Dentist Appointment Management System
Group Coursework CST2550

Team Members :

[M00958497] Yug Patel - SCRUM Master

[M00941941] Noopur Patel - Secretary

[M00951646] Atharva Garud - Developer # 1

[M00939931] Dev Patel - Developer # 2

[M00970345] Mahmudul Saif - Tester

Module: CST2550 – Data Structures & Algorithms

Table of Contents

1. Introduction

This project is a C# (.NET) console-based application for managing dental appointments and keeping track of staff members. The system's features for managing staff information and scheduling, searching, and modifying appointments are intended to help dental offices. Spectre.console library was used to create an easy-to-use command-line user interface. The library improves usability.

2. Design

2.1 Data Structure

Our key data structure is a custom Binary Search Tree (BST) implemented in the AppointmentBST class. The BST organizes appointments by patient name and allows efficient insertion, search, and removal operations.

Justification :

- **Efficiency:**
  The BST offers average-case time complexity of O(log n) for search, insert, and delete operations. This meets the performance requirements of the system.

- **Grouping:**
  Appointments for patients with the same name are grouped together in the same node, which simplifies the retrieval of all appointments for a particular patient.

- **Manual Implementation:**
  As stipulated by the project brief, we did not use any third-party libraries or built-in STL data structures for this task. Instead, the BST was implemented from scratch, ensuring a deep understanding of the underlying algorithms and time complexity.

2.2 Algorithm Analysis

- **Insertion :**
  The algorithm compares the patient name and recursively finds the correct position in the BST. If a node with the same name exists, the new appointment is added to that node's list.

  Time Complexity :
  Average-case O(log n) (worst-case O(n) if the tree becomes unbalanced).

- **Search :**
  The search algorithm traverses the tree based on a string comparison of patient names.

  Time Complexity : Average-case O(log n) (worst-case O(n)).

- **Removal :**
  Removal is implemented by traversing the BST to locate a matching appointment ID. If removal leaves a node empty, tree restructuring occurs using the in-order successor.

  Time Complexity : Worst-case O(n) due to traversal and tree re-balancing steps.

2.3 Pseudocode Example

Below is simplified pseudocode for the insertion operation in the BST :

```
function Insert(node, appointment):
if node is null:
create new node with appointment
else:
compare appointment.PatientName with node.Key
if equal:
add appointment to node.Appointments
else if less:
node.Left = Insert(node.Left, appointment)
else:
node.Right = Insert(node.Right, appointment)
return node
```

3. Testing

3.1 Testing Approach

We applied a combination of unit tests and integration tests to ensure that individual components and the overall system function correctly. Our test suite covers:

- **BST Operations:**
  Unit tests for insertion, search, removal, and in-order traversal (see **AppointmentBSTTests.cs**).

- **Database Integration:**Tests for inserting and removing appointment records, ensuring proper interaction with SQL (see **DatabaseIntegrationTests.cs**).

- **File Integration:**
  Tests that validate the system's ability to load appointments from a file, including error handling for malformed data (see **FileIntegrationTests.cs**).

- **Edge Cases & Performance:**
  Unit tests covering boundary conditions and performance testing to ensure acceptable insertion times (see **EdgeCaseTests.cs** and **PerformanceTests.cs**).

3.2 Test Cases Summary

Test Category	Purpose
BST Insertion	Verify that single and multiple appointments are correctly inserted and grouped by patient name.
BST Search	Confirm that search returns correct appointment groups or null when not found.
BST Removal	Ensure removal by appointment ID functions as expected.
Database Integration	Validate that appointments can be inserted into and removed from the SQL database.
File Integration	Check that valid file input correctly loads appointments and malformed lines are skipped.
Edge Cases	Confirm that operations on an empty BST return the correct results.
Performance	Measure that bulk insertion (e.g., 10,000 records) completes within acceptable time.

