**CitizenHelper Report**

ST10092141 – Kgothatso Theko

**1. Report Status Feature with Data Structures:**

**1.1. Role of AVL Tree in Report Management**

The **AVL Tree** ensures that the service requests are stored in a balanced manner, allowing for efficient insertion, deletion, and search operations with a time complexity of O(log n). This is crucial for applications that handle a large number of service requests, ensuring quick access and manipulation of data.

**Example:** When a user searches for a report by its ReportID, the AVL Tree allows the application to locate the report swiftly without traversing the entire dataset.

**1.2. Role of Graph in Managing Report Dependencies**

The **Graph** structure enables the representation of complex relationships between different reports. For instance, a service request may depend on the completion of another request.

**Example:**

* **Report A**: "Fix leaking roof"
* **Report B**: "Replace damaged ceiling tiles"

Here, **Report B** depends on **Report A**. The graph structure allows the application to manage such dependencies, ensuring that related tasks are addressed in the correct order.

**1.3. Role of Heap in Optimizing Display of Reports**

The **Heap** structure, specifically a priority heap, can prioritize reports based on certain criteria such as urgency or submission time. This ensures that high-priority reports are addressed first and displayed prominently.

**Example:** A report marked as "Emergency" can be given higher priority in the heap, ensuring it appears at the top of the DataGrid in the ReportStatus window.

**2. Implemented Data Structures:**

**2.1. Overview of Implemented Data Structures**

* **AVL Tree**: Used for efficient storage and retrieval of reports based on ReportID. It ensures balanced tree operations, providing O(log n) time complexity for insertions and searches.
* **Graph**: Manages complex relationships between reports, such as dependencies. It facilitates the representation of interconnected service requests.
* **Heap**: Prioritizes reports based on defined criteria (e.g., urgency, submission time) to optimize the display order in the UI.

**2.2. Contribution to "Report Status" Feature**

* **Efficiency**: AVL Trees provide quick search capabilities, essential for users tracking reports by their unique identifiers.
* **Relationship Management**: Graphs allow the application to handle interdependent reports, ensuring that related service requests are managed cohesively.
* **Optimized Display**: Heaps ensure that the most critical reports are highlighted, improving user experience by focusing attention on high-priority issues.

**2.3. Examples**

* **Searching for a Report**: When a user enters a ReportID in the ReportStatus window's search bar, the AVL Tree quickly retrieves the corresponding report.
* **Managing Dependencies**: If **Report B** depends on **Report A**, the graph structure ensures that **Report B** is only addressed after **Report A** is completed.
* **Displaying High-Priority Reports**: Emergency reports are inserted into the heap with higher priority, ensuring they appear at the top of the DataGrid.

**3. Project Completion:**

To enhance the functionality, performance, and maintainability of your WPF application, consider integrating the following technologies and tools:

**Project Overview**

CitizenHelper is a Windows Presentation Foundation (WPF) C# application designed to provide South African citizens to access and request municipal services, get information about local events and announcements, as well as get the status information on their request. The application offers features such as report issues, request status, event search based on categories and dates, personalized recommendations, binary tree search feature and multilingual support to enhance user experience.

Task 1 was the implementing of the report issue functionality and multilingual support feature. The implementing of the report became a challenge when it was a matter of how the data will be saved and retrieved in order to display it as a list view. I did not what to use a database as it required me to use API. So, to overcome challenge, I researched how else can I use WDF to locally save and retrieve data and in my search, I found a way to save it as JSON data in its in-built user settings. This was good as my experience with using JSON data came in handy as getting it and adding it did not even require me to make API’s. Making the multilingual support was not a challenge as hard-coded data had its own .xml file which can store all strings used it the application and make a function to change between the xml file language strings.

Task 2 became a challenge as I need to know and understand different data structures and algorithms in order to implement the event-based search and personalised recommendations. To overcome this challenge, I first broke down my steps, decide on the data to be used which was Sorted Dictionary for Events Organization and Hashset for unique categorization. Once I broke down my steps, I started on making the screens for the events, implementing services for the data structures and added a view model to manage the data as it was all hard coded. Most of this followed the MVVM design pattern as doing my research this data pattern was need in order to implement this categorised and recommend event-based search feature.

Task 3 did not prove to be a challenge as following the MVVM design pattern and using JSON data to save my reports made it easy to adapt and edit my code in a way the getting the report status was just a matter of making a unique identifier in my models and adding related services, screens and algorithms. Most challenges required me to do research on how to follow a Binary search tree and heap algorithm before actually editing my reports to use these algorithms.

The overall project was done in first breaking down the steps need to implement a certain feature and then doing research to first understand the concept before actually implementing in code.

**Key learnings**

The overall project was done in first breaking down the steps need to implement a certain feature and then doing research to first understand the concept before actually implementing in code. The skills that I learnt in this project was adaptability, creative thinking and resourcefulness. Planning ahead and understanding the project requirement help plan how data will be saved, retrieved and manipulated within the application. Leveraging on the skills on JSON data also made implementing all functionalities easier whether if I’m adding or getting the data. AI tools were strictly used to correct my implemented code only. As they did not provide any learning outcomes when used.

**4. Technology Recommendations:**

To enhance the functionality, performance, and maintainability of your WPF application, consider integrating the following technologies and tools:

* **Database Integration with SQLite**

**Recommendation:** Use **SQLite** as a lightweight, file-based database to store reports instead of relying on UserSettings.

**Justification:**

* **Scalability**: Handles larger datasets more efficiently.
* **Querying**: Facilitates complex queries, filtering, and sorting.
* **Data Integrity**: Ensures better data management and reduces risks associated with data loss or corruption.

* **MVVM (Model-View-ViewModel) Pattern**

**Recommendation:** Implement the **MVVM** design pattern to separate concerns, enhance testability, and improve maintainability.

**Justification:**

* **Separation of Concerns**: Distinguishes between UI logic and business logic.
* **Reusability**: Facilitates the reuse of components.
* **Testability**: Simplifies unit testing of business logic.