Video Renting System

Middlesex University Mauritius

Computer Science (System Engineering) BSc

CST2550 Software Engineering Management and Development

2024/25

Group Project Report

Submitted By:
Noor Patel – **Team Leader**
Doya Tejasweenee,
Keisha Bungchee,
Sameeha Bassa,
Benedict Kamazima

Student ID(s):
M00970749, M00992563, M01012479, M00990006, M00977722

Submission Date: 16th April 2025

1. Introduction

This project is designed to manage video rentals by allowing users to add, delete, search for, and rent videos. There are two types of users: administrators and regular users. Administrators have the authority to add and delete videos, while regular users can search for videos and rent them. Rented videos are displayed in orange on the user's profile page, while expired videos are shown in light blue. The project aims to demonstrate the use of hash tables, as well as providing an intuitive user interface. Any changes made in the hash table during the session are saved to the SQL database upon logout.

Report Layout Description

The report is structured to provide a comprehensive understanding of the **Video Rental System**'s core functionalities, data handling mechanisms, and testing procedures. The layout is divided into the following sections:

1. **Introduction**: This section outlines the purpose of the project, which involves managing video rentals. The system features two types of users: **administrators**, who have the authority to add and delete videos, and **regular users**, who can search for and rent videos. It also highlights how the system integrates a **relational database** with **user authentication** and uses a **Custom Hash Table** to temporarily store data during runtime. This section concludes by describing how changes made to the hash table are saved to the SQL database upon user logout.
2. **Data Algorithms**: This section includes:
    - **Justification of Selected Data Structures and Algorithms**: An explanation of the chosen data structures, specifically **SQL** and **Custom Hash Table**, to handle video data and rental operations efficiently. The rationale behind the selection is based on their ability to provide fast access, insertion, and deletion operations, ensuring system scalability.
    - **Analysis of Data Structure**: Detailed analysis of the core operations, such as adding, deleting, and searching for videos in the hash table. It includes explanations of how each functionality is implemented and discusses the **time complexity** of the operations.
3. **Testing**: This section is divided into two parts:
    - **Database Testing**: Tests the connection to the SQL database and verifies that users are correctly inserted into the database. It also checks if the user data can be retrieved successfully. The table includes details about the test cases, expected results, and actual results.
    - **Hash Table Testing**: Tests various operations on the **Custom Hash Table**, including adding, removing, searching for, and checking the existence of keys. Each test is documented with input data, expected outcomes, and the actual results.

2. Data algorithms

2.1. Justification of Selected Data Structures and Algorithms

The system utilizes a combination of SQL and a Custom Hash Table to efficiently handle video data and rental operations. The selection of these data structures is driven by the need for fast access, insertion, and deletion operations.

- **SQL**: SQL is used for persistent storage of video data, ensuring reliable data management. SQL databases are optimal for structured data with relationships, allowing efficient querying, updating, and retrieval of records **(GeeksforGeeks, 2025).**
- **Custom Hash Table**: A Custom Hash Table is employed to store video data temporarily during runtime. Hash tables offer **constant time complexity (O (1))** for operations such as adding, searching, and updating records, making them well-suited for large datasets with frequent lookups (**GeeksforGeeks, 2021).**
- **Efficiency and Flexibility**: The combination of SQL for persistent storage and the Custom Hash Table for fast, in-memory access balances efficiency and flexibility. This approach allows the system to work effectively.

Hash tables may be used as in-memory data structures. Hash tables may also be adopted for use with persistent data structures; database indexes commonly use disk-based data structures based on hash tables. **(LibreTexts, n.d.).**

