Phase 2 Technical Report

User Stories:

- Some of the issues that the customer had provided for we are unable to fulfill as these stories will be completed for the next phase, Otherwise, the stories that were defined for the scope of this phase were successfully completed.

Tools

- Different tools that we used include:
    - We created the python backend using SQLAlchemy and Flask.
    - We utilized MySQL for our database and it was hosted on AWS RDS
    - We used JavaScript and React for the Web framework for the front end
    - We created tests for our restful api using postman
    - Created tests for our javascript framework using jest
    - Created tests of our python back end code using unit test framework
    - Created acceptance tests using selenium

Backend Hosting

- We created an EC2 instance for our API in the back-end and for the AWS RDS to have our database hosted

API

- API documentation for our Restful API can be found on Postman
- We utilized Flask and Flask CORS in order for the api to run on the AWS server

Database

- We hosted our database on AWS RDS called senioruplift_db.
- The database consists of 3 tables: entertainment_model, nursing_home_model, health_care_model
- The connection between the tables and the database can be found on the UML diagram that was created
- We had a models.py file that defines all of the models that are present in the database. Each data source, inputs data from their requests or web scraping and places it into the database.
- googlemaps.py uses google maps API to find nursing homes across Texas with the cities that were specified and it grabs a list of information, along with a picture
- eventbrite.py is a web scraper that utilizes selenium and beautiful soup.
- HealthCenterScraper.py is a web scraper that uses selenium and Beautiful soup.

Docker

- A docker file is used in the backend where we can run the linux version for the AWS hosting. This also contains our application that is on the cloud.

Security

- We have a .env file that stores information such as API keys and a CSE ID to hide sensitive information publicly

Flask application Framework

- The framework that consists of running our application includes a models.py and application.py. These file contains information with defining the different models as well as the specific routes for our api

Testing

- We have backend tests that can be found in api_tests in the tests directory
- We also created Postman tests that test specific endpoints for our api
- We also created Front-end testing using the jest framework that tests our javascript
- We also have front-end GUI testing that utilizes Selenium

Pagination

- Pagination was created using the pagination controls class where you can flip through many pages of instances for each of the 3 models.

Dynamically creating Instances

- We were able to create dynamic instances through integrating the back-end onto the front-end. This was done through having 3 separate instance classes for each of the models. In each class they would call their specific api route that would allow them to pull the correct information on the front-end. This call was completed through an API call.
- Each instance page has several different attributes describing the instance for users to learn more about. You can also click on these instances that take you to a page that gives more information about each instance.

Challenges

- Being able to host the back-end and load the information into the database was a big challenge as we did not have much experience in doing this.
- Figuring out the web scraping for the different parts was also a huge challenge as we ran into many bugs during this process.