ZENSOFT LTD – FANCY DRESS HIRE SYSTEM

Coursework 2 Group Project:

- Gheroghe-Florin Hasna M00910464
- Raisa Silea: M00854834
- Khalil Ait Jakane: M00854040
- Sadik Hyseni: M00908409
- Fahim Shirzad : M00863589

Contents

1. Introduction

Zensoft is making a bold effort to establish a smooth relationship between merchants and customers via a complex product borrowing mechanism. This project includes not only the monitoring of items and transactions, but also the improvement of customer satisfaction and the optimisation of inventory management through the use of innovative technology solutions. The programme focuses on creating a well-organized system that allows things to be borrowed, monitored, and returned easily. It also includes an innovative way for computing fines for overdue items. Zensoft's project ensures speedy, effective, and trustworthy performance by implementing carefully designed data structures and computational approaches.

The Zensoft Product Loan System production report describes in full the system's design and analysis, as well as a thorough justification of the data structures and methods used. The report is methodically organised to guide the reader through the system's various components. The text includes a thorough explanation of the functional features of essential operations such as product issuance and return, loan monitoring, and fine handling, supported by pseudo code examples.

The study contains a section on specialised testing based on design ideas. Here we look at the many testing methods used to ensure the strength and dependability of each code component. This section is critical since it not only confirms the functioning but also emphasises the challenges encountered during the testing process. This goes on to explain the problem-solving strategies used, including the process of diagnosing various difficulties and executing strategic solutions to effectively overcome them. The material includes test cases, debugging logs, and an iterative refinement process, all of which have helped to improve the system's dependability and performance.

The report finishes by evaluating the system's ability to handle increasing demand and dependability, as well as providing a detailed analysis of potential improvements. This area is intended to propel future advancement, ensuring that Zensoft remains at the forefront of the industry by providing cutting-edge solutions in an ever-changing market.

2. Design

UMLs

We used UML (Unified Modelling Language) diagrams to conceptualise and organise the architecture and functioning of the system during the design phase of our project. The UML diagram that was crucial in establishing the architecture of the system. The class diagram we used for our final software can be found in the appendix labelled figure 1. We had two different class diagram designs and we choose figure 1 over figure 2 as it was a more efficient design for our software.

Activity Diagrams:

Activity diagrams were used to show the order of stages or activities in our system's various workflows or processes. These diagrams give a clear picture of how the system behaves, making it easier to spot possible bottlenecks, streamline procedures, and guarantee effective operation. An example of an activity diagram can be found in the appendix labelled under figure 3.

Every stage of the registration process, including user input, validation processes, and database interactions, is shown in detail in this diagram. We were able to improve user experience and system performance by streamlining the registration process by visualising the workflow in this way.

Use Case Illustration:

The relationships between actors (people, systems, or external entities) and the system under development were modelled using use case diagrams. Our use case illustration can be found in the appendix labelled figure 4.

The users in this diagram are stick figures that interact with the system through a variety of use cases, each of which stands for a particular feature or functionality of the system. We were able to determine the essential

features needed by various user roles and make sure the system's architecture complies with user needs by examining this diagram.

We were able to visualise and improve the design of our system using these UML diagrams, making sure that it satisfies user needs while upholding a solid and effective architecture.

Justification of Selected Data Structures and Algorithms

Detail why particular data structures and algorithms were chosen for the project. Discuss the benefits of these choices in terms of efficiency, scalability, and applicability to the problem. Compare with alternative options briefly to justify why the selected methods are preferable.

Zensoft's Product Loan System has been designed with a focus on efficiency, speed, and scalability. Careful consideration has been given to the selection of data structures and algorithms to ensure the system's integrity and usability.

