SafeWayz
Requirement Specifications

Lab Section Number:	**L02**
Student Names:	*Jash Mehta, Aditya Sharma, Anando Zaman, Zackary Ren, Daniel Di Cesare*
Group Number:	**02**
Student McMaster Emails:	*mehtaj8@mcmaster.ca, shara24@mcmaster.ca, zamana11@mcmaster.ca, renx11@mcmaster.ca, dicesard@mcmaster.ca.*

Table of Contents

Glossary

App: An abbreviation for “application”. It is essentially a software product designed for mobile devices. [1].

API: An abbreviation for “Application Programming Interface”. It is a software product that contains definitions, protocols and written code that is frequently used to allow for modular programming [8].

BFS: An abbreviation for the shortest-path finding algorithm “Breadth-first Search”. This algorithm is generally used to find the shortest-route between two points. [7].

DFS: An abbreviation for the path-finding algorithm “Depth-First Search”. This algorithm is generally used to find any route between two points [10].

Efficient: This term is defined by the time and space complexity of all the algorithms that the software product is composed of. A software product is said to be efficient iff (defined below) the space/ time complexities are at a global minimum [9].

IFF: If and only if

JSON: An abbreviation for the word “JavaScript Object Notation”. This is a format that is used for storing data. It is similar to a hashmap/ hashtable [4].

Optimal Path: It is defined as the safest route between two locations specified by the user.

UI: An abbreviation for the word “User Interface” [3].

UX: An abbreviation for the word “User Experience” [11].

Weighting: Also termed a ‘cost’; impacts the viability of a point within a dataset [6].

Dataset: An organized collection of data [5].

Dijkstra’s Algorithm: Efficient solution for finding the shortest route between two points in a graph [2].

Overview

SafeWayz provides a needed safety-oriented navigation system to its users. It will support the San Francisco region. Key features of this application will include providing optimized routes between two points utilizing previous crime trends, minimizing the number of intersections, and crime weight specific filtering of routes.

Each of the below sections will explain key aspects of engineering design and business requirements of the application. The Domain will consist of the project scope, information on stakeholders, and the various assumptions when designing the application.

Functional requirements will explain processing of the inputs for route guidance, filtering and creating a graph from a dataset. It will also explain the time complexity and performance analysis of the various algorithms that will be used to search,sort, and traverse through the graph.

Within the development cycle, non-functional requirements such as reliability, accuracy, and performance will be points of focus. Constraints surrounding UI, physical, and operation will also be heavily considered. These aspects will be explained in the non-functional requirements section of the document.

Attention will be put into ensuring the program is modular and maintainable for the foreseeable future. Testing will be completed for each stage/component of development. Maintenance and future upgrade support will be discussed in the development and maintenance portion of the document.

Domain

Purpose

The purpose/goal of our application, Safewayz, will be to provide the general public with the shortest and safest route between two locations. Safety when navigating in large cities can be challenging. Crimes and violence become of great concern as cities get larger. There is no software product on the market that provides users with a path with their safety as the number one priority. Our hope is to provide the general public with a tool (mobile application) that can be leveraged to make them feel, travel and ultimately be safer.

Stakeholders

The stakeholders will consist of the generic public, local authorities, and the government. The general public will be a stakeholder because they are our target demographic and the primary user of the product. The local authorities and government will also be marked as stakeholders because of the relationship they would share with our product and the general public. To elaborate, when the public uses the product, there should be a reduction in the number of crimes because the application will provide users with the safest route. This decreases the potential number of crimes and as a result, will decrease the number of cases/ investigations the local authorities and government would conduct.

Relevant Facts and Assumptions

The facts and assumptions will consist of a few topics. One is that of the way the application will be developed. It will likely be a native mobile application to utilize the core components of the specific operating system itself versus a Hybrid Mobile App. The dataset JSON will likely be stored locally versus on a cloud platform to increase reliability if a database were to shut down for reasons such as maintenance.

Product Scope

In a more detailed sense, the primary purpose of the application will be to find the optimal path between two points where the optimal path is a function of path length and crime weighting. As a deliverable, the application will be providing the user with the optimal path along with a few alternatives.

Though this will be the primary purpose, there will also be additional use cases. One, is that it will aid local authorities in locating areas with greater crime which can be of use to monitor/patrol the activity in these areas. Second, the application will be used to redirect users to adopt different safer routes that they otherwise might have not known. Overall, it will increase the safety of the generic public while redirecting authorities in high crime locations to reduce the violence in those locations.

