Sprint 1

- Design overall structure (determine project language, necessary data structures, potential algorithms, edge cases)
- Design Simulation to initialize a set amount of people distributed to list of bars
- Implement objects to represent bars
- Create initial recommendation algorithm

Requirements Engineering:

Functional requirements :

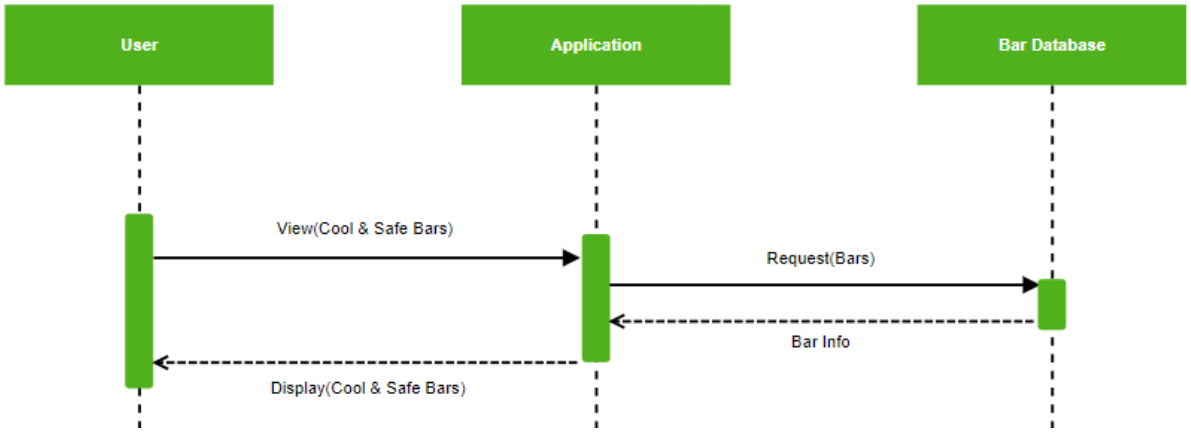
- Recommends safe and fun bars
  - Current Algorithm to achieve this:
    - Between 60% and 80% occupancy→ Recommend
      - <60% → Not cool
      - >80% → Not fun
  - Updates with as occupancy changes
  - Inputs:
    - Capacity
    - Current population
    - “Coolness” → some bars are known as cool in general, those will be more likely to attract crowds

Non-Functional Requirements:

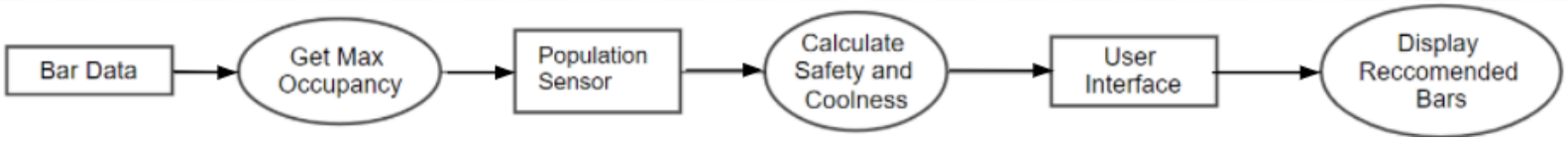
- Usability→ how easy is it to use
- Reliability→ proper response time

System Modeling:

Sequence Diagram:

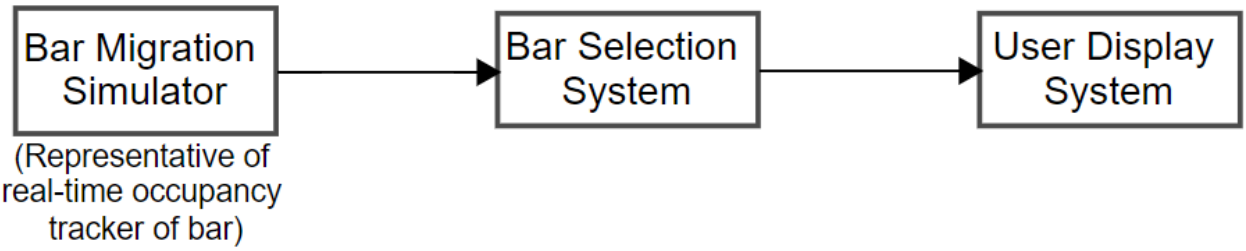


Activity Model:



