Service Request Management System: Implementation and Testing Report

Prepared By: Tiffany-Amber Jacobs

Date: 18 November 2024

1. Project Overview

- The purpose of the Service Request Management System is to effectively monitor, arrange, and display requests for municipal services. It has the following features:
- CRUD operations on service requests using a RESTful API.
- PieChart visualizations that display the distribution of request statuses.
- Data structures are integrated for better data management.
- This report summarizes the implementation, testing, and key learnings from the project.

2. Implemented Features

2.1. Core Features

1. **API Endpoints**:
    a. `GET /api/ServiceRequests`: Retrieve all service requests.
    b. `POST /api/ServiceRequests`: Add a new service request.
    c. `PUT /api/ServiceRequests/{id}`: Update an existing request.

2. **Visualization**:
    a. PieChart displays the proportion of requests by status (`Pending`, `In Progress`, `Completed`).
3. **Search Functionality**:
    a. Allows searching for requests by `ID` with detailed feedback.
4. **Data Structures**:
    a. **Binary Search Tree (BST)**: Organizes requests for efficient ID-based searching.
    b. **Graph**: Represents dependencies between service requests.

3. Data Structures Used

3.1. Binary Search Tree

- **Purpose**:
  - Organizes service requests by ID for quick retrieval.
- **Contribution to Efficiency**:
  - Allows O(log n) search operations for large datasets.
- **Example**:
  - Searching for a request with ID 3 is faster compared to a linear search.

3.2. Graph

- **Purpose**:
  - Manages dependencies between service requests.
- **Contribution to Efficiency**:
  - Enables related requests to be identified and visualized easily.
- **Example**:
  - Request 2 depends on 1. When 1 is completed, related requests are displayed.

4. Testing Summary

4.1. API Testing

All API endpoints were tested using **Swagger** and **Postman**.

Endpoint	Test Case	Result
`GET /api/ServiceRequests`	Retrieve all requests.	Passed (200 OK)
`POST /api/ServiceRequests`	Add a valid service request.	Passed (201 Created)
`PUT /api/ServiceRequests/1`	Update a request with valid data.	Passed (200 OK)

4.2. Visualization Testing

Test Case	Result
PieChart reflects request statuses.	Slices proportionate to counts; labels correct.
Data updates on adding requests.	PieChart re-renders dynamically.

4.3. Error Handling

Scenario	Result
Missing required fields (POST).	`400 Bad Request` with validation errors returned.
Non-existent ID (PUT/DELETE).	`404 Not Found` error returned.
Invalid JSON format (POST/PUT).	`400 Bad Request` error returned.

5. Challenges and Solutions

Challenge	Solution
`ECONNREFUSED` error in Postman testing.	Verified API port and adjusted `launchSettings.json`.
Empty PieChart for empty datasets.	Added validation to handle cases where no data exists.
Complex data relationships in graphs.	Implemented adjacency lists for efficient traversal.

6. Key Learnings

1. **Practical Use of Data Structures**:
    a. Leveraged BST for efficient search and Graph for dependency mapping.
2. **API Development**:
    a. Built robust APIs using REST principles with comprehensive error handling.

3. **Visualization**:
    a. Enhanced the user experience with dynamic charting (LiveCharts).

7. Technology Recommendations

7.1. Entity Framework Core

- **Purpose**: Simplify database interactions.
- **Justification**: Reduces boilerplate code and improves maintainability.

7.2. Blazor

- **Purpose**: Build a modern, interactive frontend.
- **Justification**: Streamlines UI development and integrates well with .NET.

7.3. Azure App Service

- **Purpose**: Host the application in the cloud.
- **Justification**: Ensures scalability and reliability for real-world deployment.

8. Next Steps

- Deploy the application to a cloud environment.
- Add user authentication and authorization features.
- Optimize performance for larger datasets.

9. Conclusion

The project effectively illustrates the application of sophisticated data structures, visualization, and API architecture. Every feature works as it should, and the program is prepared for distribution or additional development.