Functional Requirements

Functionality

- Inputs: start and end data points via street intersections
- Validity check: Check if start and end points provided exist within the dataset
- The JSON data will get parsed by the hashify methods to access at O(1) time for improved performance
- Sorting will be done in the worst case O(NlogN) time to sort the crime weightings in ascending order for convenient access.
- A shortest path finding algorithm will use the sorted data to determine the optimal path at the worst case O(ElogV) time where E is the number of paths with a crime weighting while V is the number of available streets.
- Output: Google Maps API will take the new found shortest path and plot it for the user to see on screen. This will be the output UI that will allow the user to visually see and interact with their navigational route
- Exception handling: Invalid starting position, invalid destination, dataset does not consist of the region’s streets, no valid path between the locations.

Non-Functional Requirements

Reliability

Safewayz will be a reliable software product because it will be rigorously tested prior to deployment. Following an agile methodology, Safewayz will be subject to unit testing upon each iteration and will be exposed to integration testing prior to assembling each module. This will ensure that the software product will output the correct response. Furthermore, potential failures will have been anticipated and failsafes will also be incorporated. Thus, ensuring the product will fail safely without negatively impacting the user.

Availability

Safewayz will be guaranteed to have 100% uptime because it will leverage only a few 3rd party products, thus minimizing downtime. For example: Safewayz will use a local dataset instead of a cloud database. This will avoid the possibility of potential downtime caused by issues with a third party cloud service.

Performance

Safewayz will be designed with mobile performance in mind. The product will leverage efficient graphing, sorting and searching algorithms that contribute to high performance. It will use a DFS graphing algorithm which will provide a space/ time complexity of O(S) and O(N) respectively where N is the number of nodes and S is the number of streets in the path stack. It will also use the QuickSort algorithm which provides a space/ time complexity of O(N) and O(N Log N) respectively. Finally, the sorting algorithm to be used will be binary search as this provides a space/ time complexity of O(1) and O(Log N) respectively.

With regards to the data structure being used, an Adjacency List will be used to represent the graph as it allows graph algorithms to leverage its O(N) time complexity for navigating all paths. Another option will be to use an Adjacency Matrix, but this representation restricts graph algorithms at $O(N^2)$ which is suboptimal when compared to an Adjacency List. The Adjacency List will be implemented as a hashmap rather than arrays as this provides O(1) search, insert and delete time which are significantly more efficient than that of arrays or lists.

Scalability

Currently implemented systems are designed to be used with ever-increasing sizes of datasets. The graphing algorithm used will be optimized for search time by using DFS and Dijkstra's algorithm depending on the situation at hand. In addition, a growing user base will not impact the current design as crime data will be pre-sorted into a traversable graph. All computation will be done locally through the users device, and then sent to Google Cloud Platform for displaying their calculated route.

Security

SafeWayz will be designed such that all code respects information hiding. Only basic interfacing methods will be available outside of any given module. All local variables will be privatized; and server/user information will be stored such that it will be inaccessible by the user directly. This ensures security and information hiding.

Human-Computer Interface

The human-computer interface will be the UI of the product. The UI will have a few components that leverage visual cues such as icons rather than text. This will ensure excellent user experience to follow the flow of the application.

Operating System Constraints

Safewayz will depend upon the Google Maps API for visualizing the path between two points. As a result, the product's operation will be constrained by the usage limits of the API. Currently, the product can only make 2500 API calls per day.

Furthermore, Safewayz will also be constrained by the size limitations of an Android application. The maximum allowable size will be 40 MB and as a result, the product size must be well-controlled. Unnecessary features and data will be excluded and emphasis on accomplishing the product’s primary function will be placed.

Finally, Safewayz will be constrained by the support of the operating system. The application will be developed to support Android Version 5+. While this encapsulates 70% of all mobile devices, there still remains 30% of the world’s population that leverage non-android mobile devices. As a result, the product will be constrained to a set demographic of users. However, in order to increase the portability of the product and eliminate this constraint, another version of the product will be released using Flutter to make the product platform/environment agnostic in the future.

Development and Maintenance

Software Engineering Principles

Separation of Concerns:

- Different concerns will be isolated and considered separately. This will improve the ability to isolate particular bugs/issues that may arise in the future with a greater degree of efficiency.

Modularity:

- In close relation to separation of concerns. This will reduce the complexity of the system by considering each aspect as a smaller component of a larger system. Each module will be used together to solve the larger problem at hand.

Abstraction:

- Only allows access to certain information while hiding the access to the rest that will likely not be needed externally.
- The application will give access to certain information that is useful to the user.

Anticipation of Change:

- allows for future iterations to be adaptive and correct during the maintenance process.
- Prevents system failure from additional features or changes to modules.

Generality:

- Problems will be solved in a more general manner to improve ability to solve complex problems.