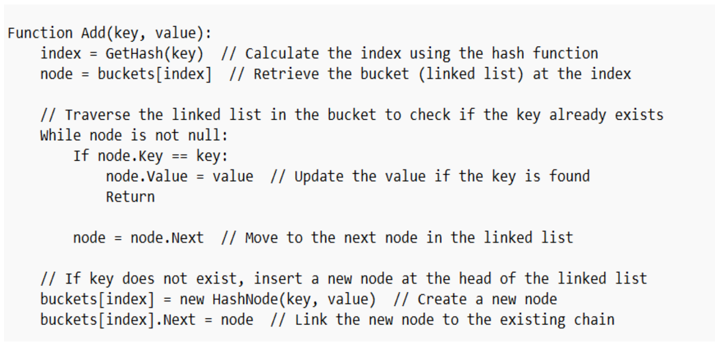
2.2. Analysis of Data Structure

2.2.1 Add Functionality

Explanation: The Add function inserts a new key-value pair into the hash table. If the key already exists, it updates the value. It first calculates the index for the key using the hash function, then checks the corresponding bucket (linked list) to see if the key already exists. If the key is found, it updates the value. Otherwise, it adds a new node at the head of the linked list for that index**. (ScienceDirect, 2012).**

Importance: This method is crucial for maintaining the hash table's integrity, ensuring that data can be efficiently added and updated. The operation is generally **O (1)**, meaning it is fast.

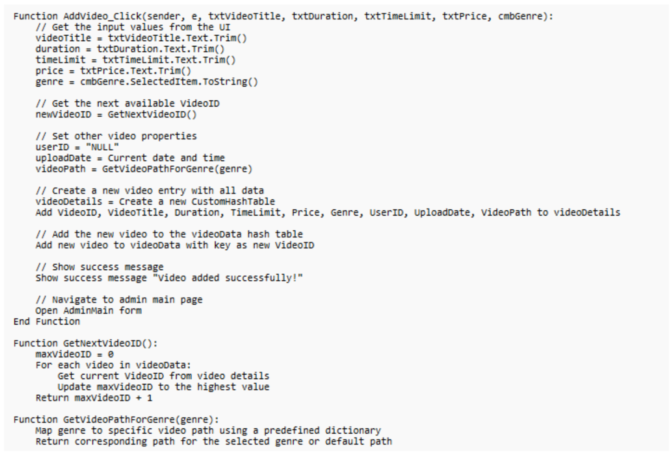
In general, the **time complexity** for insertions is **O (1)**, but it may be higher if hash collisions are frequent **(FreeCodeCamp, n.d.).**



```
Function Add(key, value):
    index = GetHash(key)  // Calculate the index using the hash function
    node = buckets[index]  // Retrieve the bucket (linked list) at the index

    // Traverse the linked list in the bucket to check if the key already exists
    While node is not null:
        If node.Key == key:
            node.Value = value  // Update the value if the key is found
            Return

        node = node.Next  // Move to the next node in the linked list

    // If key does not exist, insert a new node at the head of the linked list
    buckets[index] = new HashNode(key, value)  // Create a new node
    buckets[index].Next = node  // Link the new node to the existing chain
```

Figure 1 Add Functionality (In Hash Table)

The "add" function has been used in various scenarios, including adding videos from SQL to the Hashtable and, more importantly, adding new videos as seen in AddVideo.cs.



```
Function AddVideo_Click(sender, e, txtVideoTitle, txtDuration, txtTimeLimit, txtPrice, cmbGenre):
    // Get the input values from the UI
    videoTitle = txtVideoTitle.Text.Trim()
    duration = txtDuration.Text.Trim()
    timeLimit = txtTimeLimit.Text.Trim()
    price = txtPrice.Text.Trim()
    genre = cmbGenre.SelectedItem.ToString()

    // Get the next available VideoID
    newVideoID = GetNextVideoID()

    // Set other video properties
    userID = "NULL"
    uploadDate = Current date and time
    videoPath = GetVideoPathForGenre(genre)

    // Create a new video entry with all data
    videoDetails = Create a new CustomHashTable
    Add VideoID, VideoTitle, Duration, TimeLimit, Price, Genre, UserID, UploadDate, VideoPath to videoDetails

    // Add the new video to the videoData hash table
    Add new video to videoData with key as new VideoID

    // Show success message
    Show success message "Video added successfully!"

    // Navigate to admin main page
    Open AdminMain form
End Function

Function GetNextVideoID():
    maxVideoID = 0
    For each video in videoData:
        Get current VideoID from video details
        Update maxVideoID to the highest value
    Return maxVideoID + 1

Function GetVideoPathForGenre(genre):
    Map genre to specific video path using a predefined dictionary
    Return corresponding path for the selected genre or default path
```

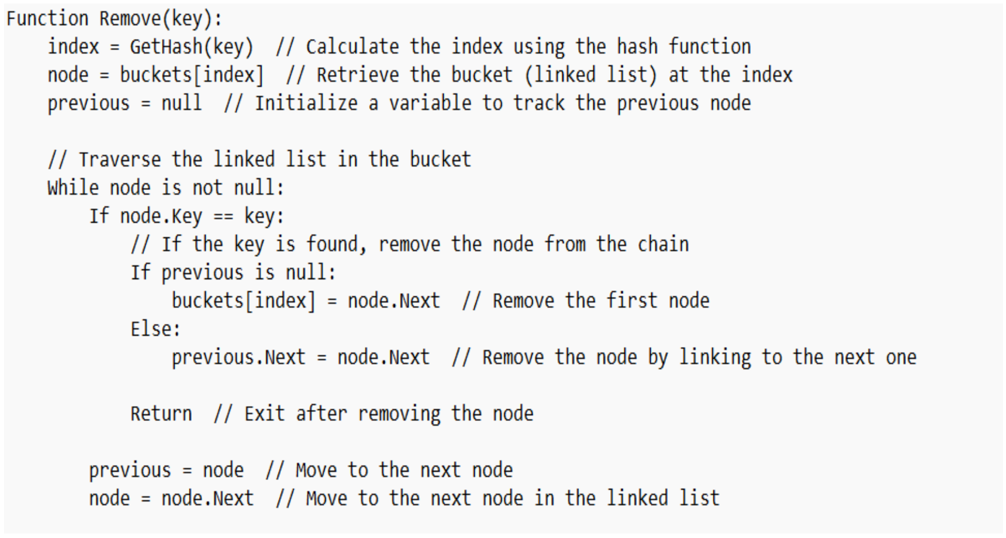
Figure 2 Add Functionality (Adding new videos to the main table)

2.2.2 Delete Functionality

Explanation: The Remove function deletes a key-value pair from the hash table using the key. The function first calculates the index of the key, then traverses the linked list at that index to find the node with the matching key. If the node is found, it is removed by updating the pointers in the linked list. **ScienceDirect, 2012)**

Importance: Removing key-value pairs from the hash table is essential for managing data, particularly when a video is deleted.

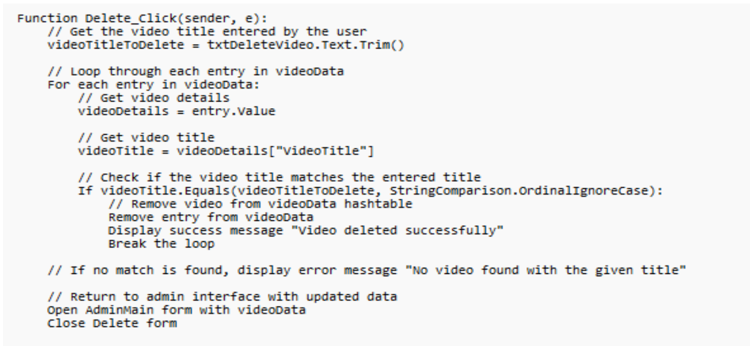
Time Complexity: O (n) in the worst case**, O (1)** on average **(FreeCodeCamp, n.d.).**



```
Function Remove(key):
    index = GetHash(key)  // Calculate the index using the hash function
    node = buckets[index]  // Retrieve the bucket (linked list) at the index
    previous = null  // Initialize a variable to track the previous node

    // Traverse the linked list in the bucket
    While node is not null:
        If node.Key == key:
            // If the key is found, remove the node from the chain
            If previous is null:
                buckets[index] = node.Next  // Remove the first node
            Else:
                previous.Next = node.Next  // Remove the node by linking to the next one

            Return  // Exit after removing the node

        previous = node  // Move to the next node
        node = node.Next  // Move to the next node in the linked list
```

Figure 3 Delete Functionality (In CustomHashTable)

