SCIENCES SORBONNE UNIVERSITÉ

Networking

Fabrice.Kordon@lip6.fr

LIP6

As an introduction

Apps make intensive use of the network

- Find CPU strength or access data
  - **You better use 3G/4G or WIFI!**
  - **You may «cache» computed results locally**
- You may interact with others
  - **Via a server**
  - **Without a server**

You need to access services

- Web services, XML, JSON, etc…

As an introduction

Apps make intensive use of the network

- Find CPU strength or access data
  - **You better use 3G/4G or WIFI!**
  - **You may «cache» computed results locally**
- You may interact with others
  - **Via a server**
  - **Without a server**

You need to access services

- Web services, XML, JSON, etc…

In short, you are doing

A bit of history

Before iOS9

- NSURL, NSURLRequest, NSURLConnection
- Event loop always visible
  - **Load piece by piece**

Since iOS9

- (NS)URL, (NS)URLRequest, (NS)URLSession, (NS)URLSessionDataTask
- Event loop possibly hidden
  - **Things are managed via handlers**
- Associated protocols
  - **NSURLConnectionDataDelegate (data reception)**
  - **NSURLConnectionDelegate (connection status)**
  - **NSURLConnectionDownloadDelegate (download status)**

Initiating a connection

You already know a bit of it…

- Create an URL
  - **(NS)URL**
- Create a request
  - **(NS)URLRequest**
- Create a session
  - **(NS)URLSession**

New stuff

- Create a (NS)URLSessionDataTask
  - **Answer passed to a handler when query is completed**
  - **Nor more event loop**
- Connexion controlled by means of protocols

Initiating a connection

You already know a bit of it…

- Create an URL
  - (NS)URL
- Create a
  - (NS)URLRe
- Create a sess
  - (NS)URLSession

Remember WKWebWiew!

It show some of the protocol methods we have here!

New stuff

- Create a (NS)URLSessionDataTask
  - Answer passed to a handler when query is completed
  - Nor more event loop
- Connexion controlled by means of protocols

URLSessionDataTask

Objective

- Retrieve the content from a URL

Methods

- No handler provided

```
func dataTask(with url: URL) -> URLSessionDataTask
func dataTask(with request: URLRequest) -> URLSessionDataTask
```

- Handler provided

```
func dataTask(with url: URL,
        completionHandler: @escaping (Data?, URLResponse?, Error?) -> Void)
        -> URLSessionDataTask

func dataTask(with request: URLRequest,
        completionHandler: @escaping (Data?, URLResponse?, Error?) -> Void)
        -> URLSessionDataTask
```

  - **Handler called at completion**

- You must call Resume() to start the task

NSURLConnectionDataDelegate

Handles data reception

- Retrieving results

```
func connection(_ connection: NSURLConnection,
                didReceive response: URLResponse)

func connection(_ connection: NSURLConnection,
                didReceive data: Data)
```

- Query completed

```
func connectionDidFinishLoading(_ connection: NSURLConnection)
```

- Handling redirections

```
func connection(_ connection: NSURLConnection,
                willSend request: URLRequest,
                redirectResponse response: URLResponse?) -> URLRequest?
```

- Handling cache

```
func connection(_ connection: NSURLConnection,
        willCacheResponse cachedResponse: CachedURLResponse) -> CachedURLResponse?
```

- Etc.

NSURLConnectionDelegate

Provides feedback about the connection

- Handling authentication

```
func connection(_ connection: NSURLConnection,
                willSendRequestFor challenge: URLAuthenticationChallenge)

func connectionShouldUseCredentialStorage(_ connection: NSURLConnection)
```

- Handling error

```
func connection(_ connection: NSURLConnection,
                didFailWithError error: Error)
```