Quality Control Procedures

Unit Testing

- JUnit tests will be done on the modules to verify correctness.
- Specifically the sorting, searching, graph traversal (BFS and DFS), and Dijkstra's shortest path modules will be tested
- The test cases will consist of normal, boundary, and exception cases to identify behaviour of the modules.
- Test inputs will be supplied via input text files and test case pass/fail outputs will be in an output text file.
- Assertions will be made to compare with the expected results for PASS/FAIL state.
- The tests ensure improved exception handling that can prevent unusual behaviour and application crashes.

Integration Testing

- Necessary to verify that software modules work in unity.
- This will ensure that the cross compatibility of the modules are valid and complete
- Top-down approach in incremental testing will be carried out to test higher-level modules first(ie; BFS uses graph, quicksort uses graph, etc).
- Prevents idle testing time compared to big-bang testing.This will result in a more agile approach.
- Normal, boundary, and exception cases will be executed to test the behaviour of the modules in relation to each other.
- This will allow for software robustness that is crucial when modules work together as part of a larger system.

System Testing

- Will ensure all the modules work in unity as part of a larger software application/system with correct end behaviour.
- Performance tests will be executed to understand the time needed to run certain parts of the application. This impacts the usability of the product.
- Usability testing will consist of user surveys to focus on the users ease and learning curve in using the application
- Use of BETA testing for usability and bugs that will be reported back to the developer responsible for a module.

System Maintenance:

- Version control will be used to maintain production and non-production builds.
- OS upgrades and features will be implemented in future builds by first testing in non-production builds with new Android API levels.
- Enhancements and new features will be thoroughly tested in each non-production build prior to releasing a stable production build. Testing will follow Quality control procedures above
- Bug reporting features will be available via Google Play Store to report on stability issues or any other feedback. Bugs will be resolved and tested in non-production builds.
- If module incompatibility or wide application crashes occur, a rollback will be performed to the previous stable build.

Modularization:

- Application of separation of concerns design principle.
- Development process will be separated into various components based on designated goals.
    - Data cleansing and quality.
    - Construction of data structures, including graphs and hashmaps.
    - Preparing functional algorithms, including BFS, DFS, Quicksort, binary search, and Dijkstra's algorithm.
    - Development of the user client on the Android platform.
- Isolation of components allows for faster debugging and testing of functionality.
- Modules will be designed with generality in mind to allow for code to be reused in future features and other components.

References

[1] “App,” *Merriam-Webster*. [Online]. Available: https://www.merriam-webster.com/dictionary/app. [Accessed: 15-Mar-2020].

[2] P. E. Black, “Dijkstra's algorithm,” *NIST*, Sep-2006. [Online]. Available: https://xlinux.nist.gov/dads/HTML/dijkstraalgo.html. [Accessed: 15-Mar-2020].

[3] “Chapter 4 Graphical User Interfaces,” *Chapter 4. Graphical User Interfaces*. [Online]. Available: https://cs.stanford.edu/people/eroberts/jtf/tutorial/GraphicalUserInterfaces.html. [Accessed: 15-Mar-2020].

[4] “Introducing JSON,” *JSON*. [Online]. Available: https://www.json.org/json-en.html. [Accessed: 15-Mar-2020].

[5] OECD Statistics Directorate, “DATA SET,” *OECD Glossary of Statistical Terms - Data set Definition*, 25-Sep-2001. [Online]. Available: https://stats.oecd.org/glossary/detail.asp?ID=542. [Accessed: 15-Mar-2020].

[6] R. Sedgewick and K. Wayne, “Shortest paths.,” *Princeton University*, 16-Nov-2018. [Online]. Available: https://algs4.cs.princeton.edu/44sp/. [Accessed: 15-Mar-2020].

[7] R. Sedgewick and K. Wayne, “Undirected Graphs,” *Princeton University*, 16-Apr-2019. [Online]. Available: https://algs4.cs.princeton.edu/41graph/. [Accessed: 15-Mar-2020].

[8] “What is an API?,” *Red Hat*, 2020. [Online]. Available: https://www.redhat.com/en/topics/api/what-are-application-programming-interfaces. [Accessed: 15-Mar-2020].

[9] D. Duran, “Algorithm Efficiency,” *Algorithm Efficiency*. [Online]. Available: http://www.cs.kent.edu/~durand/CS2/Notes/03_Algs/ds_alg_efficiency.html. [Accessed: 16-Mar-2020].

[10] “DFS algorithm,” *Programiz*. [Online]. Available: https://www.programiz.com/dsa/graph-dfs. [Accessed: 16-Mar-2020].

[11] “What is User Experience (UX) Design?,” *The Interaction Design Foundation*. [Online]. Available: https://www.interaction-design.org/literature/topics/ux-design. [Accessed: 16-Mar-2020].