ST10029788

PROG7312

POE

Project Completion Report

# Project Overview

This municipal services application was designed to both streamline and enhance the efficiency of reporting, tracking, and management of municipal service requests, with additional features for event management and issue reporting. The core functionalities of the application include allowing users to report problems, track the status of service requests, and stay updated on local events. These features aim to streamline communication between residents and municipal actors and services, ensuring that users can easily report problems and stay updated on the resolution process.

To address these challenges, the application will offer a centralized, digital platform for residents to report municipal issues, receive updates, and stay informed about local events and announcements. By incorporating advanced data structures and algorithms, the platform will ensure efficient issue tracking, real-time updates, and a seamless user experience.

The challenges the application hoped to address are:

* Inefficient Service Request Management: Traditional systems struggle with organizing and tracking requests. The app uses advanced data structures like AVL trees and graphs to optimize performance, enabling faster and more efficient management.
* Lack of Real-Time Updates: Citizens often lack visibility into service statuses. The app provides real-time updates and notifications for service requests and local events.
* Managing Service Dependencies: Coordinating interdependent tasks can be complex. Graph algorithms like Prim’s Minimum Spanning Tree optimize task prioritization and workflow.

The project was developed in C# using the .NET Framework, ensuring compatibility with a variety of Windows environments.

Task 3 of the project specifically focused on integrating advanced data structures and algorithms to further optimize the application's performance. A significant component of the application included the addition of key data structures, which included basic trees, binary trees, binary search trees, AVL trees, red-black trees, heaps, graphs, and minimum spanning trees. Each data structure played a role in improving the efficiency and scalability of the service request tracking system, enabling faster data retrieval, better resource management, and more accurate service dependency management. The "Service Request Status" feature will be integrated into the main menu, allowing users to track the progress of their service requests, and users will view a well-organized list of their service requests, with details such as the status, assigned personnel, and progress updates. The system will leverage graph traversal algorithms and minimum spanning tree algorithms to efficiently display interdependent service requests.

# Key Learnings

The completion of this project provided insights into how algorithms are used in a real-world context, as well as how to code and document the process of making an application.

**Data Structure Selection**: Both AVL trees and red-black trees offer efficient balancing and fast lookups (Villalta, 2023). Through research into their differences, I was able to implement both structures to optimize different aspects of the application. AVL trees provided efficient searching by request ID, while red-black trees helped maintain service request order consistency.

* **Graph Theory**: The project reinforced concepts of relationship management in complex systems, especially when service requests are interdependent (GeeksforGeeks, 2024). Implementing graph traversal and minimum spanning tree algorithms deepened my understanding of how to manage and optimize dependencies within large systems (Duggal, 2021).
* **C# Windows Forms Integration**: Integrating advanced data structures with a frontend UI component, such as the ListView, taught me how to ensure smooth data presentation while maintaining backend performance. This required balancing technical concerns with user interface design.
* **Problem-Solving Skills**: The project required extensive research into recursive functions, tree balancing techniques, and optimal traversal paths. These problem-solving skills were important in debugging and optimizing both the tree and graph algorithms used throughout the project.

**UI/UX Design:** Balancing functional and aesthetic design while maintaining a smooth user experience is crucial, especially considering the context of a municipal app. By working with Windows Forms and designing user-friendly interfaces, I learned how to ensure that the application is intuitive, responsive, and accessible, leading to better user satisfaction.

**System Design and Architecture:** I learned how to design and structure a complex application that integrates multiple features like service request management, event tracking, and real-time updates. This helped me understand how to break down large problems into manageable components and ensure that different parts of the system work efficiently together.

**Performance Optimization:** Integrating advanced data structures and algorithms to optimize performance taught me how to assess and enhance the speed, scalability, and efficiency of an application. I learned how to choose the right algorithms for tasks, manage data effectively, and identify performance bottlenecks through testing.

**Version Control Systems:** The use of GitHub and Git was helpful in ensuring that I always maintain a working version of my code, and that data is not lost in the process. My ReadMe also served as a form of documentation, whilst allowing me to keep my own thoughts on the app and its’ functionalities organised.

# Challenges Faced in Task 3

One of the biggest challenges during Task 3 was the simultaneous use of both AVL trees and red-black trees to manage and organize service request data. Although these two tree structures both offer balanced searching capabilities, their specific balancing methods differ. This created a challenge in managing potential duplication and ensuring the application displayed data in a clean and non-redundant manner. The difficulty lay in using both structures for different functionalities (such as AVL trees for faster searching by request ID and red-black trees for maintaining order consistency) while ensuring that the user interface only displayed a single, unified view of the data.

There were also a lot of methods doing different things, with some of them having integrated work from Part 2, and managing all of these proved to take up more time than I thought, so that I avoid code duplication or creating methods with more functions than necessary.

