Middlesex University

Please complete, sign it, scan it and put it on the FRONT page of your report

Group Name (if you have one): Pawfect Care Ltd	
Group Scribe (student uploading this report): Student Number: M00966602	
Module tutor: Ms Aisha Idoo	
Submission Date: 16/04/2025	
Individual Contribution to Group Project	Peer Assessment
Outstanding	The contribution was significantly OVER AND ABOVE the performance of any other group members; hence ONLY ONE member per group can have this ranking.
Good	The contribution was valuable and significant either in content or in underpinning the group as a whole.
Average	The contribution was average. No more or less than could reasonably be expected.
Poor	The contribution was minimal OR the member did not contribute at all.

Student Surname (Please type)	Student number	% attendance all meetings	Contribution To Essay	Signature of each student
Busgeeth	M00978514	100%	Good	
Dhunputh	M00983700	100%	Good	
Pothanah	M00965898	100%	Good	
Rojoa	M00966602	100%	Good	
Takooree	M00959918	100%	Good	

Table of Contents

Table of Figures

Introduction

The Pet Care Management System was developed for Pawfect Care LTD to digitally transform the company's manual operations for managing pets, owners, veterinary appointments, medications, prescriptions, supplier and order. The system is designed to operate across two physical store locations and ensure efficient, scalable, and accurate management of pet health records and services. With the veterinary industry facing increasing demands for streamlined digital systems, this software solution seeks to enhance customer experience and enable real-time access to data, supporting both operational and clinical decision-making.This report documents key aspects of the software engineering process, particularly focusing on data structure selection and algorithm design. The Design section discusses the reasons for selecting a hash table and its suitability in a veterinary management context. It also includes an analysis of the system's core functionalities. Following that, the Testing section describes the methods used to validate the solution, and the Conclusion offers a critical reflection on the results and future improvements.

Project Management approach

The project was managed using the **Agile software development methodology**. Daily stand-up meetings were held to review progress, address any arising issues, and adapt objectives for each sprint cycle. As each stand-up meeting corresponded to a new sprint, a total of 30 sprints were completed over the 30-day period. Tasks were organised into a product backlog, which was continuously prioritised based on feedback from the team and testing system. Stakeholder feedback (from peers and lab tutors) was integrated at each stage to refine system features and enhance usability. Our adoption of Agile is supported by Beck et al. (2001), who emphasised adaptive planning and rapid delivery, and Dingsøyr et al. (2012), who highlighted Agile's effectiveness in managing uncertainty and promoting responsiveness in software projects.

Design

Justification of Selected Data Structure(s)

The central challenge of this system is managing and retrieving large volumes of structured data efficiently. This includes thousands of pet and owner records, multiple daily appointments, and extensive medication tracking. The selected data structure must support real-time operations, scalability, and fast lookups, insertions, and deletions. After evaluating multiple options—**Array Lists**, **Linked Lists**, **Binary Trees**, and **Hash Tables**—the **custom-built hash table** was selected for the following reasons:

- **Hash Tables offer constant time complexity** (O(1)) for most operations including insertion, deletion, and retrieval when a good hash function is used (Shaffer, 2022). This efficiency makes them ideal for systems requiring frequent access to specific records, such as PetID or AppointmentID.
- **They support key-based access**, which aligns with the unique identifiers used in the system's data model. This enables immediate access to pet history, appointments, or prescriptions without sequential searches.
- **They are scalable** and maintain performance even as the dataset grows, which is essential for a long-term deployment within a veterinary business context (Goodrich et al., 2022).

Data Structure	Insertion Time	Search Time	Deletion Time
Array List	O(n)	O(n)	O(n)
Linked List	O(1)/O(n)	O(n)	O(n)
Binary Tree	O(log n)	O(log n)	O(log n)
Hash Table	O(1)	O(1)	O(1)

The hash table implementation used in the project was written manually without relying on .NET's built-in Dictionary<K,V> or other libraries, to meet academic constraints and gain full control over the hashing and collision-handling logic.

Analysis of Algorithms and Functional Design

The key operations performed by the system—adding a new pet, searching for an owner, updating an appointment, or deleting a record—are all supported through the hash table's efficient architecture. The use of **open hashing (chaining)** ensures robustness when handling collisions, especially important when dealing with high volumes of concurrent data entries. Algorithms for CRUD operations were analyzed based on their **time complexity**:

- **Create**: We used a hash table to insert records into the in-memory database by mapping primary keys to record objects for fast access and storage.
- **Read:** The in-memory hash table allowed quick retrieval of records using primary keys or field-based filters without querying the SQL database.
- **Update:** We performed updates directly on the records stored in the hash table by modifying their fields and syncing changes to the database.
- **Delete:** Records were efficiently deleted from the in-memory hash table using their keys, followed by removal from the SQL database.