Architectural Design:

Simple Box and Line Diagram of System Architecture Layout:



The Bar Migration Simulator:

- This system is responsible for creating an accurate representation of how populations move through time from bar to bar.
- It takes in different factors like the size of the bar, its overall reputation and hotness (whether the public as a whole thinks it's the place to be), as well as what percentage of the population is more likely to move.
- This simulator was created to make sure that the next system was tested correctly and dealt with realistic data.

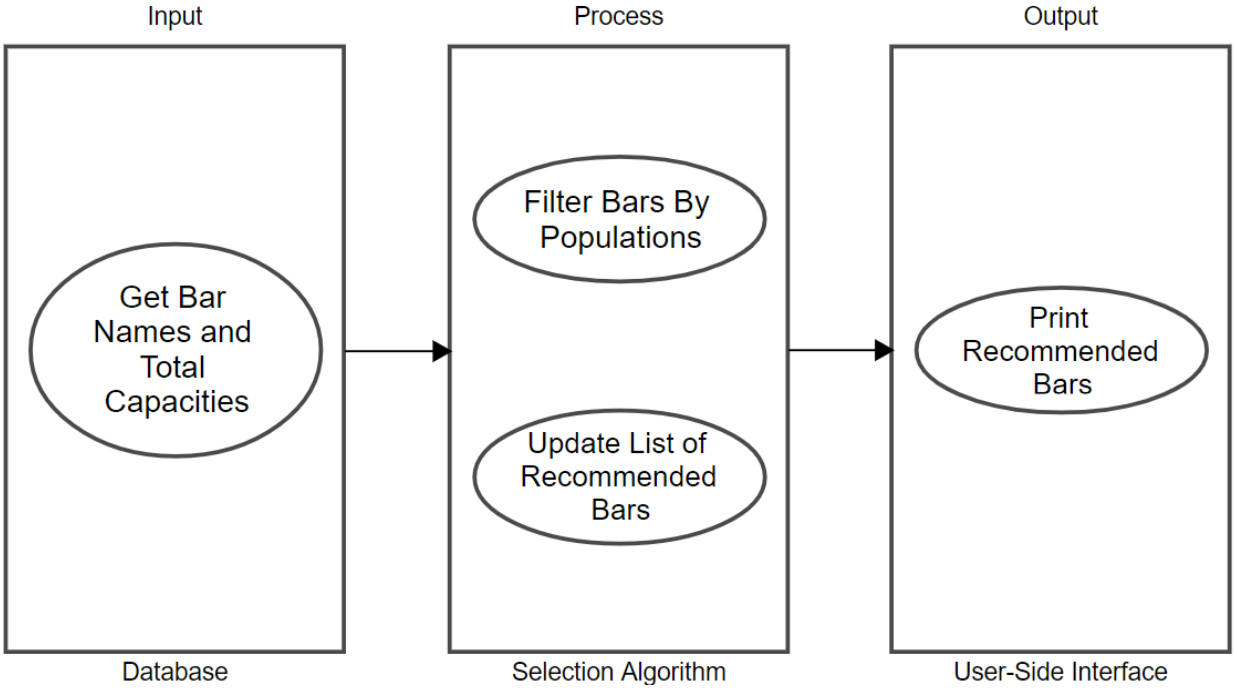
Bar Selection System:

- This system implements an algorithm that selects the safe bars that are also fun.
- This system will fulfill the user requirements
- If a bar is within 60 and 80 percent capacity it will be recommended

User Display System:

- This system displays the chosen bars if they exist.
- It is important for the user to be able to view this information to guide their decision making process
- The Usability of this system is designed so there are minimal errors made by the user, because Display gets its information input from the Selection System. There is very little input from the user aside from the refresh that ensures most up-to-date capacity results. The minimal user interaction helps narrow down variable inputs and minimizes user side bugs.