Explanation for the Choice of Data Structures
Hashmaps have been chosen for the management of customer and product data because they offer an average-case time complexity of O(1) for insertions, deletions, and search operations. (geeksforgeeks, 2020) Efficiency is crucial when dealing with real-time transactions, as any delays can have a big influence on consumer satisfaction. We used linear probing as our way to avoid collision for efficiency and maintain the data structure. (Dalal, Devroye and Malalla, 2023)

Dynamic arrays have been selected for the purpose of tracking products that have been rented to customers. These arrays offer immediate access to elements through indices, which is advantageous for tasks such as displaying all products borrowed by a client. Additionally, they can be adjusted in size, enabling the system to handle varying quantities of borrowed items without the requirement for periodic resizing of data structures.

Analysis of the Algorithms

1. Issuing products algorithm:

```
function issueProduct(customerID, productID)
if inventory.has(productID)
if customersMap.has(customerID)
customersMap[customerID].productsLoaned.add(productID)
inventory.remove(productID)
return "Product issued successfully"
else
return "Customer not found"
else
return "Product not available"
```

This algorithm ensures that a product is only issued if it is available and the customer is registered, hence maintaining the integrity of transactions.

2. Returning product algorithm

```
function returnProduct(customerID, productID)
if customersMap.has(customerID)
if customersMap[customerID].productsLoaned.contains(productID)
customersMap[customerID].productsLoaned.remove(productID)
inventory.add(productID)
return "Product returned successfully"
else
return "Product was not loaned out to this customer"
else
return "Customer not found"
```

When a product is returned, the algorithm updates the customer's loaned products list and the inventory, maintaining accurate records.

3. Calculating fine algorithm

```
function calculateFine(productID)
product = productsMap.get(productID)
if currentDate > product.dueDate
overdueDuration = currentDate - product.dueDate
fine = overdueDuration * dailyFineRate
return fine
else
return 0
```

This algorithm calculates fines based on how long a product has been overdue, ensuring that the system enforces the rules consistently.

Every algorithm has been meticulously crafted to prioritise time efficiency, ensuring prompt reactions to user actions. The pseudo code functions as a detailed plan outlining the logical operations that will be executed in the system's codebase. Additionally, it serves as a foundation for the development of more intricate features and for carrying out preliminary iterations of algorithm testing.

To summarise, Zensoft's design selections for the Product Loan System strike a harmonious equilibrium between technological performance and business objectives. This guarantees that the system will not only operate with optimal efficiency but also adhere to user expectations and operational demands.

3. Testing

The testing methodology used in the project was largely based on unit testing, and this was imperative to the effect of ensuring that functionality and reliability of each module in the program occurred independently. Definition of structuring of test cases and validation of behaviours of different independent modules, like Hash Table, Generation of IDs, and Validation functions, have been developed using Catch2 as a testing framework.

Unit Testing: The part of the test where each part of the program is chosen so that its testing is done in isolation from the others, verifying its own correctness independently from the rest. For example, tests have been created for the LinearProbingHash class to check and cover every scenario during insertion, update, retrieval, and even edge cases like getting data for keys which do not exist and table resizing to be well handled. These tests check that the table's hash structure retains its functionality against any condition that is prone to error or would require resizing of the table due to more load.

Other unit tests implemented include validation of the utility functions in the system. These tests include validating the member ID generation and inserting various input values properly, making sure the validation functions for names, emails, PINs, phone numbers are correctly failing with invalid formats and accepting valid ones.

These very broad testing measures would ensure that everything functions as intended and is able to effectively handle real-world input of data. Integration and System Testing: While it is primarily a focus towards unit testing, the structured arrangement of these tests also contributes to integration testing, especially for those features used in between themselves linked, or generating and validating IDs before insertion into data structures. Following the set of unit tests, this kind of implicit system testing, where all the parts are tested together as part of the full-blown application, would generally ensure that all parts work together seamlessly. The next testing phase will carry out test scenarios simulating real operational conditions to finally verify that the application perfectly meets all the given requirements.

The chosen testing approach is a good fit for the project, as it requires strong error-free components, which could be tested for the possibility of constant operation of the model in the production environment.

