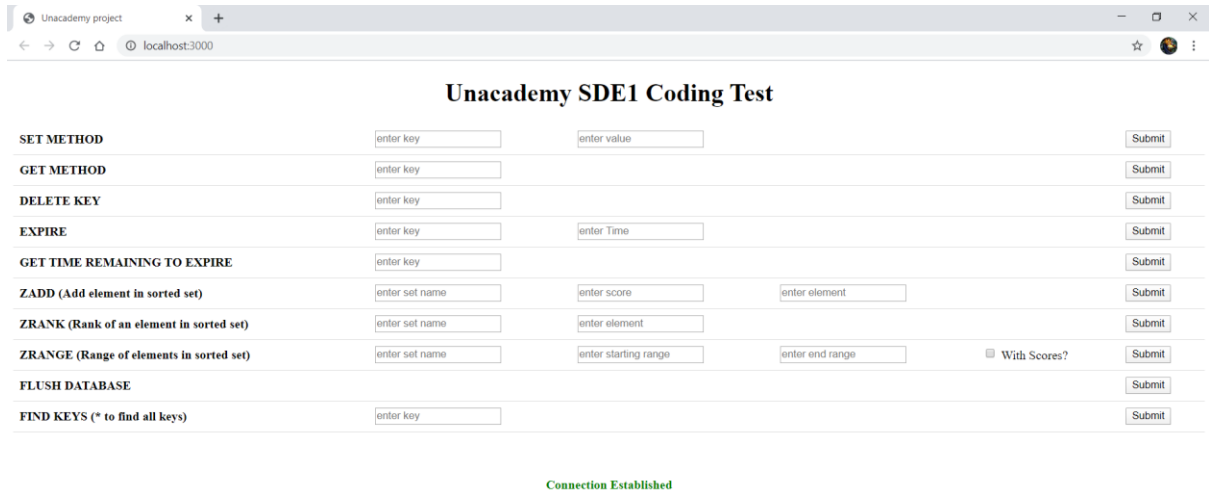


Name: Shivang Agarwal

Email: shivang2006@gmail.com

Phone: 8989931015

UNACADEMY CODING TEST REPORT

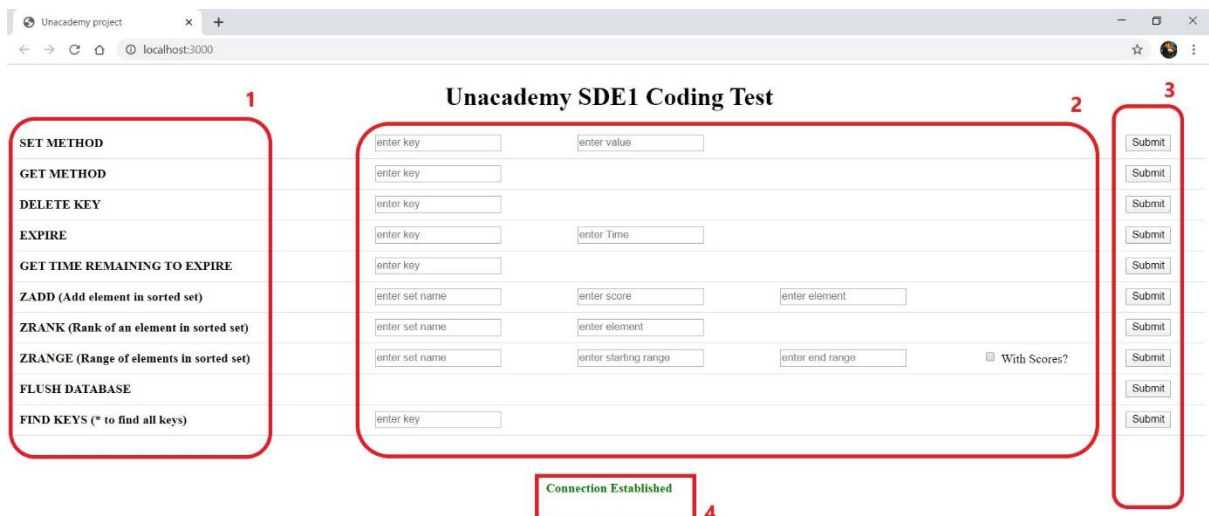


The screenshot shows a web browser window with the title "Unacademy project" and the address bar displaying "localhost:3000". The main heading is "Unacademy SDE1 Coding Test". Below the heading is a table with 10 rows, each representing a Redis command. Each row has input fields for the command's parameters and a "Submit" button. The commands are: SET METHOD, GET METHOD, DELETE KEY, EXPIRE, GET TIME REMAINING TO EXPIRE, ZADD (Add element in sorted set), ZRANK (Rank of an element in sorted set), ZRANGE (Range of elements in sorted set), FLUSH DATABASE, and FIND KEYS (* to find all keys). The ZRANGE row includes a checkbox labeled "With Scores?". Below the table, the text "Connection Established" is displayed in green.

Command	Input Fields	Action
SET METHOD	enter key, enter value	Submit
GET METHOD	enter key	Submit
DELETE KEY	enter key	Submit
EXPIRE	enter key, enter Time	Submit
GET TIME REMAINING TO EXPIRE	enter key	Submit
ZADD (Add element in sorted set)	enter set name, enter score, enter element	Submit
ZRANK (Rank of an element in sorted set)	enter set name, enter element	Submit
ZRANGE (Range of elements in sorted set)	enter set name, enter starting range, enter end range, With Scores?	Submit
FLUSH DATABASE		Submit
FIND KEYS (* to find all keys)	enter key	Submit

Connection Established

The above shows the UI for the Redis implementation done for the given Task. The app runs on localhost at port no. 3000 This contains various segments shown below



The screenshot is annotated with red boxes and numbers 1 through 4. Box 1 highlights the list of Redis commands on the left. Box 2 highlights the input fields for the commands. Box 3 highlights the "Submit" buttons. Box 4 highlights the "Connection Established" status message.

