Max Day
ST10322238

The IIE's Varsity College
Waterstone Drive
Sandton, 2196
12/11/2025

PROG7312

MUNICIPAL SERVICES APPLICATION

TECHNICAL DOCUMENT

Contents

Preamble 3
Project Overview 3
Implementation Report 3
How to Compile and Run 3
Technology Stack: 3
Installation and Setup 4
System Requirements: 4
How to Use the Program 5
Part 1: Report Issues 5
Part 2: Local Events and Announcements 6
Part 3: Service Request Status 7
Data Structures Implementation 9
Overview 9
Changelog 9
Improvements made from Part 1: 9
Improvements made from Part 2: 9
Issues fixed based on feedback: 10
New features added: 10
Development Timeline 10
Part 1: Report Issues (Weeks 1-3) 10
Part 2: Local Events (Weeks 4-6) 11
Part 3: Service Request Status (Weeks 7-10) 11
Strengths 12
What went well in the project 12
Successfully implemented features 13
Effective design decisions 13
Deployment Recommendations and Futureproofing 13
Updates Based on Feedback 14
Feedback Received: 14
Incorporation of Feedback 14
AI Declaration: 14
References/Bibliography 15

Preamble

Part 1: 100%
Part 2: 50%
Part 3: TBD

This project has a fully written outline of every single data type and data structure in the HTML file (DataStructures_Documentation.html). I urge you please to read it because I have implemented every single type of data structure in this project including extra ones from my own technical knowledge. My part two submission has sections which were not marked. The html document outlines all the in-depth technical explanations as well as where to find each data structure and algorithm in the code.

It is also worth noting that a GitHub Classroom was given only in the late stages of this project, so all commits and development histories are in MunicipalApp while the submission URL is in prog7312-poe-ML0Day0VC.

I'm so sorry for the inconvenience, but I really appreciate it.

Project Overview

The municipal app is a Windows Forms Desktop application built on the .NET 4.7.2 framework. The backend is written in C# and WPF layout forms are used for the frontend. The app is a single compiled app with minimal external DLL files. The application aims to create a reliable and sustainable application that encourages south African citizens to report issues in their municipalities. It allows users to report issues, access information about local events and as well as track the status of their and many others service requests

Implementation Report

How to Compile and Run

Technology Stack:

- **Framework:** .NET Framework 4.7.2
- **UI:** Windows Forms
- **Database:** SQLite (embedded)
- **Language:** C# 8.0
- **Development Environment:** Visual Studio 2017 Developer Builder
- **IDE:** Zed

Installation and Setup

System Requirements:

Recommended:

- Windows 10/11
- .NET Framework 4.8

Method 1: Running Pre-Built Application

1. Navigate to the release directory:

   ```
   cd bin\Debug
   ```

2. Run the executable:

   ```
   .\municiple app.exe
   ```

3. Database is created and initialized on launch

Method 2: Building from Source

Prerequisites:

- Visual Studio 2017 or later
- .NET Framework 4.7.2

Build Steps:

1. Clone the repository to local:

   ```
   git clone https://github.com/Max0xDay/MunicipalApp.git
   cd MunicipalApp
   ```

2. Open in Visual Studio or preferred editor:
   - Open municiple app.sln
3. Build the solution:
   - Press Ctrl+Shift+B
   - Or use the provided build.bat script (prefered)
4. Run the application:
   - Press F5 to run with debugging

- Or run using Method 1
-

Dependencies:

- Microsoft. (2023). .NET Framework
- SQLite Core version 1.0.118.0
- SQLite Core NetFramework version 1.0.118.0

How to Use the Program

Run the applications compiled exe file using the method stated in 1.1. You will then be presented with the main landing page UI.

Part 1: Report Issues

Purpose: Citizens can report municipal issues with location, category, and media attachments and the data will be stored in a local database.

Steps:

1. Click "Report Issues" from the main menu
2. Fill in the following required fields:
   - Title: Enter the title for the issue (Auto suggestion predictive text trie function)
   - Location: Enter the address or location of the issue
   - Category: Select the right option from the dropdown (Sanitation, Roads, Utilities, Water, Electricity)
   - Description: Provide detailed description in the text box (multi language spellcheck is available)
   - Media Attachment (Optional): Click "Attach Media" to add or remove images or documents
3. Monitor Progress:
   - Progress bar at bottom shows form completion percentage
   - Encourages users to finish with hints and messages
4. Submit:
   - Click "Submit Report"
   - Confirmation message displays with unique reference ID
   - Issue stored in SQLite database
   - Ability to share the submitted report to social media platforms. Will help boost user engagement
5. View Submitted Issues:

- Click “View Reports” button to access to all reports
- View all reported issues in a table
- Click “Refresh” to update with the latest data

Features:

- Progress bar user engagement strategy
- Social media placeholder buttons (WhatsApp, Email, SMS)
- Input validation - File attachment support (JPG, PNG, PDF, DOCX)
- Persistent Database
- Predictive text using trie algorithm
- Multi Language spellcheck for users

Part 2: Local Events and Announcements

Purpose: Browse and search municipal events with intelligent recommendations.