NSURLConnectionDownloadDelegate

Handling download

- Quantity of received data

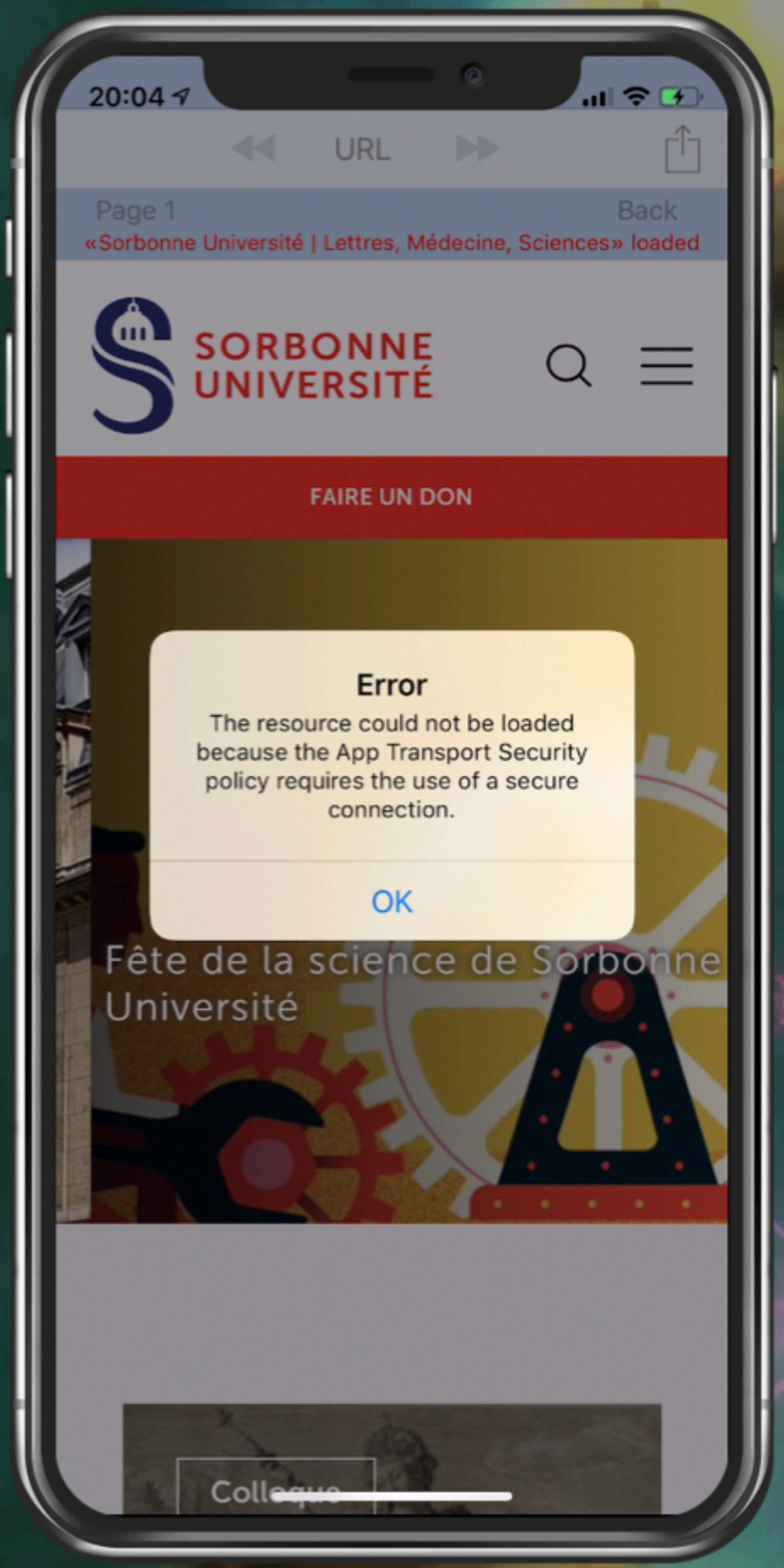
```
func connection(_ connection: NSURLConnection,
                didWriteData bytesWritten: Int64,
                totalBytesWritten: Int64,
                expectedTotalBytes: Int64)

func connectionDidResumeDownloading(_ connection: NSURLConnection,
                                    totalBytesWritten: Int64,
                                    expectedTotalBytes: Int64)
```