Command	Input Fields	Action
SET METHOD	enter key, enter value	Submit
GET METHOD	enter key	Submit
DELETE KEY	enter key	Submit
EXPIRE	enter key, enter Time	Submit
GET TIME REMAINING TO EXPIRE	enter key	Submit
ZADD (Add element in sorted set)	enter set name, enter score, enter element	Submit
ZRANK (Rank of an element in sorted set)	enter set name, enter element	Submit
ZRANGE (Range of elements in sorted set)	enter set name, enter starting range, enter end range, With Scores?	Submit
FLUSH DATABASE		Submit
FIND KEYS (* to find all keys)	enter key	Submit

Connection Established

- The 1st block shows the Action which we want to perform.
- The 2nd block shows the various inputs required to perform the given action.
- The 3rd block contains submit button which on click performs the given task.
- The 4th block shows the output of the action performed.

Various Commands are performed in given task. These are as follows:

SET: This sets the key to hold the string value. If the key is already present, it is overwritten.

Time Complexity: $O(1)$

GET: This gets the value for given key. If the key is not present, it returns *nil*.

Time Complexity: $O(1)$

DELETE: This removes the given key from DB. If the key is not present, it is ignored

Time Complexity: $O(1)$ if key is string
 $O(M)$ otherwise where M is number of elements in list

EXPIRE: This Set a timeout for given key. This is measured in seconds. It returns 0 if timeout was set else return 0 if the key does not exist.

Time Complexity: $O(1)$

TTL: This returns the given time to live for which timeout is defined. Returns -2 if the key does not exist, return -1 if the key is found for Timeout is not set else return time remaining to expire.

Time Complexity: $O(1)$

ZADD: Adds the specific members with specific scores (Integers) in Sorted Set. If the member is already present then the score is updated.

Time Complexity: $O(\log(N))$ where N is number of elements in set.

ZRANK: This returns the rank of an element present in Sorted Set with Specified Key. If the member/Key is not present it returns *nil*.

Time Complexity: $O(\log(N))$ where N is number of elements in set.

ZRANGE: This returns the specific range of elements in Sorted Set at the specified Key.

It accepts one more parameter **WITHSCORE** which returns elements with their scores

Time Complexity: $O(\log(N) + M)$ where N is number of elements in set and M is number of elements returned.

FLUSHALL: This Deletes all the keys from all existing DB.

Time Complexity: $O(\log(N))$ where N is number of keys in all existing Databases.

KEYS: This returns all the keys with matching pattern.

- * This is used to match 0 or more characters.
- ? This is used to match exactly one character.
- [] This is used to specify characters between a given range.
- ^ This is used to exclude specific character.
- \ This is used to escape special characters if there are any.

Time Complexity: $O(N)$ where N is number of keys in Database.

QUESTIONS ASKED

Ques 1: Why did you choose that language?

Ans: I have chosen JavaScript as a language and Node.js as a development tool. The reasons for choosing Node.JS are:

- Node.js provides a **Highly Scalable development platform**.
- Node.js is **Highly Extensible**, which means that you can customize and further extend Node.js as per their requirements.
- Redis and Node.js share **similar type conventions and threading models**, which makes development easier and predictable for developers.

Ques 2: What are the further improvements that can be made to make it efficient?

Ans: Redis is one of the **fastest** and efficient in-memory key - value store. The improvements that can be made to make it more efficient are:

- **Pipelining** can help in sending multiple requests at the same time without waiting for replies and finally read reply in one step.
- Turning on **TCP KeepAlive** :- KeepAlive is a method that allows the same TCP connection for HTTP instead of opening a new connection for each new request.
- **Multi-Threading:** Redis is a single-process, single-thread model. Due to which only one CPU core can be used, and the multi-core advantages cannot be utilized.

Ques 3: What data structures have you used and why?

Ans: Redis offers 5 data structures, only one of which is a typical key-value structure. These are **strings, hashes, lists, sets and sorted sets**. This removes the need of using external data structures as the one provided by Redis are even faster than the external data structures. Each of them has several advantages and disadvantages. The data structures that I have used are:

- **Strings:** Strings are the **most basic** data structure available in Redis. These are **Key – value** pairs. It can **store any kind of data** - a string, integer, floating point value, JPEG image etc.
- **Stored Set:** Sorted set data structure is the **most powerful** data structure among all data structures in Redis. They are Sets with a **score**, which provides **sorting and ranking capabilities**.

Ques 4: Does your implementation support multi-threaded operations? If No why can't it be? If yes then how?

Ans: The current solution **doesn't** support multi-threaded operations. This is because Redis also runs multiple backend threads to perform backend cleaning works, such as cleansing the dirty data and closing file descriptors. To make Redis multi-threaded, the simplest way to think of is that every thread performs both I/O and command processing. However, as the data structure processed by Redis is complex, the multi-

thread needs to use the locks to ensure the thread security. Improper handling of the lock granularity may deteriorate the performance.

KeyDB: A multithreaded fork of Redis is available which can be used in place of current Redis implementation, but for the ease of implementation and lack of need I have decided to continue with **Node Redis Client** in place of KeyDB.