Software Architecture Model of System:

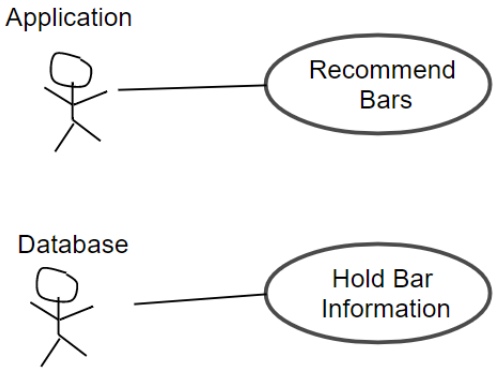


Design and Implementation:

Object Classes:

The Object Class we had to create for this application was the Bar class. It had to hold all the Bar information including max capacity, current population, and name. A database to get the static Bar name and max population data is also needed for the application to work properly.

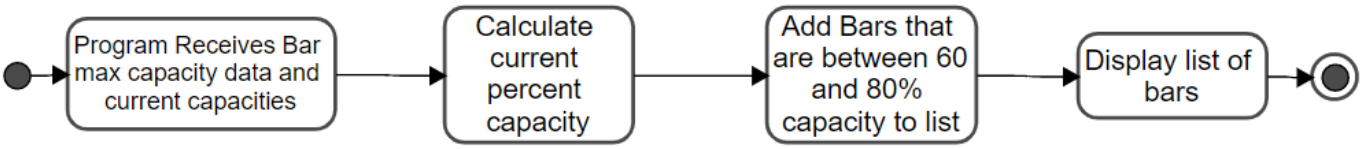
Use Cases:



Use Case Description-Report

System	**Covid Safe Bar System**
Use Case	Recommend bars that are safe and cool
Actors	Bar Database, Recommendation Application

Description	The Application retrieves a list of bars and their maximum capacities from the Bar Database. The Application performs an analysis on the bars taking into account their maximum capacities, their reputation, and current population and sends a list of approved bars that are within 60% and 80% capacity. This information is reported to the user.
Stimulus	The user requests a bar recommendation via the application interface. Then, the Application requests the bar data from the database.
Response	The database provides the app with data that Application uses to filter bars and sends the recommended data to its user interface.
Comments	Application is required to keep updating results of population migration from bar to bar when application is refreshed.



The above diagram illustrates the process of how the program makes its decision on which bars to recommend.

Software Testing:

Our Testing Process:

- A simulator was created that modeled how populations migrate between bars throughout the night to test how well our application dealt with different inputs. This simulator made accommodations for bar size, the current population within the bar, and whether the bar was popular. The simulation helped replicate real nightlife shifts that the system might encounter in real life.
- The product was implemented well. It is easily changeable to incrementally be developed and is written concisely. It is well documented throughout and easy to understand the functionalities of each segment of code. This system works as the user requirements intended. It displays bars within the 60 to 80 percent capacity range for maximum fun whilst taking into account safety precautions due to COVID-19.
- System works as intended for common test cases by displaying Bars within capacity range. Common/expected test cases are when there are enough bars that fit the criteria to be recommended. Expected system inputs are when there is a relative distribution of a population across all bars, where some are more populated than others.
- Uncommon test cases are when all the bars are either below the desired range or above it. This is normally unlikely. This is where the system doesn't perfectly give a desired result. Because none of the bars fall in range, none will be recommended to the user. However, the intent of the application is to see which bars are the best for safety and fun. Therefore, the analysis of the bars to recommend must be relative amongst the bars. Currently our program will display no bars if all of them have capacities out of the 60 to 80 percent range. This isn't optimal and will need to be adjusted in the next sprint.