Test Case ID	Test Description	Expected Outcome	Actual Outcome	Pass/Fail Status
1	Generate ID for standard names	ID is generated for "John Doe" and it is not empty	ID generated correctly and is not empty	Pass
2	Generate ID for names with numbers	ID is generated for "John123" and it is not empty	ID generated correctly and is not empty	Pass
3	Generate ID for names with special characters	ID is generated for "@Jane$%^" and it is not empty	ID generated correctly and is not empty	Pass
4	Validate Names: Typical and correctly formatted names	"John Doe" returns true; "John123", "12345", "@John!" return false	Names validated according to criteria	Pass
5	Validate Email: Typical patterns and error cases	"email@example.com" returns true; invalid formats return false	Emails validated correctly	Pass
6	Validate PIN: Length and character correctness	"1234" returns true; "12345", "abcd", "12" return false	PIN numbers validated correctly	Pass
7	Validate Phone Number: Format correctness	"+44 7700 900123" returns true; incorrect formats return false	Phone numbers validated correctly	Pass
8	Validate Age: Numeric and range correctness	"30" returns true; "abc", "-1" return false	Ages validated correctly	Pass
9	Insert and retrieve values in LinearProbingHash	Correct values retrieved for keys "one", "two", "three"	Values retrieved as expected	Pass
10	Update existing key in LinearProbingHash	Updated value for key "one" is 100	Value updated and retrieved correctly	Pass
11	Handle non-existing key in LinearProbingHash	Retrieving value for non-existing key "two" throws error	Runtime error thrown as expected	Pass
12	Resize hash table in LinearProbingHash when load factor exceeds threshold	Hash table resizes correctly, and all items remain accessible and correctly valued	Hash table resized correctly; all values correct	Pass
13	Check if product ID is set and retrieved correctly	Product ID for "Slim Fit Shirt" is set to 1 and retrieved correctly	Product ID set and retrieved correctly	Pass
14	Check if PIN is set and retrieved correctly for Merchant	Merchant PIN for "John Doe" is set to "1234" and retrieved correctly	Merchant PIN set and retrieved correctly	Pass
15	Append Merchant Data to Existing CSV Files	New merchant data is appended to	Merchant data appended and verified	Pass

		"Merchants.csv" and can be read back correctly		
16	Append Customer Data to Existing CSV Files	New customer data is appended to "Customers.csv" and can be read back correctly	Customer data appended and verified	Pass
17	Load login menu successfully	Login menu is loaded without errors	Login menu loaded correctly	Pass
18	Print product confirmation for a specific product	Confirmation message for product "XYZ" is printed	Correct confirmation message displayed	Pass
23	Constructor and Date Retrieval	Date object initialized with "01-01-2020" correctly returns the same date	Date retrieved correctly	Pass
24	Set and Retrieve Due Date	Setting due date 30 days from "01-01-2020" results in "31-01-2020"	Due date set and retrieved correctly	Pass
25	Handle Invalid Date Format	Initializing with "invalid-date" and setting due date throws a runtime error	Runtime error thrown as expected	Pass
26	Empty Date String	Initializing with empty date and setting due date does not change due date	Due date remains empty as expected	Pass

4. Conclusion

In summary, the creation of Zensoft's Product Loan System has been characterised by careful preparation, astute choices, and significant learning experiences. As of right now, the system exhibits a sound design philosophy along with the successful application of tested and proven data structures and algorithms.

We faced several obstacles during the development process, all of which were essential to fortifying our plan. For instance, unforeseen collisions during the early development of the hashmap data structure briefly reduced system performance. Moreover, handling dynamic arrays for lent products came with challenges, especially in terms of removal. This process proves to be less effective than expected, leading to O(n) complexity in certain situations.

Moreover, problems occurred from the early technique of calculating fines' failure to take uncommon events like time zones and leap years into consideration. Memory leaks in early iterations of the system were caused by improper resource deallocation in the C++ environment, resulting in memory management difficulties. These problems showed our previous beliefs to be flawed. We needed to re-evaluate how much we relied on default hash methods for complicated keys, and we also needed to thoroughly evaluate our resource allocation and deallocation processes due to memory management concerns.

