GENERAL ASSEMBLY

iOS NETWORKING

Zeke Abuhoff

Lead iOS Instructor, General Assembly

LEARNING OBJECTIVES

+ Send HTTPS requests from iOS apps
+ Translate response data into JSON objects

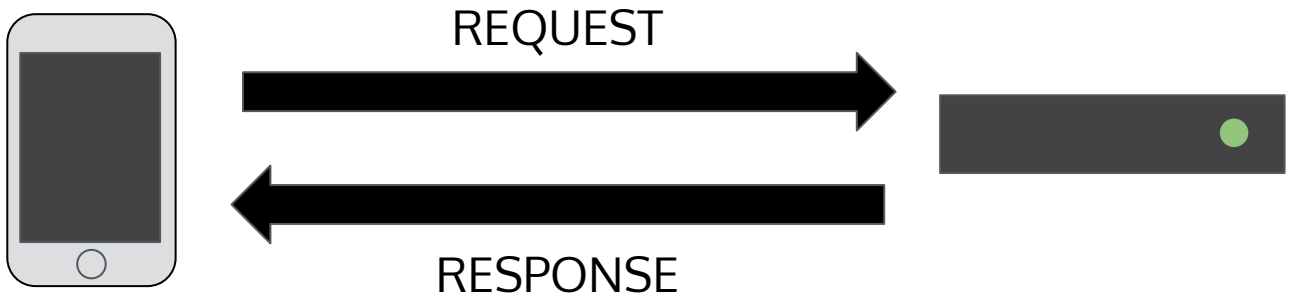
HTTPS REQUESTS

HTTPS requests serve as our primary way of retrieving data from the internet in an iOS app.

Are they different from HTTP requests? Not really. That extra ‘S’ just indicates a more recent, more secure version of the protocol. By default, iOS disallows regular HTTP requests, mandating the more secure approach.

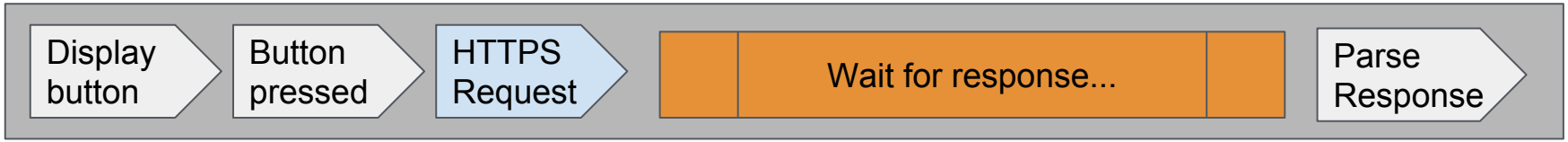
HTTPS REQUESTS

We send HTTPS requests from the device to a URL. The outcome of that request (whether it's the requested data or an error) comes back as a response.



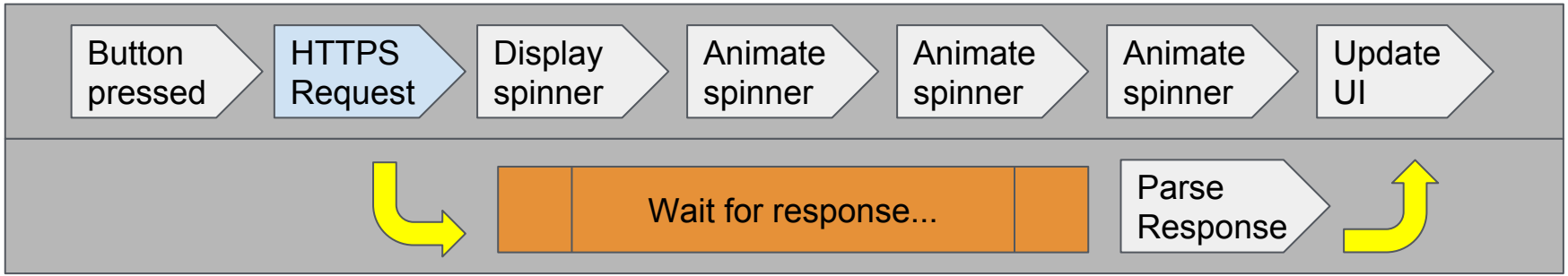
While this process is typically very fast, it does take time. Depending on network conditions and server response speed, it could take several seconds.