Pseudocode for Key Algorithms

```
Record Pseudocode.
CLASS Record
Fields (— Dictionary<string, object>
FUNCTION Get(fieldName)
RETURN Fields[FieldName]
END FUNCTION
        FUNCTION Set(fieldName, value)
Fields[FieldName] (— value
END FUNCTION
        FUNCTION OVERRIDE ToString()
RETURN stringified Field map
END FUNCTION
END CLASS
```

Figure 1: Defining record class

```
Database Pseudocode.
CLASS Database
Dictionary<string, Table> tables
FUNCTION AddTable (Table table)
tables[tables.name] = table
END FUNCTION
FUNCTION GetTable(STRING name) RETURN Table
RETRUN tables[name]
        END FUNCTION
END CLASS
```

Figure 2: Database class

```
Hash Table Pseudocode.
CLASS HashTable
SIZE (— 101
bucket (— array of SIZE linked list.
FUNCTION Add (key, value)
index (— hash(key) MOD SIZE
IF key exist in buckets[index]
THROW "Key already exist"
ADD(key, value) TO buckets[index]
END FUNCTION
FUNCTION Get (key)
index (— hash(key) MOD SIZE
FOR EACH (k, v) IN buckets[index]
IF k = key THEN
RETURN V
THROW "Key not found."
END FUNCTION
FUNCTION Remove (key)
index (— hash(key) MOD SIZE
FOR EACH (k, v) IN buckets[index]
IF k = key THEN
REMOVE (k, v) FROM buckets[index]
RETURN TRUE
THROW FALSE
END FUNCTION
FUNCTION ContainsKey (key)
index (— hash(key) MOD SIZE
RETURB TRUE IF key exists in bucket[index] else RETURN False
END FUNCTION
FUNCTION GetALL()
FOR EACH list in bucket
IF bucket != FALSE
RETURN all (k, v) pairs IN bucket
END FUNCTION
```

Figure 3: Hash table pseudocode

```
Table Pseudocode.
CLASS Table
STRING name
STRING primaryKey
HashTable<string, Record> rows
        FUNCTION Insert (Record record, bool skipDb = FALSE)
                key (— record[primaryKey]
                rows.Add(keys, record)
                IF skipDb = FALSE THEN
                        Save changes to the SSMS database
        END FUNCTION
        FUNCTION Get (STRING key) RETURNS Record
        RETURN rows.Get(key)
        END FUNCTION
        FUNCTION Update(key, fieldnName, value, skipDb = TRUE)
        record (— reows.Get(key)
        Record[fieldName] = value
                IF skipDb = FALSE THEN
                        Save changes to the SSMS database
        END FUNCTION
        FUNCTION Delete(key)
        rows.Remove(key)
        END FUNCTION
        FUNCTION GetAll() RETURN List<Record>
                RETURN rows.GetALL().Values.
        END FUNCTION
END CLASS
```

Figure 4: Table pseudocode

```
CRUD.
Create Record into Table.
FUNCTION InsertRecord(tableName , record, skipDb = FALSE)
table (— database.GetTable[tableName]
key (— record[table.primaryKey]
IF table.hashTable.Contains[key] THEN
THROW Exception "Key already exist."
Table.hashTable.Add(key, record)
        IF skipDb = FALSE THEN
                Save changes to the SSMS database
                END FUNCTION
```

Figure 5: Create Record algorithm

```
Read Record from Table.
FUNCTION GetRecord(tableName, coloum, value)
table (— database.GetTable[tableName]
        Record  (— table.Get.coloumn where coloumn= value]
IF record = NULL THEN
                RETURN "NO SUCH RECORD"
        ELSE
                RETURN RECORD
END FUNCTION
```

Figure 6: Read record algorithm

```
Update Record Field in Table.
FUNCTION UpdateRecord(tableName, primaryKey, coloumn, value, isForeignKey = FALSE, skipDb = TRUE)
table (— database.GetTable[tableName]
        IF isForeignKey = TRUE THEN
        foreignTable (— database.GetTable[foreignTableName]
IF foreignTable.hashTable.Contains[value] = FALSE THEN
THROW Exception "Value does not exist in ' foreignTable '."
table.Update(primaryKey, coloumn, value)
        IF skipDb = FALSE THEN
                Save changes to the SSMS database

END FUNCTION
```

Figure 7: Update Record algorithm

```
Delete Record Field in Table
FUNCTION DeleteRecord(Key)
table (— database.GetTable[tableName]
records (— table.Get.primaryKey where coloumn = key
        IF records = NULL THEN
        FOR EACH record IN records.
                table.Delete(records)
End FUNCTION
```

Figure 8:Delete Record algorithm

User Interface

Implementation meets design

The user interface of the Pet Care Management System was developed to offer a visually clean and functionally efficient experience for veterinary staff. The first image shows the **Login screen**, which allows authorised users to access the system using a secure username and password combination.

The **Sign-Up screen** supports new user registration by capturing essential details such as username, email, and password. Upon logging in, user is directed to a dashboard. This includes **Home**, **Tables**, **Users**, and **Operations**.The **Operations module** provides options for **registering new pet owners and booking veterinary appointments**. When user registers a new owner, registration for a new pet is required, so that an owner has at least one pet. Moreover, the tables tab allows user to select a specific table where all data are displayed in a grid. User can also filter data by searching for a specific attribute. Appointment details can also be changed upon clicking on Update button.

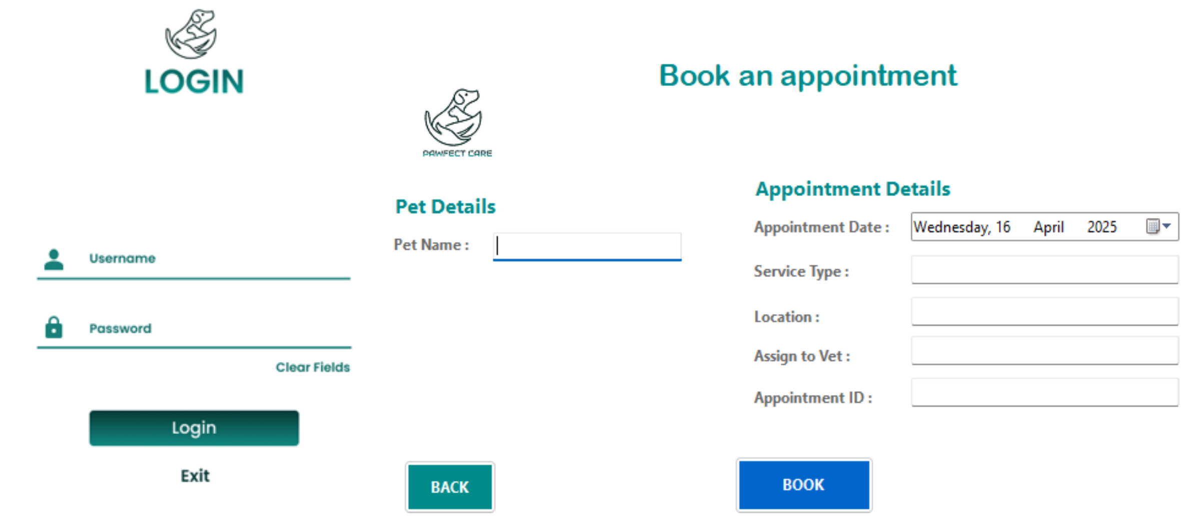

Figure 9: User Interfaces

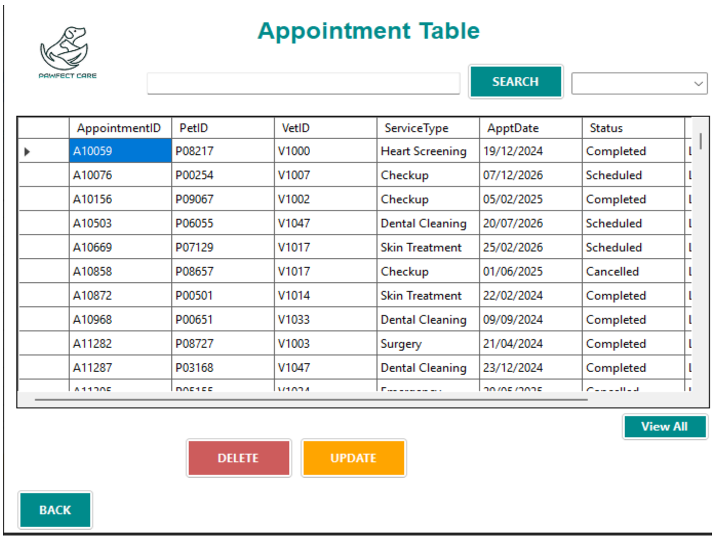

Figure 10: Appointment Table Interface

Testing

Testing Approach

The testing of the Pet Care Management System was conducted using a combination of **automated unit testing** and **manual interface testing**. The automated tests were implemented using the **MSTest framework** in C# to validate the integrity of core logic and data handling functions. MSTest was selected for its seamless integration with Visual Studio and its structured assertion-based testing model. Automated tests targeted internal logic such as hash table insertions and lookups,

Alongside unit tests, **manual functional testing** was performed on the user interface. This involved simulating user interactions such as registration, login, form submissions, and CRUD operations (insert, update, delete). Each module (e.g., Tables, Operations, Users) was tested for expected outcomes, including validation messages, error handling, and data persistence across sessions.

