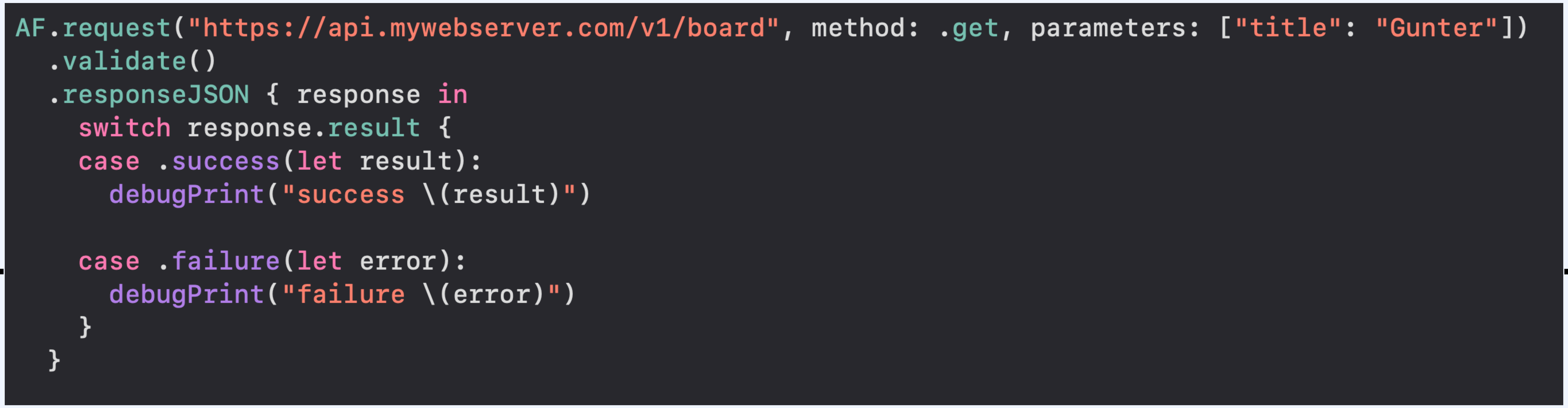
30개 프로젝트로 배우는 iOS 앱 개발

02. Alamofire 을 이용한 HTTP 통신 알아보기

Alamofire 란?

Alamofire 는 Swift 기반의 HTTP 네트워킹 라이브러리입니다.

URLSession 대신 Alamofire을 사용하는 이유



```swift
AF.request("https://api.mywebserver.com/v1/board", method: .get, parameters: ["title": "Gunter"])
  .validate()
  .responseJSON { response in
    switch response.result {
    case .success(let result):
      debugPrint("success \(result)")

    case .failure(let error):
      debugPrint("failure \(error)")
    }
  }
```

코 도

URLSession 대신 Alamofire을 사용하는 이유

URLSession

```swift
// 호출 URL 만들기
var components = URLComponents(string: "https://api.mywebserver.com/v1/board")!
components.queryItems = ["title": "Gunter"].map { (key, value) in
    URLQueryItem(name: key, value: value)
}

// Request 생성 및 실행
let request = try! URLRequest(url: components.url!, method: .get)
URLSession.shared.dataTask(with: request) { (data, response, error) in
    do {
        guard let data = data,
            let response = response as? HTTPURLResponse, (200 ..< 300) ~= response.statusCode,
            error == nil else {
            throw error ?? Error.requestFailed
        }
        let reponse = try JSONDecoder().decode(Response.self, from: data)
        print("Success \(reponse)")
    } catch {
        print("failure: \(error.localizedDescription)")
    }
}
```

Alamofire

```swift
AF.request("https://api.mywebserver.com/v1/board", method: .get, parameters: ["title": "Gunter"])
  .validate()
  .responseJSON { response in
    switch response.result {
    case .success(let result):
      debugPrint("success \(result)")

    case .failure(let error):
      debugPrint("failure \(error)")
    }
  }
```