I also struggled to managing the relationships between service requests using graphs and optimizing their traversal. Service requests can be dependent on one another, creating a complex web of relationships. I had to ensure that the dependencies were properly represented and displayed in the correct order. I implemented Prim’s Minimum Spanning Tree Algorithm and Breadth-First Search (BFS) to effectively manage and visualize these dependencies, ensuring minimal overhead in determining the most critical tasks that needed to be completed first.

For the overall application, my biggest challenge was the User Interface (UI) design. I found it difficult to strike a balance between the visual layout and the scope of information that needed to be displayed. With so many features, organizing and presenting all this data in a clear, user-friendly way without overwhelming the user was a big hurdle. I had to consider how to present information in an accessible manner while ensuring the design remained clean across different devices.

A particular struggle here was integrating these advanced data structures into the C# Windows Forms environment, as they posed compatibility challenges with UI controls, especially the ListView component. Ensuring smooth synchronization between the backend data structures and the frontend user interface required overcoming technical hurdles to achieve responsive and scalable functionality.

# How These Challenges Were Overcome

To tackle the duplication challenge with AVL and red-black trees, a design decision was made to consolidate the data into a single view for the user while retaining both tree structures in the backend. The AVL trees were used to enable efficient searching by request ID, while red-black trees ensured service request order consistency. However, only one set of results was presented to the user, minimizing any potential redundancy or confusion. This approach allowed both data structures to serve their specific purposes without affecting the user experience.

One of the key strategies to manage complex methods and avoid duplication is refactoring. I looked for opportunities to extract common code into reusable helper functions or classes. For example, if certain parts of logic were repeated across different parts of the application, I tried to encapsulate it into a utility class or a method that can be reused wherever needed.

I also used modularization. I split large methods into smaller, single-purpose functions. Each function should ideally perform one task, and this assisted in making the code easier to maintain and understand. This also made debugging and testing more straightforward for me.

The management of service request dependencies was approached by carefully implementing the graph structure using adjacency lists and node management. Prim’s Minimum Spanning Tree Algorithm helped optimize the calculation of the most critical dependencies with minimal computational overhead, ensuring that tasks were completed in an optimal order. The Breadth-First Search algorithm helped ensure efficient traversal across service requests and their interdependencies.

In terms of integration, thorough testing and debugging were critical. Extensive testing of both the tree structures and the graph traversal algorithms helped to ensure that the service request display system was functioning accurately and efficiently. The user interface was optimized to display data seamlessly while maintaining high performance, and compatibility issues between backend data structures and UI controls were resolved.

For the user interface challenge, I would identify the most important and frequently used features and prioritize them in the UI. This means focusing on the key functionalities that users will need most, like checking service request statuses or reporting issues. I would also group similar information together and avoid cluttering the interface with too many details at once. This would make the application feel less overwhelming and allow users to easily navigate to the sections they need. To avoid overwhelming users with information, I would implement a progressive disclosure strategy. This means showing the most relevant information first and allowing users to expand or drill down for more details when needed. I also ensured the UI adapts smoothly to different screen sizes by using responsive design principles.

# Programming Techniques

* **Event-Driven Programming**:
  + Event-driven programming was essential for handling user interactions in C# Windows Forms (BillWagner, 2024). Event handlers were used to respond to actions such as submitting service requests, searching for events, and viewing service statuses.
* **Algorithm Design and optimization**:
  + Prim’s Minimum Spanning Tree (MST): This algorithm is used to optimize the task prioritization process (Sayeda Haifa Perveez, 2021). By ensuring that the most important or dependent tasks are processed first, it minimizes computational overhead and ensures that interdependent tasks are completed efficiently.
  + Breadth-First Search (BFS): A graph traversal technique to explore the nodes (service requests) in a level-by-level manner (Linkedin.com, 2024). It is particularly useful for ensuring that all related service requests are handled in an optimal order and that the system doesn't miss any critical dependencies between tasks.
  + Recursive Functions: The project involved recursion, especially in tree-related algorithms like balancing AVL trees or traversing complex graph structures. Recursive techniques were employed for tree operations like insertion, deletion, and traversal. Recursion is essential in algorithms like tree balancing and graph traversal.
* **Modularization and Code Reusability**
  + Refactoring: The use of refactoring techniques to simplify and improve the structure of existing code. Breaking down large methods into smaller, single-purpose functions is an example of applying the Single Responsibility Principle (SRP), which improves maintainability and reusability (Gillis, 2021).
  + Helper Functions/Classes: Modularizing the code by extracting common logic into reusable methods or utility classes is another important technique used to avoid code duplication and improve maintainability (keySkillset, 2023). This is particularly useful when dealing with complex data structures and algorithms.
  + Separation of Concerns: The code is split into logical components (e.g., data structures, algorithms, UI), making it easier to debug and test each part independently. This is a key principle in system design to improve maintainability and scalability (GeeksforGeeks, 2023).
* **User Interface (UI) and UX Design**
  + Progressive Disclosure: A design pattern used to improve the user experience by gradually showing more information as needed (The Decision Lab, 2024). Initially, the user is presented with the most essential information, with the option to access more details later. This helps reduce information overload and keeps the interface clean and intuitive.
* **Software Engineering Practices**
  + Version Control with Git: The use of Git and GitHub for version control is crucial for managing the development process, especially in large projects. Git helps track changes and ensure code integrity across different stages of development (GeeksforGeeks, 2024).
  + Documentation: Writing clear documentation in the form of a ReadMe file and commenting code properly are vital for maintaining code clarity and for facilitating future development or handoffs to other developers.
* **System Design and Architecture**
  + Scalability: By choosing efficient data structures (AVL trees, red-black trees, graphs), the system is built to handle large amounts of data without performance degradation (Mushiba, 2024). This is critical for municipal applications that may need to handle a high volume of service requests and user interactions.