Steps:

1. Click “Local Events and Announcements” from main menu
2. Browse Events:
   - View all upcoming events in chronological order
   - Events organized using Sorted Dictionary for efficient retrieval
3. Search for Events:
   - Enter search terms in search box
   - Filter by category using dropdown
   - Select date range for temporal filtering
4. View Recommendations:
   - Based on previous search and user interactions with options, predict upcoming events for users
   - Related events appear higher in the order for users
5. Event Details:
   - Click any event to see full information
   - Includes date, time, location, category, description

Features:

- Sorted Dictionary for O(log n) event organization
- Priority Queue for upcoming events - Hash Set for unique category tracking
- Recommendation engine based on user patterns
- Multi-criteria search (category + date)

Custom Data Structures Used:

- Sorted Dictionary: Event organization by date
- Priority Queue: Prioritize upcoming events
- Hash Set: Unique categories and dates
- Dictionary: Fast event retrieval
- Queue: Manage search history and user interaction for recommendations

Part 3: Service Request Status

Purpose: Track service requests with advanced data structures visualization for multiple data structures.

Steps:

1. Click “Service Request Status” from main menu
2. View All Requests:
   - Complete list displayed in main data grid
   - Shows ID, category, status, priority, submission date
3. Search by Unique ID:
   - Enter request ID in search box
   - Click “Search”
   - Binary Search Tree for O(log n) searching
4. Filter Requests:
   - By Status: Pending, In Progress, Completed, Cancelled
   - By Category: Sanitation, Roads, Utilities, Water, Electricity
   - By Priority: High, Medium, Low
   - Combine multiple filters
5. Explore Data Structures:

- Select different tree views from dropdown:
  - Binary Search Tree (BST)
  - AVL Tree (self-balancing)
  - Red-Black Tree
  - Basic Hierarchy Tree
  - Binary Tree
- Observe how each structure organizes the same data
- Compare performance characteristics

6. View Related Requests:
   - Graph analysis reveals connected issues
   - Shows requests sharing categories or locations
   - Dependency relationships displayed
7. Graph Traversal:
   - Select traversal algorithm:
     - Breadth-First Search (BFS)
     - Depth-First Search (DFS)
   - Choose starting request
   - Visualize traversal order in results panel
8. Priority Queue:
   - View of the highest-priority requests automatically surfaced
   - Min-Heap structure ensures O(1) access to most urgent
9. Minimum Spanning Tree:
   - View minimum spanning tree calculation results
   - Identifies efficient resolution pathways
   - Shows optimal request groupings
10. Update Request Status:
    - Select request from grid
    - Choose new status from dropdown
    - Click “Update Status” to save
11. Statistical Analysis:
    - View distribution by status, category, priority
    - Tree depth and balance factors
    - Performance metrics comparison

Features:

- Multiple tree structure implementations
- Graph with weighted edges - BFS/DFS traversal algorithms

- Minimum Spanning Tree (also applies Kruskal’s algorithm)
- Min-Heap priority queue - Real-time status updates
- Related request discovery
- Use of Tree statistics

Data Structures Implementation

Overview

This application demonstrates multiple data structures and algorithms throughout the app. The role, implementation details, and contribution to efficiency for each one have been outlined visually in the DataStructures_Documentation.html file, which is inside the repo and attached to this marking submission. Please refer to this document, as it lays it out better than any word document could. It also includes the section on how each algorithm and data structure benefits the app in its own way.

Changelog

Improvements made from Part 1:

For Part 1, I achieved full marks, but I wasn't satisfied with it. From commits 4602154 to 20f0263, I not only enhanced the UI by completely overhauling it for a sleek, modern grayscale look, but I also added predictive title suggestions through a trie algorithm which auto-suggests titles. For example, a user can start typing the word "broken" and multiple options to auto complete the sentence will appear. To accept the autofill, the user will hit the spacebar to accept it; otherwise, they can continue typing. Secondly, in the description box, I added a spell-check algorithm that supports 7 South African languages.