Evaluation:

- The process models were followed to their defined specifications. The current parameters of 60 to 80 percent capacity work well for expected cases. However, the recommendation algorithm needs to be tweaked to account for cases where there are too many bars that fit the criteria, or no bars that fit the criteria. The good part about our program is that it easily supports a change in the recommendation algorithm. Our system architecture operates in a way that the bar data analysis is run separately from the user display and input database, so neither will be affected by changes. This will help aid in working with legacy software down the line. Because future iterations of the algorithm will be able to function with the display program section. Also, If changes are made to the display output, it will not negatively impact the algorithm if it needs to stay the same. Anticipated changes in the next sprint include a user display that helps the user make better decisions through a more visual conceptual model of the data (most likely in the form of a graph). The other major change that will likely be made is to the algorithm that chooses which bars to recommend. A more dynamic approach that makes recommendations based on relative safeness of all the bars in the system will better fulfill the program requirements.

Sprint 2

- Have an interval that a certain number of will change bars
- Update data in list on each interval
- Calculate and change recommendations based off of new populations
- Create GUI to display recommendations
- Allow user to refresh data and exit program

Requirements Engineering:

Functional Requirements:

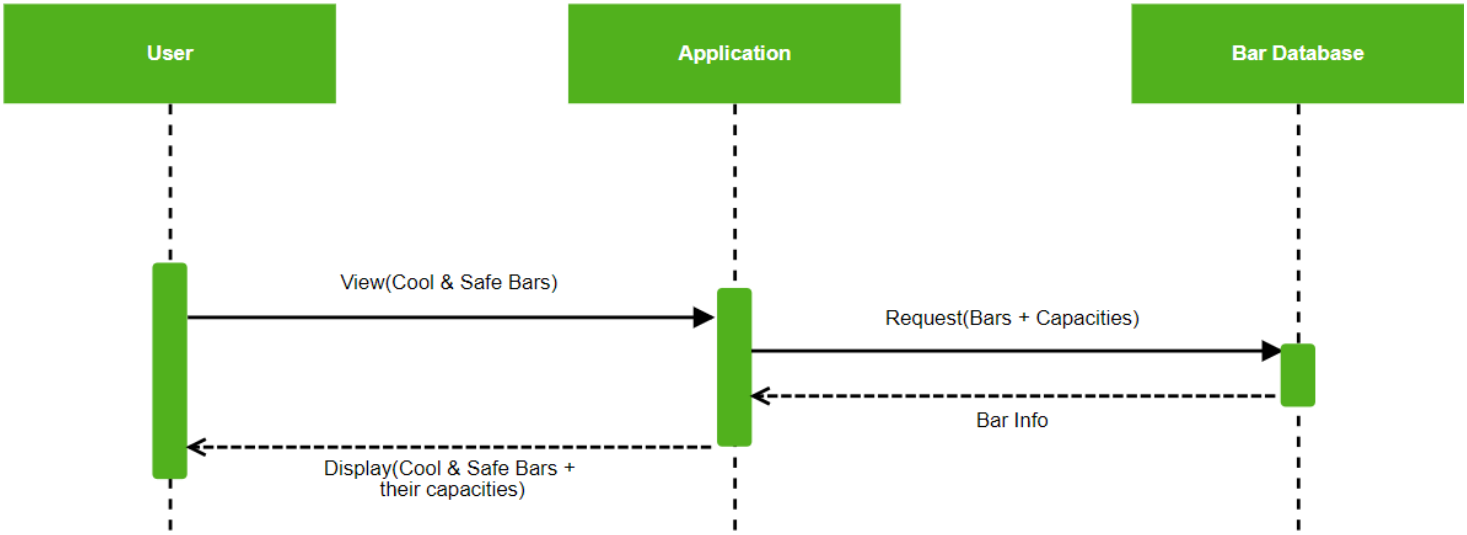
- System shall be able to generate recommendations for bars based on capacity
- System shall be able to simulate people moving from bar to bar, running multiple times over the course of the simulation
- Recommend a reasonable amount of bars (will probably aim for around 4). Want to recommend at least one or two even if the conditions are not great because you are helping the user make the best decision.
- Display current capacity of recommended bars because conditions of bars might not be ideal but you want to still give recommendations. Displaying capacities alongside the bars will allow the user to make a more informed decision. Display should be a more visual representation, like a graph.
- Recommend bars starting at 65-70% occupancy, but increase the size of the window if there are no bars which satisfy the threshold.
- (basic GUI) A pop-up window displaying information should be displayed when run, allowing the user to reshuffle population (displaying new graph) and exit on demand