# Reference list

BillWagner 2024. *Events - C#*. [online] Microsoft.com. Available at: [https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/events/](https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/events/%20) [Accessed 13 Nov. 2024].

Duggal, N. 2021. *All You Must Know About Minimum Spanning Tree in Data Structures*. [online] Simplilearn.com. Available at: [https://www.simplilearn.com/tutorials/data-structure-tutorial/minimum-spanning-tree-algorithm-in-data-structure](https://www.simplilearn.com/tutorials/data-structure-tutorial/minimum-spanning-tree-algorithm-in-data-structure%20) [Accessed 12 Nov. 2024].

GeeksforGeeks 2023. *Data Structures and Algorithms for System Design*. [online] GeeksforGeeks. Available at: <https://www.geeksforgeeks.org/data-structures-and-algorithms-for-system-design/> [Accessed 13 Nov. 2024].

GeeksforGeeks 2024a. *Git vs. Other Version Control Systems: Why Git Stands Out?* [online] GeeksforGeeks. Available at: [https://www.geeksforgeeks.org/git-vs-other-version-control-systems-why-git-stands-out/](https://www.geeksforgeeks.org/git-vs-other-version-control-systems-why-git-stands-out/%20) [Accessed 12 Nov. 2024].

GeeksforGeeks 2024b. *Graph Algorithms*. [online] GeeksforGeeks. Available at: [https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/](https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/%20) [Accessed 13 Nov. 2024].

Gillis, A.S. 2021. *refactoring*. [online] Search App Architecture. Available at: [https://www.techtarget.com/searchapparchitecture/definition/refactoring](https://www.techtarget.com/searchapparchitecture/definition/refactoring%20) [Accessed 12 Nov. 2024].

keySkillset 2023. *In the realm of programming, the quest for code elegance and efficiency drives us to vanquish a relentless foe: redundant code. As warriors of clean and maintainable software, we embark on a journey to slay this duplicate dragon, armed with the knowledge of Python’s powerful techniques.* [online] Linkedin.com. Available at[: https://www.linkedin.com/pulse/redundant-code-avoid-duplicating-instead-encapsulate-reusable](:%20https:/www.linkedin.com/pulse/redundant-code-avoid-duplicating-instead-encapsulate-reusable%20) [Accessed 13 Nov. 2024].

Linkedin.com. 2024. *What are the most effective algorithms for graph traversal?* [online] Available at: [https://www.linkedin.com/advice/0/what-most-effective-algorithms-graph-traversal-4b9dc [](https://www.linkedin.com/advice/0/what-most-effective-algorithms-graph-traversal-4b9dc%20%5b)Accessed 13 Nov. 2024].

Mushiba, A.N. 2024. *Title: Red-Black Trees: An Essential Tool for Efficient Data Structures and Algorithms*. [online] ResearchGate. Available at: [https://www.researchgate.net/publication/377471721\_Title\_Red-Black\_Trees\_An\_Essential\_Tool\_for\_Efficient\_Data\_Structures\_and\_Algorithms](https://www.researchgate.net/publication/377471721_Title_Red-Black_Trees_An_Essential_Tool_for_Efficient_Data_Structures_and_Algorithms%20) [Accessed 13 Nov. 2024].

Sayeda Haifa Perveez 2021. *Prim’s Minimum Spanning Tree (MST): All You Need to Know*. [online] Simplilearn.com. Available at: [https://www.simplilearn.com/tutorials/data-structure-tutorial/prims-algorithm](https://www.simplilearn.com/tutorials/data-structure-tutorial/prims-algorithm%20) [Accessed 13 Nov. 2024].

The Decision Lab. 2024. *Progressive Disclosure - The Decision Lab*. [online] Available at: [https://thedecisionlab.com/reference-guide/design/progressive-disclosure](https://thedecisionlab.com/reference-guide/design/progressive-disclosure%20) [Accessed 13 Nov. 2024].

Villalta, H. 2023. *Red Black | AVL Tree | Tree Algorithms | Medium*. [online] Medium. Available at: [https://medium.com/@humberto521336/red-black-avl-trees-overview-d2f1af7886ee [](https://medium.com/@humberto521336/red-black-avl-trees-overview-d2f1af7886ee%20%5b)Accessed 12 Nov. 2024].