Improvements made from Part 2:

For Part 2, I achieved 50%. This was due to the lack of communication stating that custom data types for all must be used. I had 3/5 custom types. With the overhaul to the UI that I did for the whole app, I added the last remaining 2 data types and also fixed a memory issue where the database queries wouldn't flush, meaning over time queries were taking up 5-6 MB when they should have been taking only a few kilobytes. This was easily fixed with a SQLite drop function which drops the query once it's completed and runs garbage collection on the function.

Issues fixed based on feedback:

My app had no issues on feedback, only features missing, and some of my features weren't marked. Overall, all features have been accounted for and there are no issues in the app.

New features added:

The following features were added: Global styling for the app to reduce load and make code look more presentable. Added predictive title suggestions based on the trie algorithm to simplify user reporting of issues. Added spell-checking to text boxes for easier writing of issues. Implemented suggestions for events not only based on the previous search results but based on how much a user interacts with specific items in a database table using the onClick function. Added database table isolation so not all tables are loaded into the app on startup and only the tables for each section are included in the system.

Development Timeline

Part 1: Report Issues (Weeks 1-3)

Week	Milestone	Status
1	Boilerplate implementation: Project setup, database design, form creation	✓ Complete
2	Core Functionality: Issue reporting functionality, media attachment, input validation	✓ Complete
3	Writeup and polishing of app before submission	✓ Complete

Challenges Faced and Overcome:

- I started developing this system on Linux and had forms issues. Swapped to Windows halfway through
- Database using much more memory than it should during initialization

Part 2: Local Events (Weeks 4-6)

Week	Milestone	Status
4	Boilerplate implementation: Events database schema, basic display	✓ Complete
5	Core Functionality: Search functionality, custom sorted dictionary implementation	✓ Complete
6	Addition of onClick user recommendation algo to predict better results.	✓ Complete

Challenges Faced and Overcome:

- Complications in life meant my week 6 was scattered and not well planned out
- User stories provided were extremely ambiguous assuming that we need to have custom data type
- Having to deal with C# extremely overly engineered built in sorting for tables messing up with the recommended options on the table.

Part 3: Service Request Status (Weeks 7-10)

Week	Milestone	Status
7	Boilerplate implementation: Tree structures implementations	✓ Complete
8	Core functionality: Heap and priority queue implementation	✓ Complete
9	Graph structure, traversal algorithms with additional and MST	✓ Complete
10	UI upgrade Aswell as written documentation.	✓ Complete

Challenges Faced and Overcome:

- C# has a way of making it so that when you reference a form using direct notation it forgets about everything polymorphic, and self inserts itself as a non-object. This means if you are raw typing this code and you type “reportForm.” And you press alt + space to see suggestions or continue typing to type “reportForm.ShowDialog()” for some reason it doesn’t recognise that reportForm is the name of the form and is an object and instead holds the cursor hostage breaking my whole flow and forcing me to use the trackpad

- Windows forms doesn’t like when you don’t initialise text boxes nor does it throw an error when you don’t. This is fixed in latest .NET version, but it threw me off a lot of the times.
- Graph edge weight calculation was wrong, and I forgot to check it until much later.

Overall Project Statistics

- **Total Development Time**: 6 weeks
- **Lines of Code**: ~13,500
- **Classes Created**: 47
- **Data Structures Implemented**: 17 custom structures
- **Forms Designed**: 3 major forms
- **Database Tables**: 3 tables with relationships

Strengths

What went well in the project

This project went well in the sense that what I planned could be executed. Windows Forms is often talked down upon. It's to this day the simplest and best way to create Windows applications. Not only does it have a massive ecosystem, but it's very widely supported. This means that directly typing a UI rather than using drag-and-drop GUI is very feasible and was easier.

For most of the pages, I didn't use any drag-and-drop tools as my Linux machine could not run it, but it turned into a challenge and one that was easily overcome. With a year and a half of experience with Avalonia UI, I warmed up to it quickly.

Another thing that went well in the project was not having to worry about the UI as much. For Part 1 and Part 2, I created basic UIs with the focus on functionality. For Part 3, I have completely overhauled the UI and refined the code a lot. It was bothering me how abstracted I initially wrote my code, and it was also filled with tons of console logging when I was debugging it.

Successfully implemented features:

This section has been answered many times above, so to save both you and I some time, I'm going to list the extra features I've implemented. All features that were required have been implemented.

