I've been stuck because I felt overloaded with concepts and words that I couldn't follow. When you started the video and jumped right into "Marshal" I was like wtf, what the hell is a marshall? Quickly got confused since so many other topics built ontop of this concept.

For those who also got stuck I'm going to provide some information I had to go research in order to follow along and make it through this section.

What is JSON?

First JSON(JavaScript Object Notation) everyone should know this, it's just keyvalue pairs in JS and it's easier to transmit than XML. It's a human readable, machine transferable and generally the preferred way to to send and receive data via REST APIs. It's not the most efficient way but it's the web-developer preferred way.

What does 'Marshaling' mean?

marshaling is the process of transforming the memory representation of an object to a data format suitable for storage or transmission, and it is typically used when data must be moved between different parts of a computer program or from one program to another.

The inverse of marshaling is called unmarshaling or demarshaling.

As it relates to Golang and this section

This Marshaling and UnMarshaling is Golang trying to convert struct into JSON objects and JSON objects into Golang structs. This section of the course is about how you can transfer to JSON and from JSON(string literal byte slice) back into Golang struct.

Remember Golang is a web backend language, eventually we'll learn how to get JSON objects via HTTP request. While we don't yet know how to do that, Tod is preparing us for

that future where we'll have put into a byte slice a JSON object as a string literal. Or the reverse take a JSON object that is a byte slice and convert it to a Golang struct.

Recall that in Golang when you put a string in backtick ``, it is treated as raw string literal. Meaning that it is read exactly as is a UTF-8 string with no escape characters only runes(ut-8 characters). Vs the "" double quotes which allows for escaped characters.

In quotes `""` you need to escape new lines, tabs and other characters that do not need to be escaped in backticks ``. If you put a line break in a backtick string, it is interpreted as a `'\n'`

So now we have this JSON object that we got from HTTP somehow. How do we get into a struct or how do we get our nice Golang struct into JSON? That is what the JSON package is for.

Marshal function in JSON Package

When we want to convert a Golang struct into a JSON object, we use the json.Marshal. Marshal is Golangs way of saying "encode/convert to JSON Object". Because Golang is a strictly typed language and JSON is a dynamically typed language. A few things need to be known while constructing your struct for JSON transfer.

Exposed vs not Exposed fields

As with all structs in Go, it’s important to remember that only fields with a capital first letter are visible to external programs/packages like the JSON Marshaller. Meaning that if you don't capitalize the first letter it won't get exposed when you convert it to JSON using the json.Marshall() function.

Meaning of Interface{}

If you're not familiar an empty `interface{}` is a way of defining a variable in Go as "this could be anything". At runtime Go will then allocate the appropriate memory to fit whatever you decide to store in it.

The function signature for json.marshal

func Marshal(v interface{}) ([]byte, error)

it takes a v interface{}, which can be 'any' go type. Basically everything from a struct to a primitive it iwll take it and try to convert it to a JSON object. It returns two things.

1. a slice of byte []byte, containing the literal string that is the JSON object.
2. And error, letting you know if anything went wrong

Typically you give the JSON marshal function a pre-filled struct, or a raw string literal formatted to JSON

How are go types represented in JSON by the json Marshaller?

bool for JSON booleans,

float64 for JSON numbers,

string for JSON strings,

nil for JSON null.

Only data that can be represented as JSON will be encoded(converted) by the json.marshal() function.

Only the exported (public) fields of a struct will be present in the JSON output. A field with a *json: tag* is stored with its tag name instead of its variable name. Pointers will be encoded as the values they point to, or null if the pointer is nil.

Unmarshal function in the JSON package

When we want to convert a JSON Object into a Golang struct, we use the json.Unmarshal. Umarshal is Golangs way of saying "parse this JSON object into a valid GOlang struct".

The function signature for unmarshal

func Unmarshal(data []byte, v interface{}) error

unmarshal takes the following parameters:

1. a slice of bytes (This a raw string, this is the JSON object that you want to parse)
2. A pointer to a struct to parse the JSON into

Unmarshal returns,

A error if anything went wrong with parsing.

How does Unmarshal decide which fields to try and parse?

For any key found in the JSON, Unmarshal will try to match it to a key found in the struct with the following logic.

For explanation sakes, I'm using "FieldName" to represent any member of a struct.

1. It will first look for an exported(field member with a capital letter) with a tag json:"FieldName"
2. A exported (field member with a capital letter) with the name FieldName

3. Any exported field name, that matches the fieldname if casesensitivity is not an issue, e,g fIeLdNaMe, FIELDNAME, feildname.

ONLY FIELDS FOUNDS IN THE destination type(struct) will be decoded.

Only when a field is found in the destination struct will it be decoded, meaning if there is a field in the JSON that isn't in the destination it will be ignored.

This is useful when you wish to pick only a few specific fields. In particular, any unexported fields in the destination struct will be unaffected.

Other useful information to understand this video

The escaped character %+V

The escaped character %+V, in the printf statement in this video does the following. If it's a struct it will print the value of that structure with %V, when you have the +, %+V it will print the members of the struct in the print statement.

Relationship between strings and byte slices

Recall that a string in Go is just a sequence of bytes. A byte is just an alias for a a uint8. So the underpinnings of a string is a sequence of bytes. This is why we can easily do a conversion on a sequence of bytes and turn it into a string and vice versa. So when the marshal function returns a sequence of bytes, remember it's just a raw UTF-8 string.

Additional readings if you want to really dig in:

https://eager.io/blog/go-and-json/

https://yourbasic.org/golang/json-example/

https://medium.com/go-walkthrough/go-walkthrough-encoding-json-package-9681d1d37a8f

And finally after you read those, the documentation for JSON package will make so much more sense!

https://golang.org/pkg/encoding/json/#Marshal