contain Dallas as the city or if it is present in the name. This can work for any of the three models that we have implemented.
  - For the sorting aspect, an example link for locating health centers with a sort of city in ascending order would look like this: https://api.senioruplift.me/api/healthcenters/?sort=name&descend=yes.
  - A search bar was also implemented individually in each model so that users can search for their respective instance that they are currently looking for. Users can also combine their search in which they can find something in the search bar and be able to filter specific attributes based on ascending or descending order.
  - These different routes are specified in our application.py file.

- Frontend
    - On our frontend aspect, we implemented searching, sorting, and filtering functionalities across the three model pages. This implementation was completed through the use of React and API integration. Searching is performed by updating the SearchQuery state with user input and applying it to the filters object when the Search button is clicked. This filter sends the search term as a query parameter to the API, which filters the results based on the specified attributes in the model page. Sorting is completed through the use of dropdowns, where users can choose the attribute that they would like to sort by. These specific selections update the filters state, which triggers an API call to retrieve the sorted data.
    - For our Global Search, it allows users to search across all of the data models using a unified search bar. Once a specific query is entered in the search bar, the search is processed, and three categorized tabs appear in the SubNavBar component, allowing users to navigate to results specific to for each model. Each tab directs users to a dedicated page where results are displayed using the same structure and design as the corresponding model's page. Filtering, sorting, and searching remain the same for each individual tab. The URL dynamically includes the search query as a parameter, enabling a direct access to filtered results when shared. Once the search query is outputted, the term is highlighted that the user was looking for.
- Challenges
    - When developing these different implementations, my team and I had faced many different challenges. One of our biggest obstacles was creating the global search system that would be able to successfully return the correct results that the user wanted. We also had to make sure the filtering and sorting features were dynamically working in order to ensure a seamless process. Designing the global search to seamlessly integrate with individual model pages was also tricky to implement. We also were not sure if the highlighting search terms were going to maintain its functionality when results were retrieved. Implementing pagination and ensuring everything stayed in sync with the URL was technically complex and required careful implementation. Overcoming these different challenges allowed my team and I to create a seamless working website.

- Testing
    - The tests for sorting, searching, and filtering are focused on ensuring the API's endpoints are correctly functioning and returning the correct data based on the parameters. The filtering tests validate that specific criteria, such as filtering health centers by city, nursing homes by rating, and entertainments by category, return accurate and properly formatted results in JSON. Sorting tests confirm that data can be sorted by attributes like ID, name, or city in either ascending or descending order. Searching functionality is validated by simulating queries that match records partially or completely, ensuring the API correctly handles keyword searches across multiple fields.