DEPARTMENT OF COMPUTER & INFORMATION SYSTEMS ENGINEERING
BACHELORS IN COMPUTER SYSTEMS ENGINEERING
Course Code: CS-218

Course Title: Data Structures & Algorithms

Complex Engineering Problem

SE Batch 2023, Fall Semester 2024
Grading Rubric
TERM PROJECT

Group Members:

Student No.	Name	Roll No.
S1	KHADIJA SEHAR	CS-23014
S2	SUMBAL ZEHRA	CS-23024
S3	AAMNAH AATIF	CS-23028
S4	ABEER KHAN	CS-23301

CRITERIA AND SCALES				Marks Obtained			
				S1	S2	S3	S4
Criterion 1: Has the student provided the appropriate design of LRU data structure?							
0	1	2	-				
The chosen design is too simple	The design is fit to be chosen for a class project	The choice is different and impressive.	-				
Criterion 2: How good is the programming implementation?							
0	1	2	3				
The project could not be implemented	The project has been implemented partially.	The project has been implemented completely but can be improved.	The project has been implemented completely and impressively				
Criterion 3: How well written is the report?							
0	1	2	-				
The submitted report is unfit to be graded	The report is partially acceptable	The report is complete and concise					
			Total Marks:				

PROJECT REPORT

PROBLEM DESCRIPTION:

The goal of the project was to implement a Least Recently Used (LRU) Cache in Python. An LRU Cache is a data structure that holds a fixed number of items, removing the least recently used item when the capacity is exceeded. Additionally, the cache was extended with a Time-to- Live (TTL) feature, ensuring that items expire after a specified duration. This project also required functionality to collect and display cache performance metrics, such as hits, misses, evictions, and miss rates, using Python libraries. The LRU Cache should maintain key-value pairs and efficiently manage the cache based on the constraints of limited capacity.

FLOW OF PROJECT:

The flow of the project involves the following steps, highlighting how the LRU Cache works and the key aspects of its design:

1. Cache Initialization

The project begins with the initialization of the LRU Cache class:

The constructor (_init__) accepts two parameters:

- capacity: Defines the maximum number of items the cache can hold.
- ttl: The time-to-live for each cache entry in seconds.

It initializes three main data structures:

- cache: A dictionary to store the key-value pairs.
- order: A list to track the usage order of keys, ensuring efficient eviction of the least recently used entry.
- timestamps: A dictionary to store the time each key was added or updated for TTL enforcement.

Additional attributes like hits, misses, total_accesses, and evictions are initialized to track cache performance.

2. Handling Cache Insertion (put Method):

The put method manages adding or updating entries in the cache:

- **Key Exists in Cache:**

If the key is already in the cache, it's a cache hit. The value is updated, and the key is moved to the most recent position in the order list. The timestamps dictionary is updated with the current time to reset the TTL for this key.

- **Key Does Not Exist in Cache:**

If the cache is full, the least recently used key is evicted using the _evict_if_needed method. The new key-value pair is added to the cache dictionary, appended to the order list, and its timestamp is recorded in the timestamps dictionary. Thís method tracks hits, misses, and evictions to provide accurate cache statistics.

3. Retrieving Cache Entries (get Method)

The get method allows retrieval of items from the cache:

Before accessing the cache, the _remove_expired_keys method is called to remove entries that have exceeded their TTL.

- **Key Exists in Cache:**

The key is moved to the most recent position in the order list to reflect its usage. A cache hit is recorded, and the value is returned.

- **Key Does Not Exist in Cache:**

A cache miss is recorded, and the method returns None.

4. Expiration of Entries:

- The _remove_expired_keys method is periodically invoked to ensure that expired entries (those whose TTL has elapsed) are removed from the cache.
- Expired keys are identified by comparing the current time (time.time()) with their recorded timestamp in timestamps.
- Expired keys are removed from cache, order, and timestamps. Evictions are incremented for these removals.

5. Eviction Policy:

The _evict_if_needed method ensures that the cache adheres to its capacity:

- When the cache is full, the least recently used (LRU) key is evicted. This is determined as the first key in the order list.
- The LRU key is removed from cache, timestamps, and order. The eviction count is incremented.

6. Cache Performance Metrics:

The get_statistics method provides an overview of cache operations:

- Hits: Number of successful retrievals.
- Misses: Number of failed retrievals.
- Evictions: Total items evicted, including expired entries and those removed due to capacity constraints.

The get_miss_rate method calculates the percentage of cache accesses that resulted in misses.

7. Visualization of Results:

- The display_data function uses Matplotlib to create a bar chart of cache performance metrics, including hits, misses, evictions, and miss rates.

- This helps in understanding the efficiency of the cache and identifying potential areas for improvement.