Alamofire Request

```
open func request<Parameters: Encodable>(_ convertible: URLConvertible,
                                          method: HTTPMethod = .get,
                                          parameters: Parameters? = nil,
                                          encoder: ParameterEncoder = URLEncodedFormParameterEncoder.default,
                                          headers: HTTPHeaders? = nil,
                                          interceptor: RequestInterceptor? = nil) -> DataRequest
```

Alamofire HTTP Method

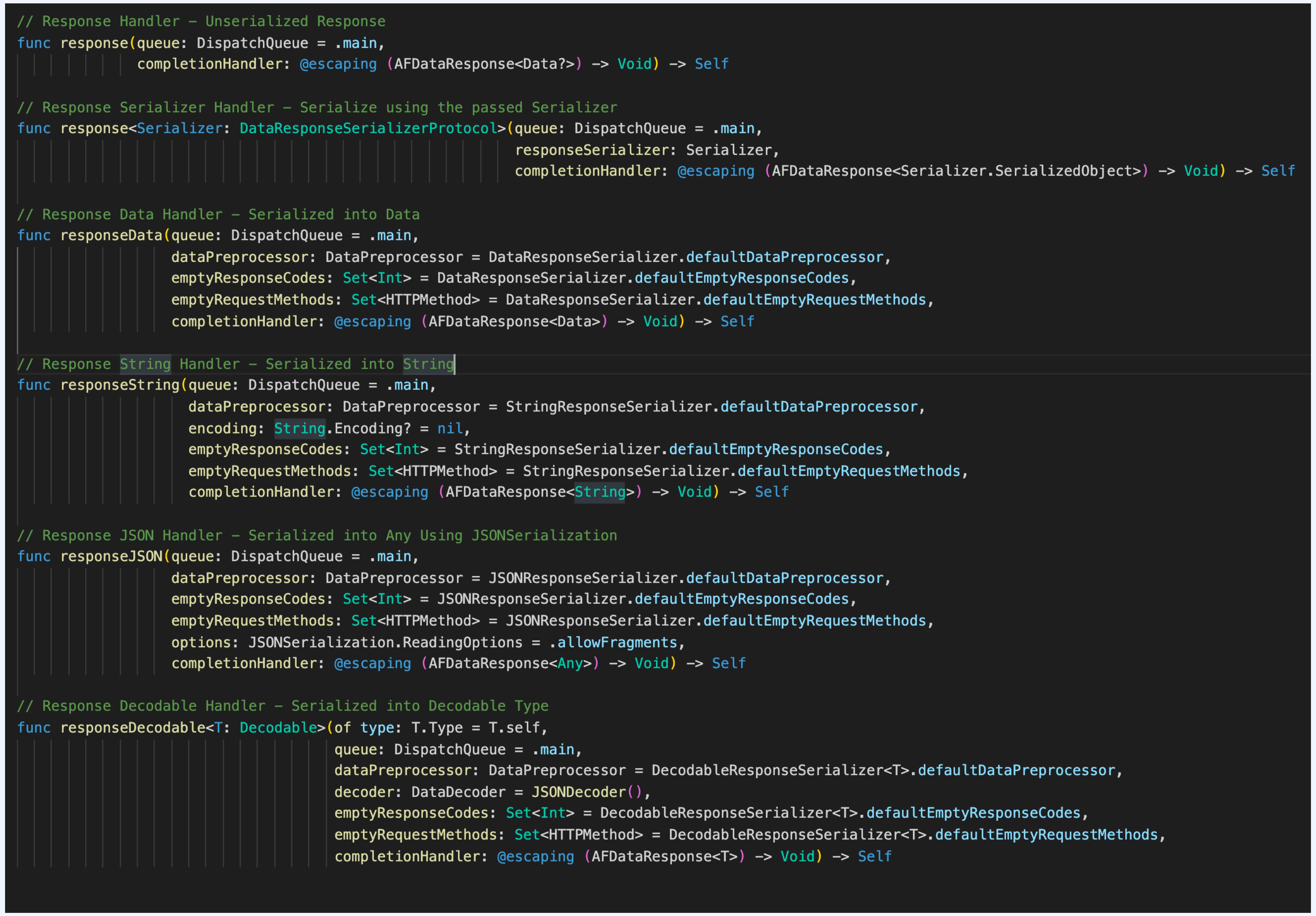
```swift
public struct HTTPMethod: RawRepresentable, Equatable, Hashable {
    public static let connect = HTTPMethod(rawValue: "CONNECT")
    public static let delete = HTTPMethod(rawValue: "DELETE")
    public static let get = HTTPMethod(rawValue: "GET")
    public static let head = HTTPMethod(rawValue: "HEAD")
    public static let options = HTTPMethod(rawValue: "OPTIONS")
    public static let patch = HTTPMethod(rawValue: "PATCH")
    public static let post = HTTPMethod(rawValue: "POST")
    public static let put = HTTPMethod(rawValue: "PUT")
    public static let trace = HTTPMethod(rawValue: "TRACE")

    public let rawValue: String

    public init(rawValue: String) {
        self.rawValue = rawValue
    }
}
```

```swift
AF.request("https://httpbin.org/get")
AF.request("https://httpbin.org/post", method: .post)
AF.request("https://httpbin.org/put", method: .put)
AF.request("https://httpbin.org/delete", method: .delete)
```

