Projecte Completion Report

ST10465727 AAPD7112

OBAKENG PITSE

Project Completion Report & Technology Recommendations

Project Completion Report

This report summarises the completion of the Municipal Services Application, with a focus on Task 3, which required the integration of advanced data structures and algorithms to enhance the Service Request Status module. This included the implementation of Binary Trees, BST, AVL Trees, Heaps, Graph Traversal, and Minimum Spanning Tree (MST) logic.

Challenges Faced During Task 3

Task 3 brought significant complexity due to the number of data structures involved and the requirement to demonstrate their impact on the Service Request Status functionality. Several challenges were encountered:

1. Coordinating Multiple Data Structures Simultaneously

A major challenge was organising and rebuilding multiple structures (BST, AVL, MinHeap, Graph) every time the user refreshed the page.
Solution:
A unified method, BuildDataStructures(), was created to rebuild all data structures from the repository efficiently and consistently.

2. Graph Connectivity and MST Failures

Initially, issues were not always connected in a meaningful way. For example, if two issues did not share a location or category, no edge was created—making the graph disconnected. When disconnected, MST could not be computed, resulting in user confusion.
Solution:
The edge creation logic was strengthened by:

- Connecting issues with similar categories/locations using low-weight edges
- Adding fallback edges to ensure full graph connectivity
- Displaying meaningful messages when MST could not be generated

3. Difficulty Implementing Prim's Algorithm (Tuple Naming Errors)

C# tuple items default to Item1, Item2, Item3 unless explicitly named. Using named tuples (.w, .from, .to) with unnamed tuple comparers resulted in compilation errors.
Solution:
Explicit tuple types were correctly defined:

Comparer<(double w, int from, int to)>.Create(...)

This resolved the tuple field access errors.

4. Ensuring Performance and UI Responsiveness

Constructing a graph with pairwise comparisons ($O(n^2)$) and inserting many issues into AVL trees could cause small performance lags.
Solution:

- Structures were rebuilt only when requested manually
- Efficient O(log n) insertion methods were used
- DataGridViews were loaded after all structures were computed

5. XML Deserialization Problems After Model Updates

Updating the Issue model with new fields (Priority, Id, Status) caused deserialization failures when the old XML file contained incomplete fields.
Solution:

- Regenerated issues.xml
- Added default values to prevent null deserialization issues
- Improved persistence handling

6. Maintaining Clear Code Structure

With many data structures and UI interactions, the code initially became cluttered.
Solution:

- Separation of concerns was applied
- Utility classes were moved into dedicated folders
- Meaningful method groupings and comments were introduced

Key Learnings

1. Mastery of Advanced Data Structures

Implementing AVL trees, MinHeap priority queues, and graph structures allowed for a deeper understanding of how these structures operate in real-world systems.

2. Applied Graph Algorithms

Graph traversal and MST algorithms were often theoretical, but this project showed their practical use in real municipal services—such as identifying clusters of related issues and optimising routes.

3. Improved Debugging and Problem-Solving

The project required:

- diagnosing tuple name mismatches
- understanding XML persistence issues
- carefully structuring tree balancing algorithms
  This improved overall debugging proficiency.

4.4 Enhanced C# Windows Forms Development

Skills gained include:

- advanced use of DataGridView
- multi-form navigation
- responsive layouts using TableLayoutPanel
- handling user input safely

4.5 Better Software Architecture Understanding

Designing with scalability in mind helped reinforce:

- clean code principles
- module separation
- reusability
- future extensibility

4.6 Exposure to Realistic Municipal Service Use Cases

Understanding how municipalities handle service tickets highlighted why performance and data analysis matter. Large volumes of service requests demand fast sorting, clustering, and prioritization - perfect use cases for the data structures implemented.

Technology Recommendations

1. SQL Database Integration (SQL Server / PostgreSQL / MySQL)

Benefit:

- Replaces XML persistence with reliable, scalable relational storage
- Supports indexing and efficient querying for large datasets
- Enables multi-user access and future API development

Compatibility:

Easily integrates with .NET through Entity Framework Core.

2. Azure Cloud Services

Recommended Services:

- **Azure SQL Database** → scalable cloud database
- **Azure Blob Storage** → efficient storage for user-uploaded attachments
- **Azure Service Bus or Azure Queue Storage** → asynchronous handling of background tasks

Benefits:

Enhances reliability, disaster recovery, and remote accessibility.

3. ASP.NET Core or Blazor Migration

Justification:

WinForms is desktop-only. Migrating would:

- enable browser access
- allow mobile users to report issues
- support municipal-wide cloud deployment

Compatibility:

Business logic remains reusable through service classes.

4. GIS / Mapping Tools (ArcGIS, Leaflet, Google Maps API)

Benefit:

Visual mapping of issue locations adds enormous analytical value:

- heatmaps of problem areas
- cluster detection
- optimised inspection routes

Compatibility:

Can be integrated via REST APIs or embedded map controls.

5. Logging & Telemetry (Serilog, Application Insights)

Benefits:

- Tracks usage patterns
- Logs errors with context
- Improves maintainability and debugging

6. Dependency Injection and SOLID Architecture

Benefits:

- Cleaner and more modular codebase
- Easier to test and maintain
- Better aligned with modern .NET practices

5. Conclusion

The Municipal Services Application is now a fully functional system capable of receiving, storing, analysing, and displaying service requests using advanced data structures. Task 3 significantly strengthened the analytical capabilities of the system and demonstrated the practical use of AVL Trees, Heaps, Graphs, BFS/DFS, and MST.

Despite several technical challenges, each was successfully resolved through systematic debugging, architectural improvements, and careful handling of algorithms and data models. The project fostered substantial growth in algorithmic understanding, software design, data structure integration, and practical C# development.

The recommended future technologies provide a clear roadmap toward transforming this academic prototype into a production-ready municipal platform capable of city-wide deployment.