The delete functionality is used only in deleteVideo.cs, where the user enters a video title, and if the video is found, it will be deleted.



```
Function Delete_Click(sender, e):
    // Get the video title entered by the user
    videoTitleToDelete = txtDeleteVideo.Text.Trim()

    // Loop through each entry in videoData
    For each entry in videoData:
        // Get video details
        videoDetails = entry.Value

        // Get video title
        videoTitle = videoDetails["VideoTitle"]

        // Check if the video title matches the entered title
        If videoTitle.Equals(videoTitleToDelete, StringComparison.OrdinalIgnoreCase):
            // Remove video from videoData hashtable
            Remove entry from videoData
            Display success message "Video deleted successfully"
            Break the loop

    // If no match is found, display error message "No video found with the given title"

    // Return to admin interface with updated data
    Open AdminMain form with videoData
    Close Delete form
```

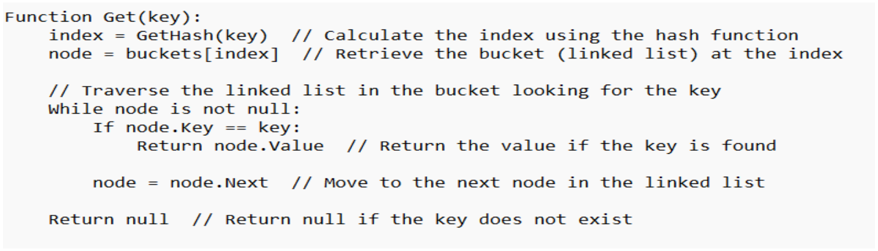
Figure 4 Delete Functionality (Deleting existing videos to the main table)

2.2.3 Search Functionality

Explanation: The Get function retrieves the value associated with a given key. It computes the index using the hash function, then searches the linked list at that index for a node with the matching key. If found, it returns the value; otherwise, it returns null. (**ScienceDirect, 2012)**

Importance: This function is critical for looking up values in the hash table, and its performance is generally **O (1)**. It's essential for quickly retrieving video details, rental information, or any other data stored in the hash table.

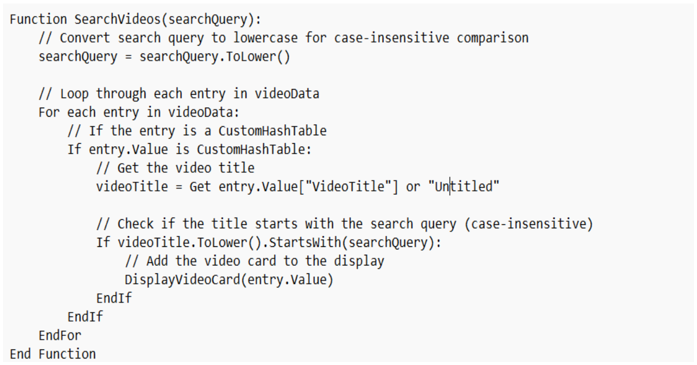
Time Complexity: O(n) in the worst case**, O (1)** on average **(Wikipedia, 2021).**



```
Function Get(key):
    index = GetHash(key)  // Calculate the index using the hash function
    node = buckets[index]  // Retrieve the bucket (linked list) at the index

    // Traverse the linked list in the bucket looking for the key
    While node is not null:
        If node.Key == key:
            Return node.Value  // Return the value if the key is found

        node = node.Next  // Move to the next node in the linked list

    Return null  // Return null if the key does not exist
```

Figure 5 Search Functionality (In CustomHashTable)

The search functionality in the project filters videos by title on the main page, displaying results that match the beginning of the title entered by the user.

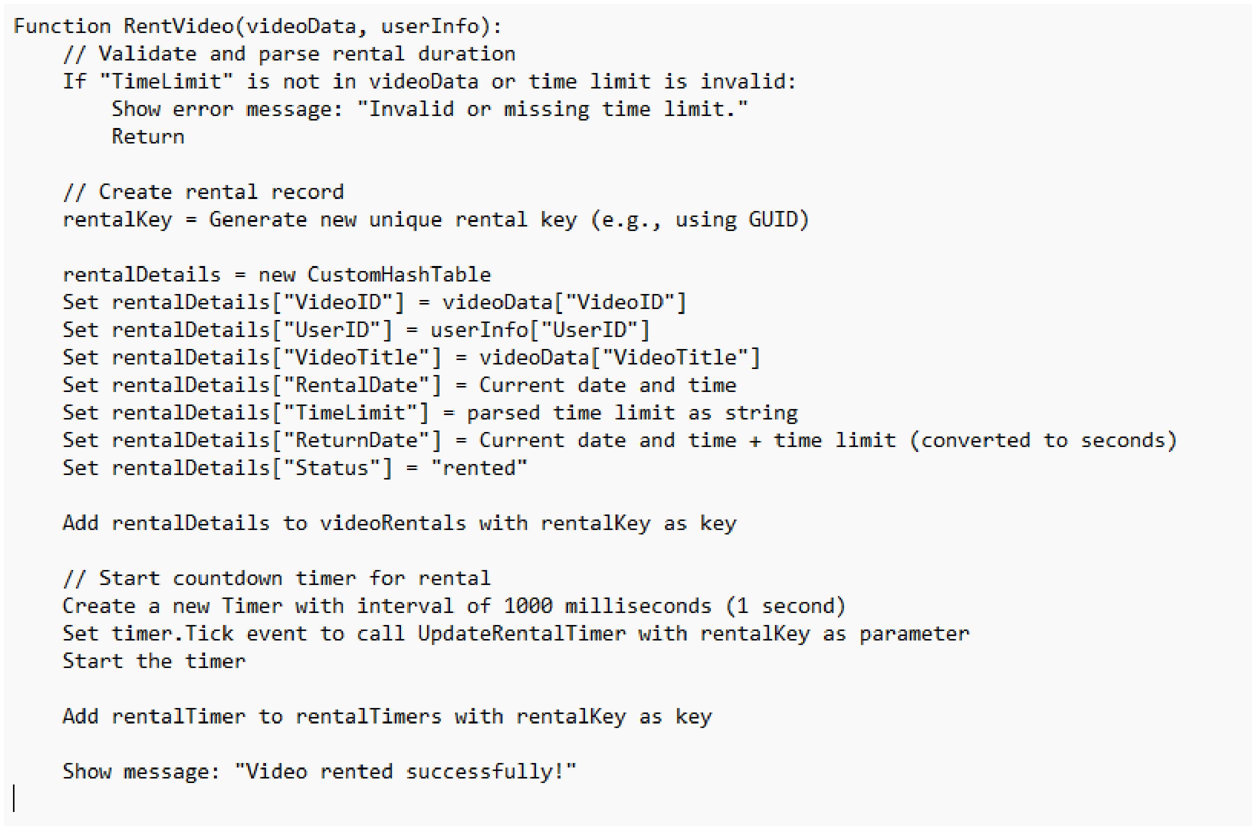


```
Function SearchVideos(searchQuery):
    // Convert search query to lowercase for case-insensitive comparison
    searchQuery = searchQuery.ToLower()

    // Loop through each entry in videoData
    For each entry in videoData:
        // If the entry is a CustomHashTable
        If entry.Value is CustomHashTable:
            // Get the video title
            videoTitle = Get entry.Value["VideoTitle"] or "Untitled"

            // Check if the title starts with the search query (case-insensitive)
            If videoTitle.ToLower().StartsWith(searchQuery):
                // Add the video card to the display
                DisplayVideoCard(entry.Value)
            EndIf
        EndIf
    EndFor
End Function
```

Figure 6 Search Functionality (Searching for specific videos to the main table)

2.2.4 Rent Option

The RentVideo function validates the rental duration by checking if the TimeLimit key exists in the video data and if it can be parsed correctly; if not, it shows an error message and exits. If valid, it creates a unique rental record containing the VideoID, UserID, VideoTitle, RentalDate, TimeLimit, ReturnDate, and Status, and adds this record to the videoRentals hash table. A countdown timer is then started with a 1-second interval, updating the rental status or time remaining via the UpdateRentalTimer function, and the timer is stored in the rentalTimers hash table. Once the rental process is complete, the system displays a success message to the user.



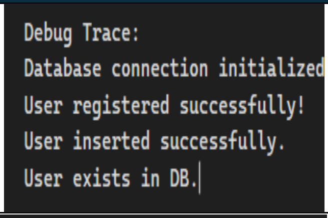
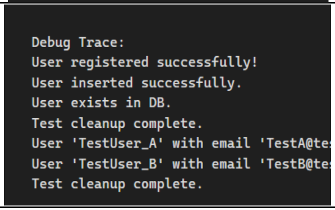
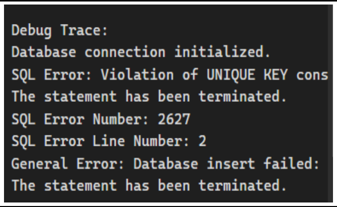
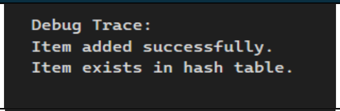
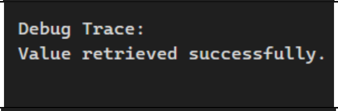
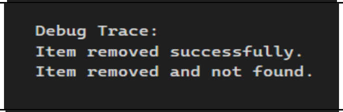
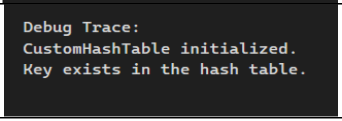
```
Function RentVideo(videoData, userInfo):
    // Validate and parse rental duration
    If "TimeLimit" is not in videoData or time limit is invalid:
        Show error message: "Invalid or missing time limit."
        Return

    // Create rental record
    rentalKey = Generate new unique rental key (e.g., using GUID)

    rentalDetails = new CustomHashTable
    Set rentalDetails["VideoID"] = videoData["VideoID"]
    Set rentalDetails["UserID"] = userInfo["UserID"]
    Set rentalDetails["VideoTitle"] = videoData["VideoTitle"]
    Set rentalDetails["RentalDate"] = Current date and time
    Set rentalDetails["TimeLimit"] = parsed time limit as string
    Set rentalDetails["ReturnDate"] = Current date and time + time limit (converted to seconds)
    Set rentalDetails["Status"] = "rented"

    Add rentalDetails to videoRentals with rentalKey as key

    // Start countdown timer for rental
    Create a new Timer with interval of 1000 milliseconds (1 second)
    Set timer.Tick event to call UpdateRentalTimer with rentalKey as parameter
    Start the timer

    Add rentalTimer to rentalTimers with rentalKey as key

    Show message: "Video rented successfully!"
```

Figure 7 Rent Option (In main page)

Overall Time Complexity is O (1).

3. Testing

The testing approach for the **Video Renting System** involves using **MS Testing** to verify core functionalities such as **Add**, **Delete**, **Search**, and **Insert User** in the database. Unit tests were created to ensure that the **Add**, **Delete**, and **Search** functions in the **Custom Hash Table** work as expected, verifying that items are correctly added, removed, and searched for based on key values. Additionally, integration tests were implemented to validate that **Insert User** operations successfully add user information to the database, ensuring the data is correctly stored and retrievable. Test cases were designed to check both positive and negative scenarios, including valid and invalid inputs, to confirm the robustness of these operations. All tests were automated using **MS Testing**, and the results were analyzed to ensure the system behaves as expected under different conditions, providing confidence in the integrity and performance of the application.

