In memory key value database

Create a simple in memory key value datastore that performs operations on it based on certain commands and uses REST API for communication.

Transport

Read commands via HTTP REST API.
Use JSON encoding for requests and responses.
Use appropriate HTTP status codes for responses.

Query Validation

Ensure that the input command is valid before processing the request.

Storage

Use appropriate data structures.
Must support **concurrent** operations.

Commands to implement

SET

Writes the value to the datastore using the key and according to the specified parameters.

Pattern: SET <key> <value> <expiry time>? <condition>?

<key>
The key under which the given value will be stored.

<value>
The value to be stored.

<expiry time>
Specifies the expiry time of the key in seconds.
Must contain the prefix EX.
This is an optional field,

The field must be an integer value.

<condition>
Specifies the decision to take if the key already exists.
Accepts either NX or XX.

NX -- Only set the key if it does not already exist.
XX -- Only set the key if it already exists.
This is an optional field. The default behavior will be to upsert the value of the key.

Examples

SET key_a 2

Sets the value 2 in key_a and does not expire

SET key_b 3 EX 60

Sets the value 3 in key_b and expires in 60 seconds

SET key_c 4 EX 60 NX

Sets the value 4 in key_c, expires in 60 seconds and only sets the value if the key does not already exist.

SET key_d 5 XX

Sets the value 5 in key_d, does not expire and only sets the value if the key already exists.

GET

Returns the value stored using the specified key.

Pattern: GET <key>

Examples

GET key_a

QPUSH

Creates a queue if not already created and appends values to it.

Pattern: QPUSH <key> <value...>

<key>
Name of the queue to write to.

<value...>
Variadic input that receives multiple values separated by space.

Examples

QPUSH list_a 1
QPUSH list_a 2 3 4

Bonus Commands (Optional)

QPOP

Returns the last inserted value from the queue.

Pattern: QPOP <key>

<Key>
Name of the queue

Examples

QPUSH list_a 1
Returns OK

QPOP list_a
Returns 1

QPUSH list_a 1 2
Returns OK

QPOP list_a
Returns 2

QPOP list_a
Returns 1

QPOP list_x
Returns null

BQPOP

Blocking queue read operation that blocks the thread until a value is read from the queue.
The command must fail if multiple clients try to read from the queue at the same time.

Pattern: BQPOP <key> <timeout>

<key>
Name of the queue to read from.

<timeout>
The duration in seconds to wait until a value is read from the queue.
The argument must be interpreted as a double value.
A value of 0 immediately returns a value from the queue without blocking.

Example

Scenario 1

QPUSH list_1 a
Returns OK

BQPOP list_1 0
Returns a

Scenario 2

BQPOP list_1 0
Returns null

Scenario 3

BQPOP list_1 10
Blocks the request as the queue is empty.
Returns null after 10 seconds

Scenario 4

BQPOP list_1 10
Blocks the request as the queue is empty.

*The below command is issued from a different client
QPUSH list_1 a
Returns OK

“BQPOP list_1 10” will stop blocking the request and return the value 10 as “QPUSH list_1 a” inserts a value into the queue

Sample input and output

—----------

Input

```
{
   "command": "SET hello world"
}
```

Output

—----------

Input

```
{
   "command": "123 SET COMMAND"
}
```

Output

```
{
   "error": "invalid command"
}
```

—----------

Input

```
{
   "command": "GET hello"
}
```

Output

```
{
   "value": "world"
}
```

—----------

Input

```
{
   "command": "GET hello-123"
}
```

Output

```
{
   "error": "key not found"
}
```

—----------

Input

```
{
   "command": "QPUSH list_a a"
}
```


Output: BLANK

———————————

Input


```
{
   "command": "QPOP list_a"
}
```


Output


```
{
   "value": "a"
}
```


————————————————

Input


```
{
   "command": "QPOP list_a"
}
```


Output


```
{
   "error": "queue is empty"
}
```