PET MANAGEMENT SYSTEM

CST2550 GROUP COURSEWORK

Prepared by
Mehtaab andoo & Team

16/04/25

REPORT

TABLE OF CONTENTS

INTRODUCTION

The Pet Care Management System by Paws&Co is a comprehensive digital hub for veterinary clinic personnel. It streamlines access to integrated patient information, enabling receptionists and veterinarians to manage client data efficiently. Key features include new pet registration, profile viewing, and advanced searches that retrieve details like species, breed, age, owner data, medical history, and treatment follow-ups—all through a unified interface.

Lucca Veterinary Data Security (2024) notes that robust data management is vital for quality clinical care and operational efficiency. By consolidating pet and owner records, medical histories, and appointment scheduling, the system enhances data accuracy and accessibility in fast-paced veterinary environments

Report layout

The remainder of this document is organised into five substantive sections that correspond to the assessment brief:

1. **Design** – presents a rationale for the chosen data structures and algorithms, followed by a detailed analysis of the algorithms that deliver the system's core functionality. Each algorithm is expressed in clear pseudo-code to emphasise logic over implementation detail.
2. **Testing** – outlines the overall test strategy adopted (including levels, techniques, and coverage goals) and provides a comprehensive table of test cases that evidences how each requirement was verified.
3. **Conclusion** – synthesises the work undertaken, highlights the system's limitations with a critical reflection on their root causes, and proposes how the development approach would be refined on a comparable future project to avoid repeating mistakes.
4. **References** – supplies full Harvard-style citations, each matched to in-text references, alongside inline code comments that acknowledge external sources used in the software artefact.

Design

Justification of selected data structure and algorithms.

The primary data structure used in our project is the Hash table, chosen for its remarkable efficiency and suitability for fast data access. A hash table consists of an array that either stores values directly or through structures like linked lists.

Its key feature is the hash function, which converts a key into an array index, enabling rapid insertion, lookup, update, and deletion operations. Hash tables stand out for their extraordinary performance; most operations run in constant time (O (1)) on average, regardless of data size. According to Tapia-Fernández, García-García, and García-Hernandez (2022), this level of efficiency explains why hash tables are both widely adopted in real-world applications and frequently studied in computer science.

A key advantage of hash tables is their ability to provide immediate access to records. Maurer and Lewis (1975) emphasize that, in most cases, data can be retrieved without multiple comparisons, thanks to the direct indexing provided by the hash function. This constant-time retrieval capability aligns perfectly with systems where speed and efficiency are critical. Compared to other data structures, hash tables consistently outperform when handling large datasets, which makes them an ideal choice for our project's needs.

HashTable Structure (Pseudo-code)

Initialize HashTable with:
Capacity ← 11
LoadFactor ← 0.75
Buckets ← array of size Capacity
Size ← 0
2. Insert(Key) Algorithm (with linear probing)

Function Insert(key):
HashValue ← GetHash(key)
Index ← HashValue mod Capacity

While Buckets[Index] is not empty:
If Buckets[Index].Key = key:
Update the value at Buckets[Index]
Return
Index ← (Index + 1) mod Capacity

Buckets[Index] ← key
Size ← Size + 1

If Size ≥ LoadFactor × Capacity:
Resize the hash table

3. Remove(Key) Algorithm

Function Remove(key):
HashValue ← GetHash(key)
Index ← HashValue mod Capacity

While Buckets[Index] is not empty:
If Buckets[Index].Key = key:
Buckets[Index] ← null
Size ← Size - 1
Return success
Index ← (Index + 1) mod Capacity

Return failure (key not found)

4. Search(Key) Algorithm
Function Search(key):
HashValue ← GetHash(key)
Index ← HashValue mod Capacity

While Buckets[Index] is not empty:
If Buckets[Index].Key = key:
Return Buckets[Index].Value
Index ← (Index + 1) mod Capacity

Return null (key not found)

Operation	**Best Case**	**Average Case**	**Worst Case**
Insert	O(1)	O(1)	O(n)
Remove	O(1)	O(1)	O(n)
Search	O(1)	O(1)	O(n)

Testing

Statement of testing approach used

Unit Testing

- HashTable Operations: We rigorously tested insertion, search, and deletion (with linear probing and tombstones) using standard unit tests to validate collision handling and performance.
- Unique ID Generation: Unit tests confirmed that generated IDs are unique across both the in-memory hash table and the database.
- Input Validation: Functions for name, email, phone, and address validation were verified with both valid and invalid inputs.

Integration Testing

- Database Upsert and Deletion: Using an in-memory SQL Server (via EF Core), we tested that our upsert methods correctly add or update records and that cascading deletions (e.g. deleting an owner also removes associated pets and appointments) work as intended.
- CSV Parsing: End-to-end tests with sample CSV files ensured that data is correctly parsed into hash tables, while erroneous rows are properly handled.

UI & Manual Testing

- CLI Navigation: Manual tests confirmed that the text-based interface (via Spectre.Console) is intuitive, with clear prompts, validations, and status messages.
- User Interaction: Input prompts (including default values and error cases) were tested to ensure a smooth user experience.

Performance & Edge-Case Testing

- Scalability: We simulated large data sets to verify that hash table operations remain efficient under load.
- Error Handling: Tests covered invalid inputs (e.g. impossible dates) and ensured that operations such as deletion in CSV mode are correctly disabled to prevent data loss.

Below is a table with the testing scripts used and the status.

Test Id	Tester	Test Script	Test Data	Expectation	Status	Notes	Retest Status
1	All	Browse to Vs Code and run the code. Make sure the whole project is choosen, instead of 1 file.		The code should run without any issues.	fail	Some nuggets packages were missing. We used donet.restore in package manager console.	Pass
2	All	Browse to Vs code. Make a change in the code. Save the code. Click on the 'Git' button on top and push the code to the main or branches being used after adding a commit. Browse to git hub using the link in the test data to the respective repository. Check in the commit log.	https://github.com/umarAmeerally/Paws-Co.-Management/tree/master	The recent commit should be visible	Pass		
3	All	Run the program and choose option load data from csv file		The data in the csv file should load with a successful message: CSV parsed and data loaded into HashTables.	Pass		
4	All	Run the program and choose option load data from database		The data from database should load with a successful message: Data loaded from database into HashTables.	Pass		
5	All	Run the program. After choosing a data source (CSV or Database) choose option 'Display all owners'		The owners should be displayed and for further checking run the following query in SSMS: select * from owners	Pass		
6	All	Run the program. After choosing a data source (CSV or Database) choose option 'Add new owner' and click save all data.		The owner should be added successfully and for further checking run the following query in SSMS: select * from owners	Pass		
7	All	Run the program. After choosing a data source (CSV or Database) choose option 'Update Owner'. Add the owner id and update the details.		The owner shoud be updated and for further checking run the following query in SSMS: select * from owners	Pass		
8	All	Run the program. After choosing a data source (CSV or Database) choose option 'Delete Owner'.		The owner shoud be deleted and for further checking run the following query in SSMS: select * from owners	Pass		
9	All	Run the program. After choosing a data source (CSV or Database) choose option 'Display all pets'.		The pets should be displayed and for further checking run the following query in SSMS: select * from pets	Pass		
10	All	Run the program. After choosing a data source (CSV or Database) choose option 'Add new pet' and click save all data.		The pet should be added successfully and for further checking run the following query in SSMS: select * from pets	Pass		

Test Id	Tester	Test Script	Test Data	Expectation	Status	Notes	Retest Status
11	All	Run the program. After choosing a data source (CSV or Database) choose option 'Update pet'.		The pet shoud be updated and for further checking run the following query in SSMS: select * from pets	Pass		
12	All	Run the program. After choosing a data source (CSV or Database) choose option 'Delete pet'.		The pet shoud be deleted and for further checking run the following query in SSMS: select * from pets	Pass		
13	All	Run the program. After choosing a data source (CSV or Database) choose option 'View Appointment'.		The appointment should be displayed and for further checking run the following query in SSMS: select * from appointments	Pass		
14	All	Run the program. After choosing a data source (CSV or Database) choose option 'Add Appointment' and click save all data.		The appointement should be added successfully and for further checking run the following query in SSMS: select * from appointments	Pass		
15	All	Run the program. After choosing a data source (CSV or Database) choose option 'Update Appointment'.		The appointment shoud be updated and for further checking run the following query in SSMS: select * from appointments.	Pass		
16	All	Run the program. After choosing a data source (CSV or Database) choose option 'Search'.		The correct searched data should be displayed	Pass		
17	All	Run the program. After choosing a data source (CSV or Database) choose option 'Exit'.		The console window should close after pressing any key on the keyboard	Pass		

Conclusion

Development:

- We implemented a Pet Management System using C# with a custom hash table for in-memory data storage, EF Core for database operations, and Spectre.Console for a robust text-based user interface. Key functionalities included CSV parsing, unique ID generation, CRUD operations, and data validation.

Testing:

- We conducted unit and integration tests alongside manual user interface testing. This ensured that data insertion, deletion, search, and CSV/database synchronisation worked reliably. Performance under load was also simulated to validate efficiency.

Limitations and Critical Reflection

- A significant limitation was the instability of VSCode experienced during Git uploads. The crashes disrupted workflow and added overhead to managing source control.

- The issues with the development environment were a primary factor in slowing progress. Limited hardware resources and misconfigurations in VSCode contributed to these disruptions.

Process Impact:

- These challenges led to delays, occasional data loss, and increased manual intervention in tracking changes. It highlighted the need for a more resilient version control process and a stable IDE setup.

Future Approach – Avoiding Repeated Mistakes

- Prioritise a more stable development environment. Consider using an alternative IDE or ensuring VSCode is optimally configured (extensions, memory settings, etc.) to handle the project's scale.

- Adopt a more cautious approach when uploading large commits. This might include breaking down changes into smaller commits, utilising branching strategies, and testing the Git upload process in a controlled environment to avoid crashes.

- Integrate continuous integration tools to run automated tests and Git operations in a contained environment. This will help isolate issues related to local development setup from the codebase itself.

References

Lucca Veterinary Data Security (2024) Available at: https://lucca.vet/the-critical-importance-of-data-management-in-veterinary-it-clinics/ (Accessed: 10 April 2025).

Maurer, W.D. and Lewis, T.G. (1975) Hash table methods. ACM Computing Surveys (CSUR), 7(1), pp.5-19. Available at: https://doi.org/10.1145/42404.42410 (Accessed: 10 April 2025).

Tapia-Fernández, S., García-García, D. and García-Hernandez, P., (2022). Key Concepts, Weakness and Benchmark on Hash Table Data Structures. Algorithms, 15(3), p.100. Available at: https://doi.org/10.3390/a15030100 (Accessed: 10 April 2025).