- End of a download

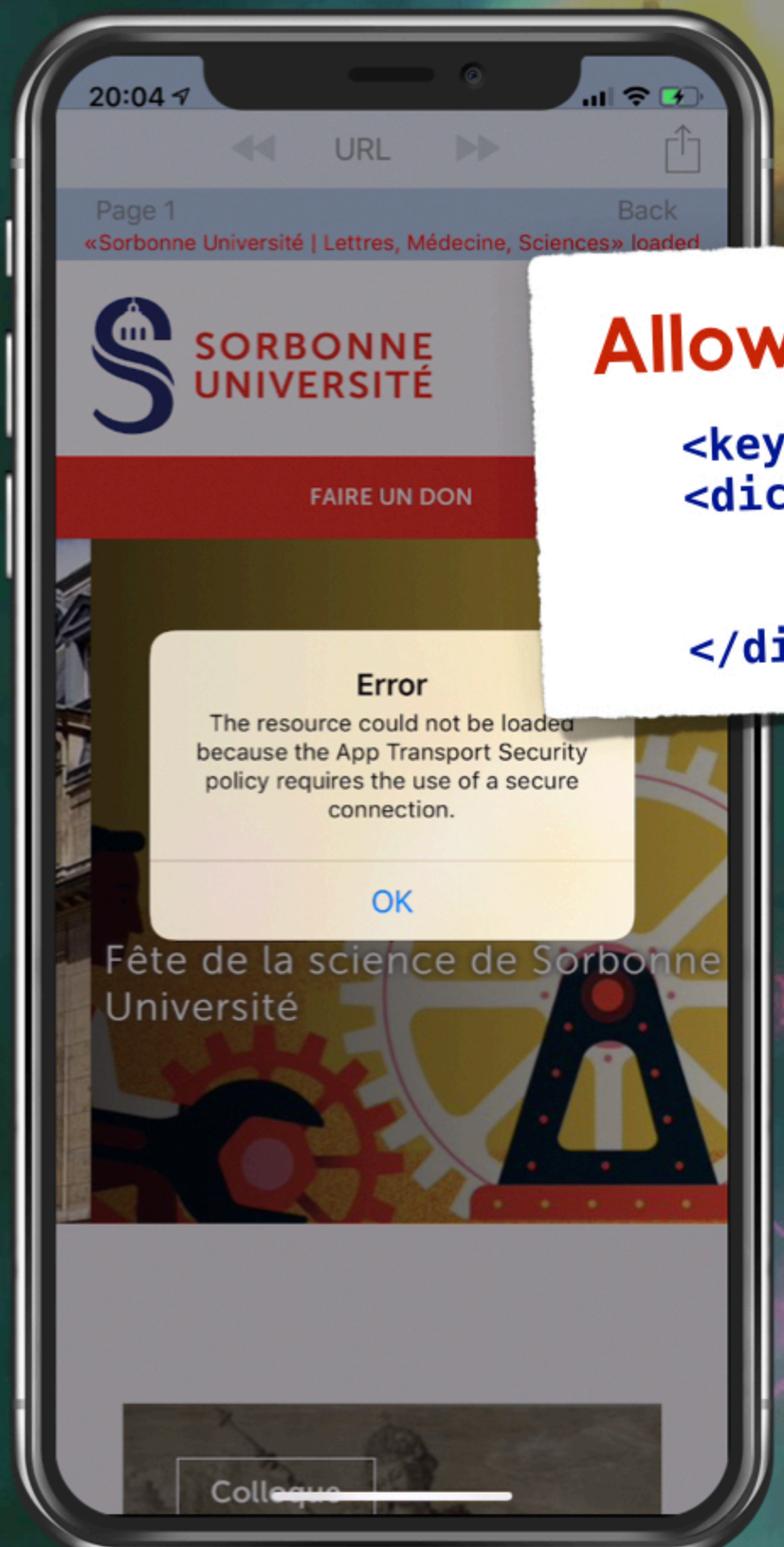
```
func connectionDidFinishDownloading(_ connection: NSURLConnection,
                                    destinationURL: URL)
```