Database Testing					
Test Case ID	Description	Input Data	Expected Result	Actual Result	Status
TC1	Test if a new user is inserted into the database (valid input)	Username: TestUser_A, Email: TestA@test.com, Password: TestPassword123	Users should be inserted into the database.	 Debug Trace: Database connection initialized User registered successfully! User inserted successfully. User exists in DB.	**Passed**
TC2	Test insertion of another user	Username: TestUser_B, Email: TestB@test.com, Password: AnotherSecretXYZ	Users should be inserted into the database.	 Debug Trace: User registered successfully! User inserted successfully. User exists in DB. Test cleanup complete. User 'TestUser_A' with email 'TestA@te User 'TestUser_B' with email 'TestB@te Test cleanup complete.	**Passed**
TC3	Test for the same username	Username: TestUser_A, Email: TestA@test.com, Password: NewPassword123	Database error	 Debug Trace: Database connection initialized. SQL Error: Violation of UNIQUE KEY cons The statement has been terminated. SQL Error Number: 2627 SQL Error Line Number: 2 General Error: Database insert failed: The statement has been terminated.	**Passed**
Hash table Testing					
TC1	Test adding an item to the hash table	Key: TestUser, Value: Name = "John", Age = 25}	Items should be added successfully to the hash table.	 Debug Trace: Item added successfully. Item exists in hash table.	**Passed**
TC2	Test searching for an item on the hash table	Key: TestUser	Should return the correct value {Name = "John", Age = 25}	 Debug Trace: Value retrieved successfully.	**Passed**
TC3	Test removing an item from the hash table	Key: TestUser	Item should be removed and no longer found on the hash table	 Debug Trace: Item removed successfully. Item removed and not found.	**Passed**
TC4	Test checking if a key exists in the hash table	Key: TestUser	Should return true indicating the key exists in the table	 Debug Trace: CustomHashTable initialized. Key exists in the hash table.	**Passed**

4. Conclusion

The **Video Renting System** project successfully achieved the goal of managing video rentals, providing a platform for both administrators and regular users. Administrators can add and delete videos, while regular users can search for and rent videos. A **Custom Hash Table** was used to handle video data and rental operations in memory, ensuring fast access and efficient data management, while the **SQL database** served as permanent storage for the system. The project demonstrated the integration of in-memory data structures with a relational database, as well as the implementation of essential operations such as adding, deleting, and renting videos.

Throughout the development of the project, the system was tested rigorously using **MS Testing** to ensure the core functionalities of adding, deleting, searching, and renting were working as expected. The test results confirmed the successful execution of these operations, and the ability to handle both valid and invalid inputs. However, during testing, the issue of **duplicate usernames** was highlighted, and the test was successful in verifying that the **unique constraint** was correctly enforced by the database.

Limitations and Critical Reflection:

One limitation is that handling **video expiration** or managing rental **time limits** may require more comprehensive testing to ensure the system could handle all scenarios under different conditions.

In terms of **user experience**, the **UI responsiveness** could have been improved. The current system did not offer immediate feedback when a video was rented or deleted, which could confuse users. Additionally, the **search functionality** could be enhanced with more advanced filters (e.g. Duration) to improve the search experience for users.

Future Approach:

If approached with a similar task in the future, several changes would be made to improve the system. First, **optimizing the hash table** and considering **alternative data structures** (e.g., **balanced trees**) would be essential for handling larger datasets more efficiently.

Additionally, **automating the rental expiration process** and integrating **automated notifications** for users when a rental expires or is approaching expiration would improve the user experience. **Enhanced search functionality** with additional filters and more comprehensive error handling would also be prioritized.

By addressing these limitations, the system could be made more efficient, and user-friendly, and improve its overall performance

Referencing

1. **FreeCodeCamp**: What is a Hash Map? Time Complexity and Two Sum Example. Available at: https://www.freecodecamp.org/news/what-is-a-hash-map/?utm_source=chatgpt.com [Accessed 14 Apr. 2025].
2. **GeeksforGeeks** (2021). *Hash Table Data Structure*. Available at: https://www.geeksforgeeks.org/hash-table-data-structure/ [Accessed 14 Apr. 2025].
3. **ScienceDirect** (2012). *Hash Table Overview*. Retrieved from https://www.sciencedirect.com/topics/computer-science/hash-table [Accessed 14 Apr. 2025].
4. **W3Schools**. (n.d.). *SQL Tutorial*. Available at: https://www.w3schools.com/sql/ [Accessed 14 Apr. 2025].
5. **Wikipedia** (2021). *Hash Table*. Available at: https://en.wikipedia.org/wiki/Hash_table [Accessed 14 Apr. 2025].
6. **LibreTexts**. (n.d.). *7.1: Time complexity and common uses of hash tables*. Available at: https://eng.libretexts.org/Courses/Folsom_Lake_College/CISP_430%3A_Data_Structures_%28Aljuboori%29/07%3A_Hash_Tables/7.01%3A_Time_complexity_and_common_uses_of_hash_tables [Accessed 14 Apr. 2025].