HTTPS REQUESTS



The gap between making a request and getting a response presents a programming problem for us. If we have the app do nothing while it waits for a response, then the app is frozen for the duration of the gap.

HTTPS REQUESTS



To keep the app working while we wait for a response, the wait and the response handling happen on a **background thread**. That means we'll have to write code differently. We'll have to use...

CLOSURES!

In the example below, we create a URL object and a completion handler (which is a closure), then we pass them in as parameters when the request kicks off.

```
let url: URL = URL(string: "https://google.com")!
let completion: (Data?, URLResponse?, NSError?) -> Void = { data, response, error in
     // This code will handle the response.
     // It will run later, when the response comes in.
}
URLSession.shared.dataTask(with: url, completionHandler: completion).resume()
```

CLOSURES!

In the example below, we create a URL object and a completion handler (which is a closure), then we pass them in as parameters when the request kicks off.

```
let url: URL = URL(string: "https://google.com")!
let completion: (Data?, URLResponse?, NSError?) -> Void = { data, response, error in
    // This code will handle the response.
    // It will run later, when the response comes in.
}
URLSession.shared.dataTask(with: url, completionHandler: completion).resume()
```

NOW

LATER

URLSESSION

URLSession is the class we use to make requests. We'll use the sharedSession instance.

```
URLSession.shared
```

The most basic data task a URLSession can produce simply pings a URL.

```
URLSession.shared.dataTask(with: myURL)
```

To elaborate, we can add a completion handler.

```
URLSession.shared.dataTask(with: myURL, completionHandler: myCompletionHandler)
```

URLSESSION

In order to specify more details, we can make a data task with a URLRequest object.

```
URLSession.shared.dataTask(with: myURLRequest, completionHandler: myCompletionHandler)
```

However we make a data task object, remember: it doesn't run until we call resume() on it.

```
myDataTask.resume()
```

Once we call resume(), the request kicks off. Later (maybe only by a fraction of a second), the response will come in and our completion handler will run.

URLSESSION

URL: http://owlbot.info:80/api/v1/dictionary/lexicon?format=json

Practice:

1) In your project's Info.plist file, set App Transport Security to allow arbitrary loads.
2) Create a data task that pings the provided URL.
3) Create a data task that handles data from the provided URL.
4) Create a data task that uses a URLRequest and handles data from the provided URL.

COMPLETION HANDLERS

You can now make HTTP requests with completion handlers - great!

But what do you put in those completion handlers?

Step One - Handle errors

Step Two - Parse data

Step Three - Deliver data to your model

COMPLETION HANDLERS

Step One - Handle errors

Remember: ‘guard’ and ‘guard let’ statements are your friends!

```
guard error == nil else {
     return
}
```

```
guard let responseData = data else {
     return
}
```

Take care of unexpected outcomes right away so you’re not trying to perform operations with invalid data.

COMPLETION HANDLERS

Step Two - Parse data

Swift can't guarantee the validity of data from a response, so its APIs for handling such data are couched in optionals and possible thrown errors.

```
do {
     let jsonObject = try JSONSerialization.jsonObject(with: responseData, options:
JSONSerialization.ReadingOptions.allowFragments)
     let jsonString = String(jsonObject)
     print("RESPONSE JSON: \(jsonString)")
} catch { return }
```

THROWN ERRORS?

What do mean, ‘thrown errors’? And what’s all this ‘do’ and ‘try’ stuff?

THROWN ERRORS?

What do mean, ‘thrown errors’? And what’s all this ‘do’ and ‘try’ stuff?

NEXT TIME!