Justification for Using the Spectre.Console Library

We opted to use Spectre.Console in our project to provide a modern, intuitive, and visually appealing user interface for our console application. The primary focus of our coursework is on designing and implementing a custom data structure with detailed time complexity analysis, and using Spectre.Console allows us to reduce the amount of custom code needed for the UI. This decision offers several advantages:

- **Enhanced Aesthetics and Usability:** Spectre.Console provides rich formatting options (such as rounded panels, centered headers, interactive selection prompts, and customizable rules) that greatly enhance the end-user experience. This allows our application to have a professional look and feel while ensuring clarity in information presentation.

- **Focus on Core Functionality:** By leveraging Spectre.Console, we minimize the time spent on developing custom UI elements. This lets us allocate more effort toward designing and analyzing our custom binary search tree and associated algorithms, which are the core focus of our project.

- **Efficient Code Organization:** The library simplifies the implementation of a dynamic and interactive menu system, which helps maintain clean and readable code. This is critical given the 7-page report limit, as it allows us to present our design choices and algorithm analysis without cluttering our report with lengthy UI implementation details.

- **Industry-Relevant Tools:** Although Spectre.Console is a third-party library, it is widely adopted and maintained within the .NET community, and its usage aligns with modern practices for building console applications. We ensure that its adoption does not conflict with the brief, as we have explicitly kept its use to improve user experience and code clarity.

In summary, incorporating Spectre.Console enables us to deliver a polished, user-friendly interface while allowing us to concentrate on the innovative aspects of our project—namely, our custom data structure design and algorithm analysis. This strategic decision enhances both our development process and the final product, ensuring compliance with project requirements while maximizing our report's focus on core technical contributions.

4. Conclusion & Summary of Work Done

In this project, we designed and implemented a custom Binary Search Tree (**AppointmentBST**) from scratch—without relying on any builtin or third-party data structures—to store and efficiently manage appointment records keyed by patient name. We built a rich, interactive console UI using Spectre.Console, supporting both Admin and Employee roles, CRUD operations on appointments and staff, bulk file import, and full SQL Server integration for persistence. To ensure quality and robustness, we authored a comprehensive MSTest suite covering unit, edge-case, file-integration, performance, and database-integration tests, and automated our build/test workflow via a Makefile locally and GitHub Actions in CI.

The Dentist Appointment Management System demonstrates a robust and scalable solution for managing appointments and employee records in a dental practice. The use of a custom BST provides efficient search and management of appointment data, and the solution adheres strictly to the project requirements by avoiding third-party libraries for data structures. Comprehensive testing ensures the integrity of the solution, while detailed input validations help maintain data quality.

Limitations and Future Improvements :

- **Unbalanced BST:**
  In worst-case scenarios, the BST could become unbalanced leading to slower operations. Future work could involve self-balancing trees (e.g., AVL or Red-Black trees).

- **User Interface Enhancements:**
  While the console interface is functional, a graphical user interface (GUI) could improve user experience.

- **More Comprehensive Testing:**
  Further automated tests for interactive methods could be implemented by refactoring for better testability.

Reflection :

The project was a valuable exercise in designing custom data structures and integrating them into an application. Through iterative development and testing, we gained deeper insights into algorithm analysis and code maintainability, learning how to balance functionality with user input validation and error handling

5. References

McGugan, W. (2025) Spectre.Console Documentation.
Available at : https://spectreconsole.net
(Accessed: 10 April 2025).

Microsoft (2023) Unit testing C# with MSTest.
Available at: https://learn.microsoft.com/dotnet/core/testing/unit-testing-with-mstest
(Accessed: 1 March 2025).

Microsoft (2024) .NET 8.0 SDK Documentation.
Available at: https://learn.microsoft.com/dotnet/core/install
(Accessed: 1 March 2025).

Microsoft (2023) SQL Server Express Edition.
Available at: https://www.microsoft.com/en-us/sql-server/sql-server-downloads
(Accessed: 1 March 2025).