This document details what data structures I would be implementing in my CRUD app and outlines how I would be doing so.

NOTE: In my actual CRUD application, I only interacted with the user and portfolio data structures.

Redis Data Structures Implementation

1. Hashes for User Data Caching

- Structure Used: Redis Hash
- Key: user:{userId}
- Fields: First name, last name, email, last login
- Purpose: Hashes are ideal for storing object-like data with multiple fields. Using a hash for each user allows for efficient storage and retrieval of user details in a structured format.
- Operations:
  - Set Data: HSET user:{userId} first_name "John" last_name "Gomez" email "john.gomez@gmail.com" last_login "2024-04-15"
  - Get Data: HGETALL user:{userId}
  - Delete Data: DEL user:{userId}
- Usage: This structure is used to cache user data after it's fetched from the MongoDB database to reduce read latency and database load.

2. Strings for Portfolio Data Caching

- Structure Used: Redis String (with JSON content)
- Key: portfolio:{userId}
- Value: JSON string containing portfolio details
- Purpose: The string type is used to store serialized JSON data.
- Operations:
  - Set Data: SET portfolio:{userId} "{json_data}" EX 3600
  - Get Data: GET portfolio:{userId}
  - Delete Data: DEL portfolio:{userId}
- Usage: Caches portfolio details associated with a user, with an expiration set to maintain freshness of the data.

3. Sorted Sets for Real-Time Property Value Leaderboard

- Structure Used: Redis Sorted Set
- Key: property_leaderboard
- Values: Property IDs as members with their values as scores
- Purpose: Sorted sets automatically maintain entries sorted by score, which is perfect for leaderboards where scores/values can frequently change.
- Operations:

- Add/Update Score: ZADD property_leaderboard {value} {propertyId}
    - Retrieve Top Entries: ZREVRANGE property_leaderboard 0 -1 WITHSCORES
    - Remove Entry: ZREM property_leaderboard {propertyId}
- Usage: This structure dynamically ranks properties by their value, allowing for real-time updates and retrievals of ranking positions.

4. Lists for Tenant Interaction Logs

- Structure Used: Redis List
- Key: tenant_interactions:{tenantId}
- Purpose: Lists provide a simple way to log events in order as they happen, which is useful for maintaining a history of tenant interactions.
- Operations:
    - Log Interaction: LPUSH tenant_interactions:{tenantId} "interaction details"
    - Retrieve Interactions: LRANGE tenant_interactions:{tenantId} 0 -1
    - Delete Log: DEL tenant_interactions:{tenantId}
- Usage: Used to log and retrieve tenant interactions, providing a temporal record of events.

5. Hashes for Financial Report Summaries

- Structure Used: Redis Hash
- Key: financial_report:{reportId}
- Fields: Income, expenses, summary
- Purpose: Hashes are efficient for storing and accessing multiple related data fields of a financial report.
- Operations:
    - Cache Data: HSET financial_report:{reportId} income 50000 expenses 30000 summary "Profitable"
    - Retrieve Data: HGETALL financial_report:{reportId}
    - Clear Data: DEL financial_report:{reportId}
- Usage: Enables quick access to pre-calculated financial summaries, reducing the need for repeated costly calculations.