Test Case Table for Owner CRUD

Test Case ID	Test Scenario	Input	Expected Result	Actual Result	Status
TC001	Add new owner	Valid owner data	Owner saved in DB and retrievable	As expected	Pass
TC002	Retrieve existing owner by ID	OwnerID = O10001	Owner retrieved with correct email	As expected	Pass
TC003	Update existing owner's address	Change Address to "Home"	Change Address to "Home"	As expected	Pass
TC004	Delete existing owner	OwnerID = O10003	Owner no longer in DB (null)	As expected	Pass
TC009	Delete a medication record	MedicationID: M00034	Record removed from database	As expected	Pass
TC010	Attempt to insert duplicate PetID	PetID: already exists	Error: "Duplicate entry not allowed"	As expected	Pass

Conclusion

Summary of Work Done

The Pet Care Management System was developed as a desktop application using C# and SQL Server to provide comprehensive digital support for managing veterinary operations at Pawfect Care LTD. The core logic of the system was built using a custom hash table to ensure high-performance data access and manipulation. An Agile methodology guided the development process, with 30 daily sprints supporting continuous delivery, feedback, and iteration. Testing was conducted through both unit tests using MSTest and manual functional validation of the user interface to ensure end-to-end reliability.

Limitations and Critical Reflection

One of the key limitations encountered was the complexity involved in implementing a fully custom hash table from scratch while ensuring robustness against collisions and memory fragmentation. While chaining helped address some of these issues, occasional inefficiencies were noted when handling a high number of deletions without rehashing. Additionally, although the use of an in-memory hash table improved runtime efficiency, syncing with the SQL database required additional update logic to maintain data consistency, increasing the overall system complexity. Moreover, initial versions of the user interface suffered from inconsistent form validation and limited error feedback, requiring several iterations to improve usability.

The challenge of managing code integration across multiple developers was also evident, with some early merge conflicts arising due to inconsistent naming conventions and lack of modular separation. This delayed certain components and highlighted the importance of stricter version control discipline.

Future Improvements

In future projects of a similar scope, a few key changes would be made to improve both development workflow and software quality. Firstly, implementing unit testing from the outset would ensure continuous integration reliability rather than relying primarily on testing in later stages. Test-Driven Development (TDD) could also be explored to ensure better design alignment (Ammann and Offutt, 2017).

Thirdly, the team would consider leveraging lightweight frameworks like SQLite for local data persistence during development, reducing the overhead of SQL Server configuration until deployment stages. Finally, formal documentation standards for code and commit messages should be enforced from the start, coupled with regular peer code reviews to improve team coordination and software robustness. Additionally, implementing a **critical path method** during the planning phase would allow the team to identify essential tasks, allocate appropriate time to each, and prevent overwhelming pressure close to the deadline due to poor time estimation.

References

1. Ammann, P. and Offutt, J. (2017). *Introduction to Software Testing*. 2nd ed. Cambridge University Press. DOI: 10.1017/9781108233657
2. Beck, K. et al. (2001) *Manifesto for Agile Software Development*. Available at: https://agilemanifesto.org
3. Chatterjee, S., Nguyen, B. and Ghosh, S.K. (2021). "Healthcare information systems: Architecture and algorithms". *Journal of Biomedical Informatics*, 118, 103774. Available at: https://doi.org/10.1016/j.jbi.2021.103774
4. Dingsøyr, T., Nerur, S., Balijepally, V. and Moe, N.B. (2012) ‘A decade of agile methodologies: Towards explaining agile software development’, *Journal of Systems and Software*, 85(6), pp. 1213–1221. Available at: https://doi.org/10.1016/j.jss.2012.02.033
5. Goodrich, M.T., Tamassia, R. and Goldwasser, M.H. (2022). *Data Structures and Algorithms in C#*. Wiley. Available at: https://onlinelibrary.wiley.com/doi/book/10.1002/9781119868005
6. Microsoft Docs (2024). *Unit testing C# in .NET using MSTest*. Available at: https://learn.microsoft.com/en-us/dotnet/core/testing/unit-testing-with-mstest
7. Shaffer, C.A. (2022). *Data Structures and Algorithm Analysis*. Available at: http://123.200.31.10:8080/jspui/bitstream/123456789/105/1/Data%20Structure%20%26%20Algorithm%20Analysis.pdf