Alamofire Response



```swift
// Response Handler - Unserialized Response
func response(queue: DispatchQueue = .main,
              completionHandler: @escaping (AFDataResponse<Data?>) -> Void) -> Self

// Response Serializer Handler - Serialize using the passed Serializer
func response<Serializer: DataResponseSerializerProtocol>(queue: DispatchQueue = .main,
                                                          responseSerializer: Serializer,
                                                          completionHandler: @escaping (AFDataResponse<Serializer.SerializedObject>) -> Void) -> Self

// Response Data Handler - Serialized into Data
func responseData(queue: DispatchQueue = .main,
                  dataPreprocessor: DataPreprocessor = DataResponseSerializer.defaultDataPreprocessor,
                  emptyResponseCodes: Set<Int> = DataResponseSerializer.defaultEmptyResponseCodes,
                  emptyRequestMethods: Set<HTTPMethod> = DataResponseSerializer.defaultEmptyRequestMethods,
                  completionHandler: @escaping (AFDataResponse<Data>) -> Void) -> Self

// Response String Handler - Serialized into String
func responseString(queue: DispatchQueue = .main,
                    dataPreprocessor: DataPreprocessor = StringResponseSerializer.defaultDataPreprocessor,
                    encoding: String.Encoding? = nil,
                    emptyResponseCodes: Set<Int> = StringResponseSerializer.defaultEmptyResponseCodes,
                    emptyRequestMethods: Set<HTTPMethod> = StringResponseSerializer.defaultEmptyRequestMethods,
                    completionHandler: @escaping (AFDataResponse<String>) -> Void) -> Self

// Response JSON Handler - Serialized into Any Using JSONSerialization
func responseJSON(queue: DispatchQueue = .main,
                  dataPreprocessor: DataPreprocessor = JSONResponseSerializer.defaultDataPreprocessor,
                  emptyResponseCodes: Set<Int> = JSONResponseSerializer.defaultEmptyResponseCodes,
                  emptyRequestMethods: Set<HTTPMethod> = JSONResponseSerializer.defaultEmptyRequestMethods,
                  options: JSONSerialization.ReadingOptions = .allowFragments,
                  completionHandler: @escaping (AFDataResponse<Any>) -> Void) -> Self

// Response Decodable Handler - Serialized into Decodable Type
func responseDecodable<T: Decodable>(of type: T.Type = T.self,
                                     queue: DispatchQueue = .main,
                                     dataPreprocessor: DataPreprocessor = DecodableResponseSerializer<T>.defaultDataPreprocessor,
                                     decoder: DataDecoder = JSONDecoder(),
                                     emptyResponseCodes: Set<Int> = DecodableResponseSerializer<T>.defaultEmptyResponseCodes,
                                     emptyRequestMethods: Set<HTTPMethod> = DecodableResponseSerializer<T>.defaultEmptyRequestMethods,
                                     completionHandler: @escaping (AFDataResponse<T>) -> Void) -> Self
```

Alamofire Response

```
AF.request("https://httpbin.org/get").responseJSON { response in
    debugPrint(response)
}
```