API Converter to struct

<https://app.quicktype.io/>

Documentation of how to use custom coding key

<https://sarunw.com/posts/how-to-set-custom-codingkey-for-convertfromsnakecase-decoding-strategy/>

xcode short cuts

<https://betterprogramming.pub/13-xcode-shortcuts-to-boost-your-productivity-329c90512309>

**Codable**

The Codable protocol in Swift

Well, JSON is arguably the most common data format that we use to exchange data on the web. And Swift's Codable protocol was designed to provide a powerful and useful mechanism to convert JSON data into Swift structs.

In Swift, the Codable protocol is used to go from a JSON data object to an actual Swift class or struct. This is called decoding, because the JSON data is decoded into a format that Swift understands. It also works the other way: encoding Swift objects as JSON.

*Codable is actually a type alias that combines two protocols — Encodable and Decodable — into one.*

The Codable protocol is a composition of two protocols, Decodable and Encodable. These functions mean that for a type to be “decodable” or “encodable”, they’ll need to “decode from something” and “encode to something”.

Used to, we would use libraries like SwiftyJSON to make working with JSON easier, but we still needed to map the JSON data to Swift objects and properties.

After Swift 4 was introduced, you can use the Codable protocol instead!

Your Swift struct or class just has to adopt the Codable protocol, and you get JSON encoding and decoding, easily!

**Encoding and decoding JSON**

Let’s start with the basics: converting some JSON into Swift structs.

First, here’s some JSON to work with:

let json = """

[

{

"name": "James",

"age": 34

},

{

"name": "David",

"age": 31

}

]

"""

let data = Data(json.utf8)

The last line converts it to a Data object because that’s what Codable decoders work with.

Next we need to define a Swift struct that will hold our finished data:

struct User: Codable {

var name: String

var age: Int

}

Now we can go ahead and perform the decode:

let decoder = JSONDecoder()

do {

let decoded = try decoder.decode([User].self, from: data)

print(decoded[0].name)

} catch {

print("Failed to decode JSON")

}

That will print “James" – which is the name of the first user in the JSON.

**Heirachical Data**

We might have JSON that has hierarchical data (data nested inside another)

Take the following example:

let json = """

[

{

"name": {

"first\_name": "Matthew",

"last\_name": "Hunt"

},

"age": 22

}

]

"""

Codable can handle this as long as we handle the relationships.

One way to do this is using nested structs, like this:

struct User: Codable {

struct Name: Codable {

var firstName: String

var lastName: String

}

var name: Name

var age: Int

}

If you want to read a user’s first name, you would use user.name.firstName