**Project Completion Report**

**St10019602 Toby Dwyer**

**Overview**

The **Municipal Services Application** was designed to help users manage and track service requests for municipal services such as streetlight repairs, garbage collection, and other maintenance tasks. The primary goal was to create an interactive system that allows users to add, search, and view the status of service requests efficiently.

The key features implemented include:

* Adding and managing services within the application.
* Reporting issues with the relevant details.
* Viewing Events and announcements.

The application was developed using C#, WPF, and .NET, ensuring that the user interface was easy to navigate, and the backend logic was efficient and scalable.

**2. Task 3 Implementation: Challenges and Solutions**

Task 3 required us to implement a Service request feature. This feature used an AVL Tree for efficient service request management and a Graph to represent relationships between requests. The following challenges were faced during its implementation:

**Challenge 1: Designing the Graph for Related Requests**

* **Issue:** Initially, the application was very basic but using the graph to then implement a related services feature was difficult at first since it was difficult to define what would relate the services to one another.
* **Solution:** After analysing the requirements, I decided to link requests that share common keywords in their descriptions. I used the Contains method to match keywords between requests, allowing us to create edges in the graph between nodes that shared similar descriptions.

**Challenge 2: Efficiently Traversing the Graph for Related Requests**

* **Issue:** After setting up the graph, it became necessary to implement an efficient way to traverse and retrieve all related requests for a selected service. It was important that this transversal remained efficient as the number of values grew for the app to work appropriately at larger scale.
* **Solution:** The graph was implemented using an Adjacency List, which allowed for O(1) average-time lookups of neighbours. I used Depth First Search to traverse the graph and retrieve all connected components for a node. This method ensured that even with a larger scale transversing the graph remained efficient.

**Challenge 3: User Interface Integration**

* **Issue:** Displaying related service requests on the user interface presented challenges in ensuring that the results were presented clearly and promptly.
* **Solution:** I utilized WPF’s ListView control to display the results. The ListView was dynamically updated based on the results of the graph traversal, ensuring that the user always saw the most relevant service requests.

**3. Key Learnings and Insights**

Throughout the development of this project, several key learnings were acquired, both in terms of **technical skills** and **problem-solving approaches**:

* **Efficient Data Structures:** The project highlighted the importance of choosing the right data structures to best achieve a task. The AVLTree proved best for efficient management and retrieval of requests, while the Graph allowed there to be relationships set up between all the requests to allow for more features. Understanding how all the different data structures work was a big learning point for me.
* **Graph Theory and Traversal:** Implementing the graph-based relationships helped deepen my understanding of graph theory. Specifically, learning how to represent relationships with an adjacency list and apply algorithms to improve the functionality of the graphs was also very interesting.
* **UI/UX Considerations:** Designing the user interface to handle dynamic data, such as related requests meant I had to improve my knowledge on what were the best components to use such as list views and using a page setup within the main frame with navigation set up between the pages for the best user experience.
* **Problem-Solving Approach:** One thing that this project taught me about problem solving is how setting up a structured plan will help in the long run of the project. Initially between tasks 1 and 2 I had no plan for long term development, and it meant changes made in task 2 were sometimes difficult to integrate but I changed some of my methods and task 3 went considerably smoother from a development standpoint.
* **Programming Techniques:** In this project I used techniques such as OOP for integrating multiple classes to perform a single task and using LINQ queries to help improve the efficiency of my coding for tasks such as manipulating and filtering data throughout the project.

**Technology Recommendations**

**1. Database Integration**

**Recommendation:** Integrating a database would allow that would allow for the persistence of data when the application is run is a big feature that this app is missing. It would allow for the data to be shared amongst users better and for the app to be better for long term use.

**Benefits:**

* **Persistent Storage:** Data would be saved eliminating the need for re-entry or loss of data if the application crashes or is closed.
* **Efficient Querying:** Databases are optimized for queries, making searching and filtering large datasets much faster.
* **Scalability:** A database can handle larger amounts of data compared to simply storing data on memory.

**Compatibility:** The application is already built using .NET, so integrating SQL Server or SQLite would be straightforward using something like entity framework and linking the app to the SQL database.

**2. Improved Security**

**Recommendation:** Establishing better security for the application such as user verification and other features such as single-sign-on.

**Benefits:**

* **Data Security:** By only allowing users who have an account and have logged in it will help prevent unwanted users from accessing the application and viewing the data.
* **Prevent against attacks:** Having proper security in place will help protect the app and other users from unwanted attacks (such as DDOS attacks) which may cause the app to have to go down whilst the issue is resolved.

**Compatibility:** A simple login page that has hashed passwords and communicates with the server in the backend would provide at least some barrier for entry which can then be improved upon in the future by adding features such as SSO and OAuth.

**3. Cloud Deployment**

**Recommendation:** Deploying the application to the cloud using Microsoft Azure or another cloud platform would improve accessibility, scalability, and performance.

**Benefits:**

* **Scalability:** Cloud services offer the ability to scale resources up or down based on demand, ensuring consistent performance.
* **Availability:** Cloud platforms offer better stability and better data security, ensuring the application remains available more often.
* **Managed Services:** Azure provides managed database services, serverless computing, and other services that can also improve the application in the future.

**Compatibility:** Azure already integrates with .NET applications, providing options for hosting, databases, and serverless computing, which would suit the needs of this project.

**Lecturer Feedback Changes**

Based off the lecturer’s feedback from the last task I have made the following improvements to the application

1. Implemented a better navigation between pages by adding back buttons to allow the user to not have to close the app to open a different request.

2. Added a better recommendation algorithm for the recommendation feature in Task 2.

3. Ensured that I did my best to maintain my coding standards through my work and keep more consistent in how I code my application.