Non-Functional Requirements:

- Speed/efficiency
- Scalability allowing the system to accompany a large number of bars (potentially infinite)
    - Make sure not to overwhelm the user with information. Give a more concise list of bars.
- Simplicity allowing user to easily access information

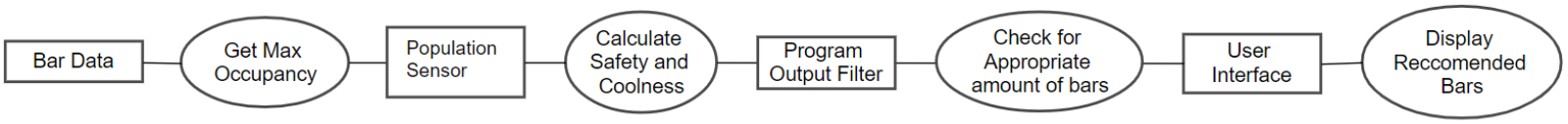
System Modeling:

Sequence Diagram:



The sequence diagram stays relatively the same. The main difference is that now the capacities are also queried.

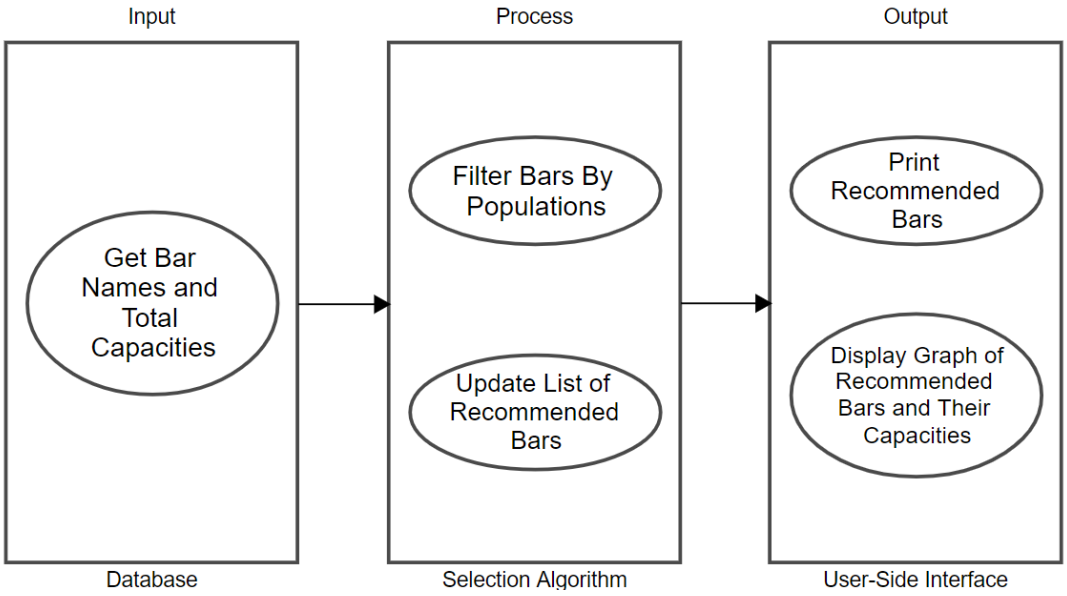
Activity Model:



The activity diagram has a new step where the Application will check if there is an appropriate amount of bars to be displayed to the user. In the previous model, this check didn't exist and as a result the program displayed way too many bars if a lot of them met the very basic criteria in the first sprint.

