**Nosophobic**

**Design Specification**

**Version 1.1**

Group09:

Owen Mitchell

Bartosz Kosakowski

Anthony Mella

Hamid Yuksel

Comp. Sci. 2XB3 – L01

McMaster University

**Revisions**

* Owen Mitchell (001450204) – Project Leader, Programmer, Tester
* Bartosz Kosakowisk STD#– Log Admin, Programmer, Tester
* Anthony Mella STD# – Programmer, Tester
* Hamid Yuksel STD# – Programmer (Front end), Tester

*By virtue of submitting this document we electronically sign and date that the work being submitted by all the individuals in the group is their exclusive work as a group and we consent to make available the application developed through [CS]-2XB3 project, the reports, presentations, and assignments (not including my name and student number) for future teaching purposes.*

*Revision History*

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Contributions**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Role(s)** | **Contribution(s)** | **Comment(s)** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Executive Summary**

The problem being addressed in this report is chronic mental and physical health in the United States. While these problems are not solvable by a single initiative, we are concerned with reducing the prevalence of these issues.

Nosophobic is an application designed to provide information about chronic disease indicators (CDIs) in The United States. Using public data collected from the Centers for Disease control and prevention, CDI risk factors will be calculated and used to rank regions.

The core functionality of Nosophobic is simple: the application must deliver appropriate information about the CDIs distributed across America based on user selected inputs.

Inputs:

The user will specify values for either, or both, of the following:

1. State (i.e. one of the 50 American states)
2. Disease Topic (e.g. Alcohol, Cardiovascular, Oral, …)

Outputs:

There will be two information sources outputted for every query with some modification based on the input type:

1. A graphical map of the United States colorized by state, based on the severity of CDIs, which will be referred to as the “heatmap” for the remainder of this document.
2. A table of textual information, which will be referred to as the “information table” for the remainder of this document.

**Table of Contents**

|  |  |
| --- | --- |
| **Section** | **Page #** |
| Title Page | 1 |
| Revisions | 2 |
| Contributions | 3 |
| Executive Summary | 4 |
| Overview | 5 |
| Module Relations |  |
| Semantics |  |
| Review |  |

**Overview**

Nosophobic will be decomposed into the following hierarchy.



The first decomposition is the separation of “front-end” user interface, and the “back-end” modules.

Back-End

The “back-end” contains 3 distinct functions in order to facilitate the application.

1. “CDI”: The data type used to store the information in a single data entry.
2. “Data Collection”: The module that initializes the CSV data into a collection of “CDI” objects
3. “Processing” A collection of libraries that facilitate the various functions of the application. Including sorting, filtering, and graph processing

CDI:

The CDI data type uses some, but not all of the information present in the original data set.

Because of this the CDI data structure has been decomposed into its own module to enable possible modifications to the information collected from the original data set.

CDI contains 10 values.

1. Year : int (The year the data was collected)
2. State : string (The state being surveyed)
3. Topic : string (The disease addressed in the survey)
4. ValueAv : float (The average value of the survey)
5. ValueHc : float (The “high confidence” value)
6. ValueLc : float (The “low confidence” value)
7. Lo : float (Longitude)
8. La : float (Latitude)
9. Danger : float (A measurement of the risk posed by the survey)

CDI contains 11 methods.

1 – 10. Accessor methods for all state variables.

1. getDanger() : Void (An evaluation function that assigns the danger of a given CDI)

Data Collection:

Sort:

Sort is a static library that is used for sorting data elements by various criteria.

Sort contains 1 method.

1. sort(Comparable[] cdis, String sortBy) → Comparable[]

The string sortBy is used to determine what criteria is used in the sort, and will be one of (Disease, State, Danger). This structure is employed to avoid code duplication, as all the sorts will be using the same sorting implementation.

Filter:

Filter is a static library that is used for isolating specific sections of data.

Filter contains 2 methods.

1. filterDisease(ArrayList<CDI>, string disease) → ArrayList<CDI>
2. filterState(ArrayList<CDI>, string state) → ArrayList<CDI>

The filterDisease method returns a new ArrayList<CDI> that contains only the CDI objects who’s topic matches the disease input.

The filterState method returns a new ArrayList<CDI> that contains only the CDI objects who’s state matches the state input.

Graph:

**Module Relations**

**Semantics**

Sort.java is a static class that sorts the data based on several parameters; it can sort the data alphabetically by state, by disease, by year, or by the danger level the disease poses. These options are based on the criteria that the user can search for. This class is static because we do not need for there to be any more than one instance of it; it just modifies the data it is given, meaning that it can be called whenever we need to sort some data and not require any instance to be present before doing so. This class implements a quick sort because it is slightly faster than any other sorts, such as the merge sort. Even though both it and merge sort have the same time complexity (O(nlogn)), merge sort is slightly slower since it must then merge all the subarrays together. It is also more memory-efficient because it is an in-place algorithm, meaning that we do not need to create any extra data structures to perform the sort (such as a heap for a heap sort, or subarrays for merge sort).

The filter class implements a linear search algorithm. The runtime is O(n), this is the most efficient searching implementation given the way the data is organized. The data is first sorted and then traversed through by the filter class, then the specific elements are returned as an Arraylist. Since the data is passed to the filter class already sorted, it was tested using binary search to find the items, but since there are multiple items in the list, we would have to run binary search for each item in the list. This would have given us a runtime of O(nLog(n)). Comparing these two implementations, the linear search algorithm was faster given the size of the data being passed to the filter class. This is the reasoning behind implementing a basic linear search algorithm to our filter class, it is the most efficient option.

There are two methods in the Filter class. The first method filterDisease, filters the CDI's based on the disease they have. It will return an ArrayList containing all the CDI's that contain the specific input disease. The second method filterState, filters the CDI's based on the given state within the USA. It will return an ArrayList containing all the CDI's that are in the specified input state.

**Review**