- Spell-check system with 7 supported African languages
- Lightweight database structure with memory management
- Auto-prediction system for title of issues (trie algorithm)
- Recommended feature algorithm based on user interaction with table and not only with search history to better predict which events the user likes
- All advanced data structures implemented. All are visible on the Part 3 implementation
- Custom data structures and types
- Custom data generation to populate databases for testing and visualizing the different data structures

Effective design decisions

- Consistent UI design patterns and structures that are user-friendly
- Consistent and uniform file structure and project structure
- Easy to maintain, develop, and scalable
- Very defensive and robust programming (0 errors thrown, only try-catches)
- Persistent data storage
- Camel case naming with prettier code formatting

Deployment Recommendations and Futureproofing

For the future, if this app had to be implemented properly, it firstly needs to shift from SQLite to PostgreSQL for the database. It is not great to run everything locally. Secondly, there's no input validation for SQL injection, and .NET's framework is out of date and doesn't have the latest regex filtering. Secondly, this app requires a Windows computer to run. Most South Africans have a phone at most. A PWA web app would have been a much better choice for this, as anyone on any device, regardless of operating system or type, could use it. Finally, it needs to be simplified and use non-custom data structures. While they demonstrate understanding and were

included in the user stories for this task, in the real world it can be seen as a maintainability issue and adds unnecessary complexity to the app. Built-in functions that work the exact same way exist and are proven and tested to be more secure, run better, and are more reliable.

Updates Based on Feedback

Feedback Received:

- No feedback was given for Part 1
- For part 2 my lecturer gave one line of feedback saying: "no custom ds implemented". DS we later found out to be data structures, and they have all been fully implemented.

Incorporation of Feedback

- Inside the DataStructures_Documentation.html, I have outlined better about what code changed, but the main issue was that custom dictionaries and custom hashes were not directly implemented in the code. These were easy fixes and have been implemented.

AI Declaration:

There has been no direct use of AI. Grammarly was used to proofread spelling and grammar in the documents and for the programming Zed editor as a built-in auto complete with suggested IntelliSense which uses data trained of my code only. This was used minimally and was more of an annoyance than a help and was eventually removed from the editor at a later stage. No Vibe coding or large generative AI use was used in the development of any of the documentation or Programming

For more info please refer to https://zed.dev/ai

References/Bibliography

No direct sources were used in Part 3 write-up. Part 1 has all the sources listed; however, some resources I used to either fix bugs or learn about implementing features in the app will be listed below in IEEE format:

[1] Anonymous, "Strategy to avoid running out of memory in memory intensive application," Software Engineering Stack Exchange. [Online]. Available: https://softwareengineering.stackexchange.com/questions/295063/strategy-to-avoid-running-out-of-memory-in-memory-intensive-application. [Accessed: Sept. 17, 2025].

[2] PC SOFT, "WinDev Documentation," doc.windev.com. [Online]. Available: https://doc.windev.com/?1410090992. [Accessed: Oct. 3, 2025].

[3] Anonymous, "Winforms C# thing on Linux," Reddit - r/linux4noobs. [Online]. Available: https://www.reddit.com/r/linux4noobs/comments/17tijne/winforms_c_thing_on_linux/. [Accessed: Aug. 29, 2025].

[4] Anonymous, "Stack overflow exception that led to a hard crash," Visual Studio Developer Community. [Online]. Available: https://developercommunity.visualstudio.com/t/stack-overflow-exception-that-led-to-a-hard-crash/1093662. [Accessed: Nov. 5, 2025].

[5] Microsoft, "GraphNode Class," Microsoft Learn. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/api/microsoft.visualstudio.graphmodel.graphnode?view=visualstudiosdk-2022. [Accessed: Sept. 22, 2025].

[6] Anonymous, "GitHub Community Discussions," GitHub Community Forum. [Online]. Available: https://github.com/orgs/community/discussions/147437. [Accessed: Oct. 18, 2025].

[7] Anonymous, "Has anyone tried to code a C# project with Zed," Reddit - r/ZedEditor. [Online]. Available: https://www.reddit.com/r/ZedEditor/comments/1mlvyr6/has_anyone_tried_to_code_a_c_project_with_zed_i/. [Accessed: Aug. 26, 2025].

[8] Anonymous, "Difference between tries and trees," Stack Overflow. [Online]. Available: https://stackoverflow.com/questions/4737904/difference-between-tries-and-trees. [Accessed: Oct. 31, 2025].