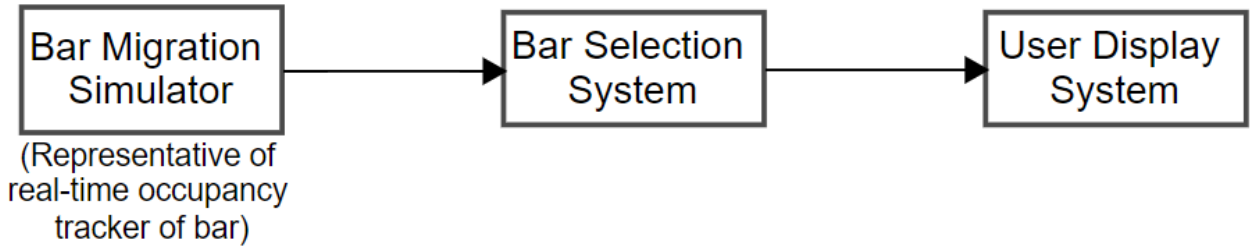
Architecture Design:

Software Architecture Model of System:



The Architectural Design has changes to the output where there will be a graph attached to the output that will display capacities so that users have more information to guide their decision making.

Simple Box and Line Diagram of System Architecture Layout:



The Bar Migration Simulator:

- This system has remained relatively unchanged. It generates migration patterns of people from bar to bar every time a user refreshes their program.

Bar Selection System:

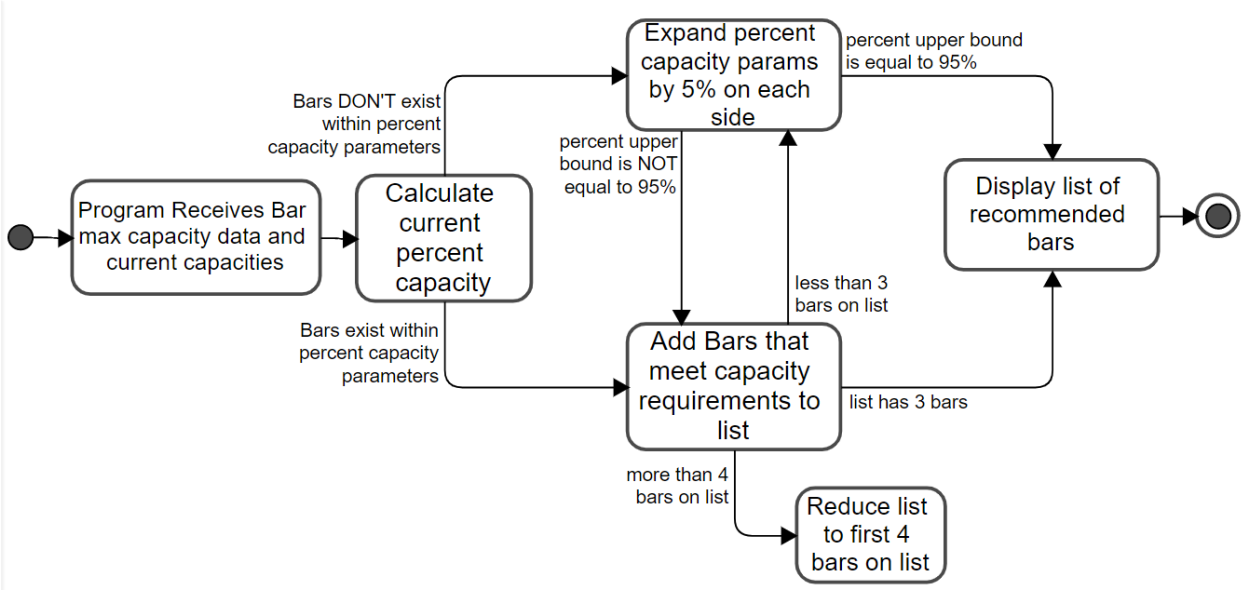
- This system implements an algorithm that selects the safe bars that are also fun.
- This system will fulfill the user requirements more accurately and dynamically than before.
- If a bar is within 65 and 75 percent capacity it will be recommended. If there are too few bars within this range to make a well rounded recommendation, the window of allowable bars will increase by 5 percent until enough bars are recommended or the capacity reaches a max of 95 percent which is most certainly no longer safe.

