# POE

PROG7312

18 NOVEMBER 2024

LEONARD LOUW BESTER

ST10026396

# CONTENTS

# INTRODUCTION

# DATA STRUCTURE IMPLEMENTATIONS

## TREE DATASTRCUTURE

### TREE TYPE

A **binary tree** has been implemented as means to organise all the records from the local **SQLite** Database.

### CORE IDEAS

The tree is represented with **ServiceRequestNode** objects, each of these objects contains the following data fields:

- Id
- Location
- Category
- Description
- Image
- Status
- Priority

The tree pointers, **right** and **left** are based on the Id property of each node.

## DATA SOURCE

Each node in the tree gets its data populated from a SQLite Database

| Field Name | Type |
|---|---|
| Id | String |
| Location | String |
| Category | String |
| Description | String |
| Image | Byte array (Kept in **BLOB** storage) |
| Status | String |
| Priority | String |

The Id field is an automatically incremented field while Location, Category, Description and Image are all inputted from the user.

Status is always defaulted as Pending for new entries while Priority is determined by the source code logic showcased in the following:

| Category | Priority Value |
|---|---|
| Sanitation | High |
| Utilities | Medium |
| Roads | Low |

## TREE CONSTRUCTION

During the construction of the tree there are two major processes, **Node Construction** and **Node Insertion**.

To construct a node for the binary tree, a connection to the database is opened, a query retrieves all records from the table in the database and each result is used to populate a **ServiceRequestNode.**

When **the ServiceRequestNode** nodes are inserted into the tree, Id field is used to determine the **pointer** of the node, if the node is smaller it goes to the **left** subtree and if the node is larger it will go to the **right** subtree.

## SEARCHING THE TREE

The search starts from the root that uses the **SearchTreeHelper** and checks if the status and priority filters match the node's values of each node. If both conditions are satisfied, the node data is converted into a **ServiceRequestUserControl** that will be displayed to the user.

### KEY FEATURES OF THE TREE

- Binary Search Tree Implementation: Optimised data insertion based on the Id field and organisation.
- Database Integration: The Tree pulls data from a SQLite database dynamically.
- Filtered Search Capabilities: The tree can filter base on status and priority at the same time.

## GRAPH

### GRAPH STRUCTURE

The graph structure utilises two classes, **ServiceRequestGraph** and **ServiceRequestNode.**

**ServiceRequestGraph** represents the overall graph, containing a collection of nodes. Each node represents a service request entry from the database. This class features the **AddNode** method to add a node to the graph.

**The ServiceRequestNode** class represents an individual node within the graph with the following properties:

- Id
- Status
- Priority
- Neighbours

### GRAPH CONSTRUCTION

To build the graph, the **BuildServiceRequestTree** method is utilised. This method gets data from the SQLite database and creates a **ServiceRequestGraph** containing nodes for each record.

# PROJECT COMPLETION

## PROJECT OVERVIEW

This POE has been built up through three iterations, Part 1, Part 2, and POE (Part 3).

### PART 1

**Part 1** featured the creation of the **Main page** which serves an **entry point** and introduces the user to the **navigation tools** as well as the **single exit** point. The functional requirement was for a user to be able to add a Service Report to a **List data structure**.

A page, in the form of a **User-Control**, was created where the user could input text fields and an image. Backend logic is executed by a **custom library** which made use of a **Validation class**, **Report Modal Class** and **Report Class**.

The **Validation class** ensure the user did not input unusable data according to the parameters of each field and would throw and would throw and error.

**Report Modal** just served as a blueprint for the **List**.

The **Report Class** added the data as a record into the list.

### PART 2

**Part 2** extended Part 1 by using the same project but adding the **Events** page.

The events page displays a series of events read from a file. The page features normal filtering and searching capabilities using **user input**, **selected date** and **type** of event as well as a **Recommendation Algorithm.**

This functionality utilised a new class in the **custom library** class. The **Events** class handles all filtering logic and recommendation algorithms.

### PART 3

Part 3 saw a large overhaul of the Reports page by implementing a local SQLite database to store report, a class that runs asynchronously from the main program to simulate a service request being completed and updating the information, the implementation of the **Service Request** page and improvements to the general exception handling of the code.

For the functional requirement, the **Service Request** page was added that references to two local classes that handle the implemented **Binary-Tree** and the **Graph**.

## CHALLENGES

### USER INTERFACE DESIGN

As a developer, I have always lacked the creative and technical skills to make very appealing **User-Interface**. Balancing colour schemes and designing appealing and readable elements to a page always come with great difficulty.

To overcome these challenges, I made sure to follow a strict colour pallet for the application, design elements with round edges where possible and reasonable and to incorporate the advantages of using **User-Controls** to create modular and re-usable elements.

### USER-CONTROLS

For this project, the **WPF** platform was utilised with the intention of remaining on the same "Window" to avoid the "Popping" experience when creating new windows. To achieve this **User-Controls** were implemented.

This feature allowed me to stay within the same main window while being able to seamlessly change the **user-interface**, providing the experience of changing pages without the mention popping affect.

As this was my first time using user-controls in WPF, organising events and triggers that needed to interact with the main window logic was a new challenge that I overcame. Organising and separating the different user controls to occupy different components of the main window was implemented seamlessly at the end of the project.

### RELATIVE CONNECTION STRING

During the third iteration of the POE, making use of a proper **relative path** for the database **connection string** was incredibly challenging due to the lack in a technical detail in the **appl.xaml**.

Towards the end of the project, with the assistance of reading the **SQLite documentation** (Kreibich, 2010), the solution was discovered and implemented.

### EXCEPTION HANDLING IMPLEMENTATION

Implementing affective **exception handling** into the already existing code was a challenge that took some time to implement. The **logic** of **methods** and their **returns** had to be heavily adjusted and changed to incorporate exception handling.

This did result in more robust and safer source and minimised the potential for crashes.

# TECHNOLOGY RECOMMENDATIONS

## DEDICATED DATABASE SERVER

The use of a dedicated database such as Microsoft SQL Server Manager to store users, events, and reports could be very beneficial to the project. (Ross Mistry, 2014)

The project uses a local database which will increase the file size of the project, depending in the number of records. By using a dedicated database server, the application will be made smaller and lightweight, therefore easier to run on an individual computer.

## DEDICATED BACKEND-SERVER FOR SERVICE REQUESTS

The application currently runs a simulated backend of updating the Service Requests in real-time. This process does use processing power and could be done, instead, by a dedicated server to interact with the database and update the status of each record by an admin user.

This would make the program even more lightweight and provide a far more realistic experience. (Somnath, 2001)

## REFERENCES

Kreibich, J. (2010). *Using SQLite.* Sebastopol: O'Reily Media Inc,.

Ross Mistry, S. M. (2014). *Introducing Microsoft SQL Server 2014.* Redmond: Microsoft.

Somnath, D. &. (2001). *Remote diagnosis server architecture.* IEEE Systems Readiness Technology Conference.