8. Testing and Usage:

The test script simulates a series of put and get operations to demonstrate the cache functionality:

- Adding items to the cache.
- Accessing existing items.
- Handling cache misses and evictions.
- Collecting statistics for analysis.

The workflow ensures that all aspects of the cache, including TTL, eviction policy, and performance tracking, are thoroughly tested.

MOST CHALLENGING PART OF THE PROJECT:

The most challenging aspect was implementing dynamic expiration of cache entries based on the TTL. Managing timestamps for each entry and ensuring expired entries were removed without compromising the cache's efficiency required careful thought. Another complexity was maintaining consistency between the cache, order, and timestamps dictionaries to avoid errors such as stale references.

KEY LEARNINGS IN PYTHON:

1. Handling Expiration with Timestamps:

Using time.time() to store and compare timestamps for TTL enforcement was a valuable skill learned during this project.

2. Efficient List Management:

Using list.remove() and list.append() to maintain the order of access while ensuring the least recently used key is easily identified.

3. Data Visualization with Matplotlib:

Creating bar charts and annotating them with values for better representation of statistical data.

4. Python Object-Oriented Programming (OOP):

Strengthened understanding of OOP concepts like encapsulation, class methods, and modular design.

5. Using Streamlit:

Integrating data processing and visualization seamlessly, creating interactive dashboards and deploying user-friendly web apps for real-time insight.

6. Error Handling:

Prevented issues like division by zero when calculating miss rates and ensured smooth operation of the cache.

TIME AND SPACE COMPLEXITY:

Time Complexity: O(n)
Space Complexity: O(n)

FUTURE IMPROVEMENTS:

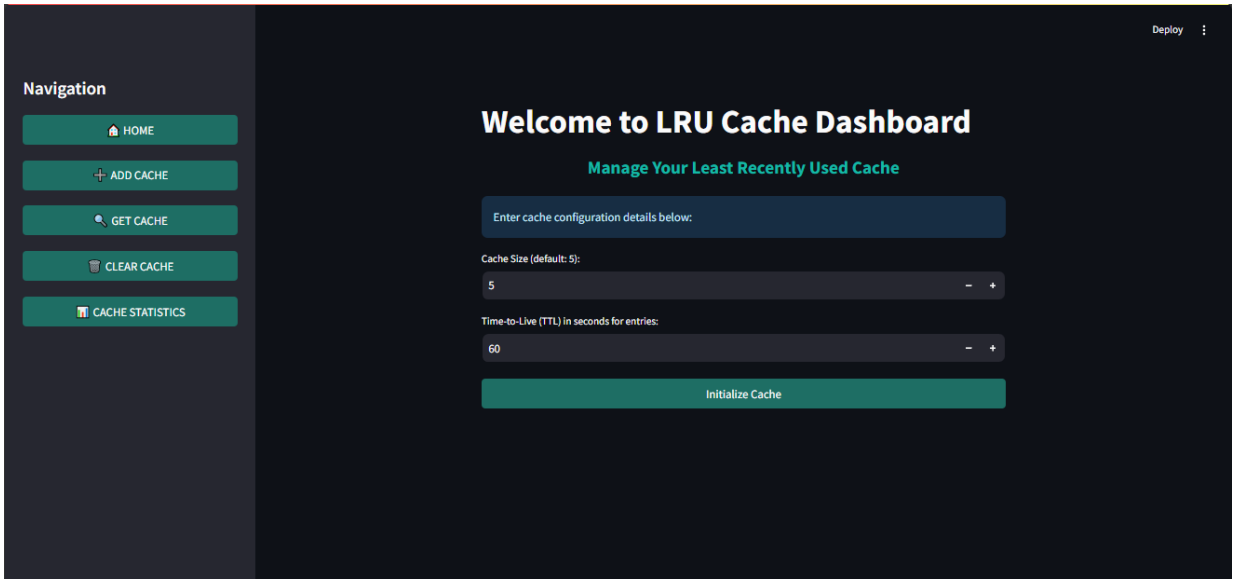
Implement proper error handling for invalid input keys. Currently out of range keys are defaulted to zero, but they should trigger an error message instead.

CONCLUSION:

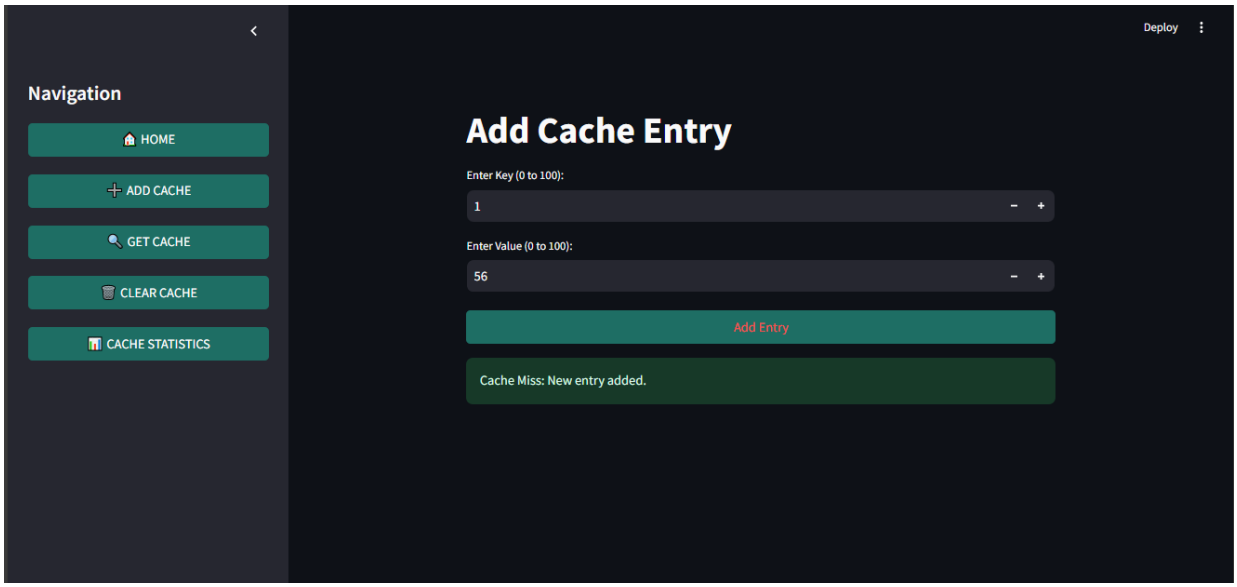
This project successfully implemented an LRU Cache with TTL functionality and a performance visualization module. The modular design, combined with efficient data structures, ensured that the cache met performance expectations. Additionally, challenges like expiration handling and visualization enhanced problem-solving skills and provided a deeper understanding of Python programming concepts.

TEST RUN:

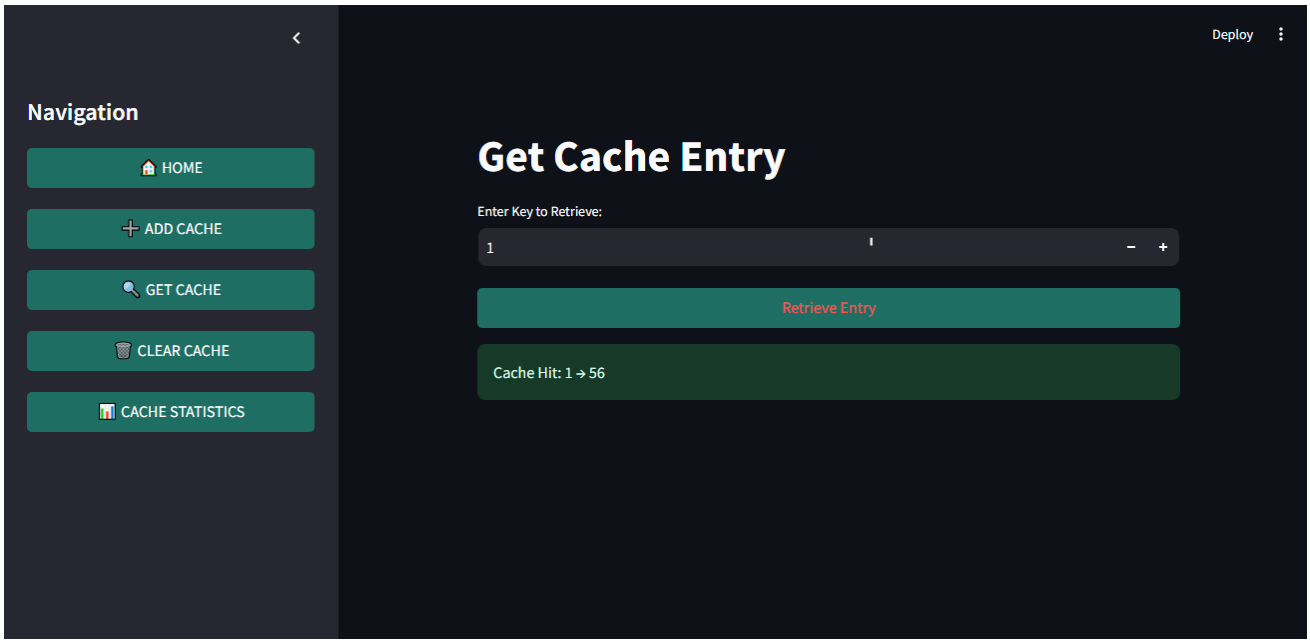
FOR HOME PAGE:



FOR ADD CACHE:



FOR GET CACHE:



FOR CACHE STATISTICS: