**Section 1 – Features**

The program first requires the user to choose which CSV file located in the “data/” directory (same level as “src/”) they would like to load from a dropdown menu. This immediately shows the table with all the data. There is a search button which searches for an exact match, and this also has a reset button to show all of the data again. The default sort is by ascending ID, but this can be changed to any combination of ascending and descending with the column names from the dropdown menu – the reset button sets it back to the default.

The search and sort can also be done simultaneously, but it has to be in the order search then sort. The user can view a bar chart based on the frequency of values in a particular column – this only works for “BIRTHDATE”, “DEATHDATE”, “SSN”, “SUFFIX”, “ZIP” – as well as for a patient’s age which is calculated based on birth date and death date. The graphs generated will only display data based on the results of what is in the search bar (empty means all the data will be shown).

The user can also download a JSON file of the data currently displayed – even after both searching and sorting, and after row operations. The user will be prompted where to save the file and for the filename. The user can also edit any row, changing the values for that row’s column except for the ID), delete any row, or add a new row (a unique ID is automatically generated). All of these changes will be reflected in the patientList page. By default, every row operation (edit, delete, or add new row) results in a new CSV file being created – the file name will be the current file name with the current timestamp appended, and it is saved into the “data/” directory; the operations persist until the user goes back to the home page and selects a different file. When adding a new row, there is a “clear” button which clears all fields except the ID field. There is a similar “revert” button when editing a row, which resets the values to what they were before editing that row began.

**Section 2 – Describing & Evaluating my Design & Programming Process**

I tackled the project in quite a straightforward fashion – I went through the requirements in order, as this gave me a good idea of what to work on during the next stage of the development process. I began by simply writing the classes with the methods described in the first couple of requirements, but when it came to writing the DataLoader class, I knew that I really only required a single method, but I was unsure as to whether I should make it static or not. Looking at the following requirement, which stated that the Model class should manage a Data

Section 2: Describe and evaluate your design and programming process. You should reflect on how you went about designing your classes, why they are appropriate classes, whether you have used good OO design practice (e.g., good use of abstraction, cohesive classes), and the overall quality of your work (you decide the criteria for this)