User Display System:

- This system displays the chosen bars if they exist.
- It is important for the user to be able to view bar capacity information to guide their decision making process. So that has been an added feature.

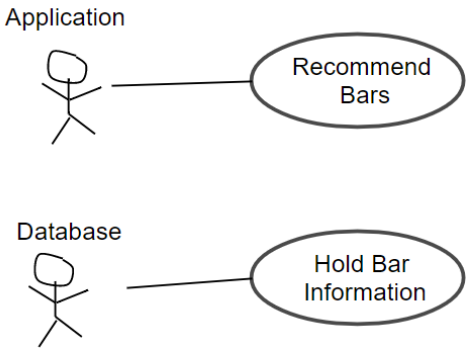
- The Usability of this system is designed so there are minimal errors made by the user, because Display gets its information input from the Selection System. There is very little input from the user aside from the refresh that ensures most up-to-date capacity results. The minimal user interaction helps narrow down variable inputs and minimizes user side bugs.

Design and Implementation:



Above is an updated process flow of how the recommendation algorithm dynamically chooses bars in the second sprint version in the program.

Use Cases:
The use cases of this program have not changed because the main requirements of the system have stayed relatively the same.



Use Case Description-Report

System	**Covid Safe Bar System**
Use Case	Recommend bars that are safe and cool
Actors	Bar Database, Recommendation Application

Description	The Application retrieves a list of bars and their maximum capacities from the Bar Database. The Application performs an analysis on the bars taking into account their maximum capacities, their reputation, and current population and sends a list of approved bars that are within 65% and 70% capacity. If there aren't enough bars in the list to recommend. The program incrementally expands its capacity range to accommodate for relative bar capacity. This information is reported to the user.
Stimulus	The user requests a bar recommendation via the application interface. Then, the Application requests the bar data from the database.
Response	The database provides the app with data that Application uses to filter bars and sends the recommended data to its user interface.
Comments	Application is required to keep updating results of population migration from bar to bar when application is refreshed.

Software Testing:

Our Testing Process:

- Our Testing process was relatively the same as the one in our previous sprint. We used the simulation to test expected inputs into our program. Like in the last sprint, the program was able to pass these test cases without any issues.
- The main difference came to when testing edge cases. We were able to navigate inputs where bar capacities were really low or high much better. This is because our more dynamic algorithm was able to adapt to these large variations in the data. The program was able to give recommendations when the conditions weren't perfect. Implementing the graph display also helped a lot in these cases because then users could compare the capacities of the different bars themselves. Displaying the data in the form of a bar graph is an easy way for the user to make a visual comparison of the recommended bars. This new data helps users make more informed decisions.

Error handling:

- When there was an error displaying the output, we used regression testing to revert our changes to a previous version. The reason this error occurred was that the new graph was causing the output to glitch, so we went back to an older version. Then we step by step integrated the changes back in to see how this affected the output. We then found out that we were resetting a variable that was not allowing us to display all of the bars correctly. Once this bug was fixed. The system was able to display the intended output with both the old selection algorithm and with the new one.

Evaluation:

In this sprint many of the imperfections from the last sprint were fixed. The algorithm was able to adapt to unexpected bar capacities much better. This helped us fulfill the main requirements much better than the previous sprint. We also put in a provision to cap the total amount of bars displayed which is extremely helpful when it comes to scalability. If for example one hundred bars fit the desired criteria, that doesn't help the user make a decision. Therefore by limiting the amount of bars displayed users have an easier time making a choice. The last

major change to the program was the creation of a graph that is displayed to the end user. This also helps the user make decisions. By pressing a button within a basic GUI, movement of the population between bars can be simulated, and a new graph displaying recommendations is generated over and over until the user presses an "Exit" button. The usability of the software has remained high because of the minimal interaction allowed from the user to the program. In the future, possible ways to expand on this application might include location settings that take in proximity of the bars to the user as well as the users past history of bars. That way the program would be able to make more personalized recommendations. However, as of right now, the program fulfills the requests of the client to make a program that recommends safe and fun bars to users.