**In the HTTP handler function, the two arguments are http.ResponseWriter and \*http.Request. These two arguments are used to read and write data to and from the client.**

* **http.ResponseWriter is an interface that provides methods to send HTTP response back to the client. It is used to write the response headers and the response body.**
* **\*http.Request is a struct that contains information about the incoming HTTP request. It includes things like the URL, headers, and body.**

**In the handler function, we use w to write the response back to the client, and r to read information about the incoming request. We can use w to set the response headers and write the response body, and r to read information about the request, such as its method, URL, headers, and body.**

**err.Error() returns the error as string**

**import:**

 encoding – is used to encode and convert json requests from the body when parsing requests

 "fmt": This library is used for printing and formatting text. In the code, it is used for printing error messages and writing response bodies to HTTP requests.

 "net/http": This library provides HTTP client and server implementations. In the code, it is used to handle HTTP requests and send HTTP responses.

 "strings": This library provides functions for manipulating strings. In the code, it is used to split the command string in the "/set" HTTP handler into parts.

 "sync": This library provides synchronization primitives such as mutexes. In the code, it is used to protect the data store from concurrent access.

 "time": This library provides functions for working with time. In the code, it is used to set and check expiry times for keys.

in golang the & gets the address, meanwhile the \* will fetch the value from the address that is stored in the variable

**datastore struct:**

data is the field of the struct which is pointing to the map

map[string] means that the map has keys of type string and it’s values is gonna be address of the datavalue

mu sync rwmutex is for assigning locks to the struct so that concurrent access to modify can be prevented ( more readers but only one writer)

**dataValue struct:**pretty much the same struct, collection of variables of different types

**function parseExpiry:**

The parseExpiry function is used to parse the expiry time parameter (if specified) for a set command. It converts the time duration string (e.g. "10s", "5m", "2h", "3d") to Unix timestamp format.

int64 and error ( in purple ) are actually return values from the func

unit is the last character we pass in the request body

duration is the integer value we extract using the slice of array with len of expiry

we convert the duration to integer using atoi from strconv package

return in this function is giving out two values separated by a “ , ”

**func setvalue()**

The blank line in the if statement is to ignore the first value of the two values being returned by the d.data[key]

d.data is a map and the key returns two values, one being the value and the other being the bool if key exists or not

we do the d\* datastore after function declaration because this way allows the func to access the contents of the struc

the syntax is called as receiver

line 59, we are setting value of the key in the data map which is of datastore and belongs to object d

d is an object, data is the map of datastore

**func getValue()**

Here again the receiver is allowing to access the datastore struc and the d is the pointer to the datastore

d.mu.rlock and runlock ensure that the datastore can be read by multiple readers at the same time

then we first check if the key already exists or not by using the map lookup d.data[key] and fetching the values and ok captures whether or not if the key exists

because the d.data[key] lookup gives back values as well as if whether the key exists by Boolean or not

then we check if the key has expired or not, If it has expired then we will delete the key and then if the key is valid, we return it on line 73 error is nil

if all else fails then we return key not found

line 87: we are checking if it is nil or not because we wanna know if the queue is initialized or not, if it’s not then we initialize it

line 100: we use the `make()` to make the map empty to store values, or else it would be a nil map

line 102: we wanna check if the exptime is 0 ie if it is set to never expire

**func main()**

**handlefunc:** this is the function that responds to requests made to the end point as described

**we** make a new instance of the struc data, we do this because only space is allocated when we declared it above, now we need to make a new instance of the it to make the map be stored in

line 122: json.newdecoder will return a decoder which will read from r.body, we will then call decode on this decoder and put it all in req struct

if error is not nil here then we return an error

line 127: splitting the command on white space

line 139: we then parse the remaining part of the request to see for EX NX and XX

line 140: If there is nothing after the EX then error

line 147: If there is something after EX then we pass the next element to parseexpiry func and get the expiry time and err from it, check if error is non nil, then return error

NX means if it does not exist then only set the values

XX means assign it only if it exists

line 165: it will catch error from the setvalue func, if it is not nil then we check for strings if they have any specific error type and then give our the writeheader content and also print the error in the end

line 182: we check if correct http method is being used or not

line 187: we retrieve the key from the query, if it is empty then return error that key was not found

line 194: we pass the key to func getValue and capture the value as well as error returned by the func

line 195: if error is not nil then we have to check if the error had some particular type of message and we send the wrteheader accordingly

line 205: if no error found then we print status ok and then give the value of the said key passed

**func qpush:**

line 221: We call a decoder on the decode which was on r.body by NewDecoder and if error is not null then we return the error, So, json.NewDecoder(r.Body) creates a new Decoder that reads from the request body (r.Body) and Decode() method decodes that JSON-encoded value and stores it in the variable req which is passed as a pointer &req

line 227: check if the command is indeed QPUSH, if not then return error

line 233: if at two least elements are in the request, if not then return error

line 239: capture key and values from Args

line 242: we push values using qpush to data

line 261: we capture the return value from getall() and use json.marshal to encode it in a json slice which we will then send back to the json response body

line 268: it is setting up the content type to application/json and this tells that the client is supposed to expect json format

line 274: we capture error when we say listen and serve at port 8080, if any error is captured then we check if err if it is not nil, if not then it will panic and exit the program

­­