Remember security aspects



- **Explicitly allow insecure connections**
  - And possibly avoid them

Remember security aspects

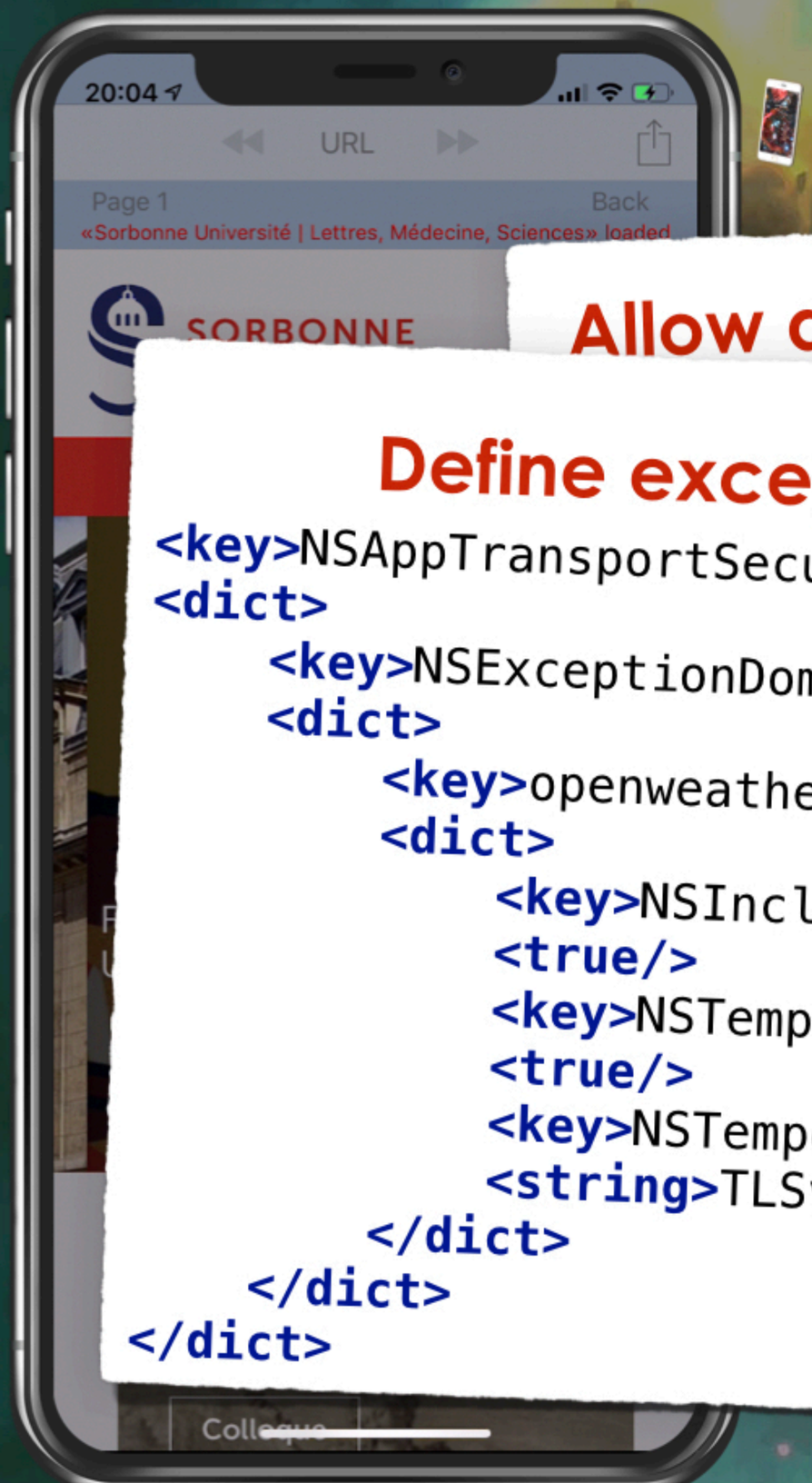


Explicitly allow insecure connections

Allow all insecure connections

```
<key>NSAppTransportSecurity</key>
<dict>
    <key>NSAllowsArbitraryLoads</key>
    <true/>
</dict>
```

Remember security aspects



Explicitly allow insecure connections

Allow all insecure connections

Define exception for insecure connections

```
<key>NSAppTransportSecurity</key>
<dict>
    <key>NSExceptionDomains</key>
    <dict>
        <key>openweathermap.org</key>
        <dict>
            <key>NSIncludesSubdomains</key>
            <true/>
            <key>NSTemporaryExceptionAllowsInsecureHTTPLoads</key>
            <true/>
            <key>NSTemporaryExceptionMinimumTLSVersion</key>
            <string>TLSv1.1</string>
        </dict>
    </dict>
</dict>
```

As a conclusion…

Not so complex is'n't it?

- Mainly for simple (but often used) stuff

You may now avoid the event loop

- Hidden to the user

In short, nothing but «traditional networking»