After carefully evaluating our work, it is evident that, even if the selected data structures and techniques made theoretical sense, there were practical problems that needed to be further investigated, like collision handling and memory management in C++. In similar future studies, we would give greater priority to and attention to:

- Carrying out thorough profiling in order to detect and resolve performance problems early in the development process.

- Developing custom hash functions for each of our unique requirements in order to reduce the amount of collisions.

- Putting into effect recommended approaches for memory management right away, particularly the automated resource management provided by C++'s smart pointers.

- Predicting edge scenarios early in the design process and providing robust error handling are better practices than managing them later.

The information gleaned from these observations will be extremely helpful for subsequent endeavours. The essay emphasises the value of thorough testing, taking into account real-world events at every level of the design process, and employing a flexible development approach that can adjust to unforeseen challenges. Our ability to plan strategically as well as technically has improved thanks to the Zensoft Product Loan System. Since we now have a greater understanding of system design and implementation, this will benefit future projects.

5.Reference

GeeksForGeeks (2015). *Open Addressing Collision Handling technique in Hashing*. [online] GeeksforGeeks. Available at: https://www.geeksforgeeks.org/open-addressing-collision-handling-technique-in-hashing/.

geeksforgeeks (2020). *Time complexities of different data structures*. [online] GeeksforGeeks. Available at: https://www.geeksforgeeks.org/time-complexities-of-different-data-structures/.

Dalal, K., Devroye, L. and Malalla, E. (2023). Two-Way Linear Probing Revisited. *Algorithms*, [online] 16(11), p.500. doi:https://doi.org/10.3390/a16110500.

Appendixes:

Figure 1: This is the class diagram for our system.

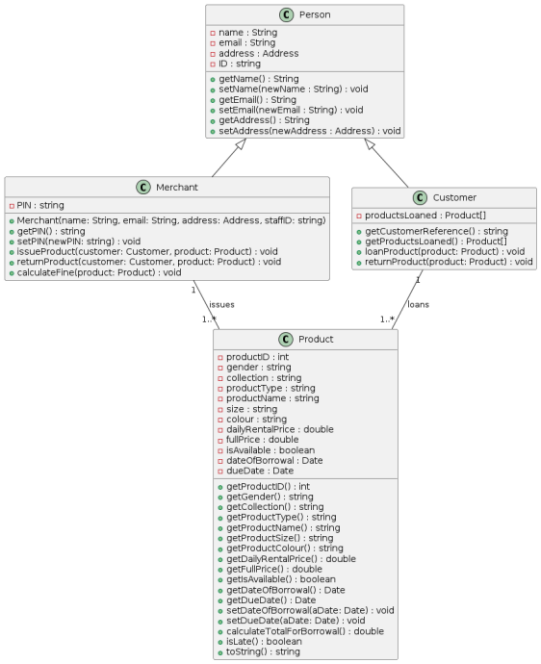

Figure 2: This is the use case illustration.

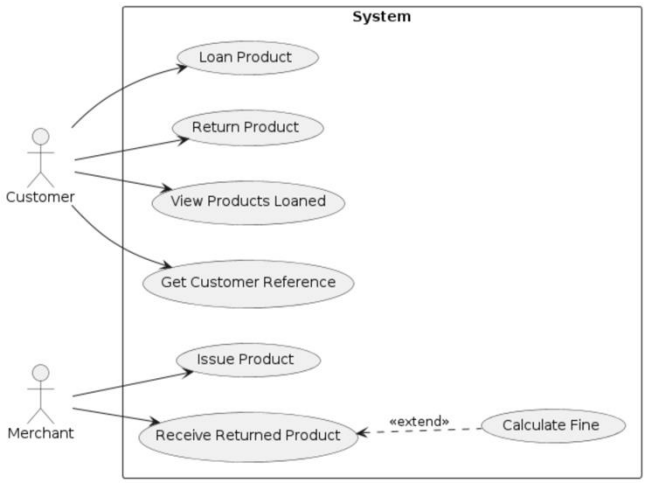

Figure 4: This is the activity diagram.