Сетевое взаимодействие

- показ веб-страниц
  - просто показ
  - дополнительное взаимодействие
- быстрые загрузки
  - парсинг данных
- долгие загрузки (фон)

Показ веб-страниц

Просто показывать веб-страницы умеет SFSafariViewController

- внутри обычный safari
- общие с safari пароли и прочее
- никакого контроля со стороны приложения

Открыть в Safari


```
guard let url = URL(string: “https://google.com") else {
    return
}

UIApplication.shared.open(url)
```

Показ веб-страниц

Иногда этого недостаточно, и надо использовать WKWebView

- контроль над навигацией и вводом
- обработка actions из webView
- ускоренный js

WebSocket

Постоянно живущее двустороннее соединение поверх http

- много запросов
- получение обновление от сервера
- сторонние библиотеки

Низкий уровень (TCP, UDP etc.)

Это если совсем что-то странное нужно

- CFNetworking
- Библиотеки по ссылкам

HTTP

По большей части надо не показывать страницы, грузить данные

- json
- картинки

HTTP

Загрузка данных как с сервера, так и на сервер делается посредством URLSession

- http, ftp протоколы
- https
- работа в фоне

HTTP

При загрузке json его надо распарсить

- JSONSerialization превращает в Dictionary/Array

```
let json = try? JSONSerialization.jsonObject(with: data, options: [])
```

HTTP

JSON объект


```
{
    "userId": 1,
    "id": 1,
    "title": "sunt aut facere repellat provident occaecati excepturi optio reprehender
it",
    "body": "quia et suscipit\nsuscipit recusandae consequuntur expedita et cum\nrepre
henderit molestiae ut ut quas totam\nnostrum rerum est autem sunt rem eveniet architec
to"
  }
```


```
struct Post {
    let userId: Int
    let id: Int
    let title: String
    let body: String
}
```

Достаем все руками из словаря


```
extension Post {

    init?(dict: NSDictionary) {
        guard
            let userId = dict["userId"] as? Int,
            let id = dict["id"] as? Int,
            let title = dict["title"] as? String,
            let body = dict["body"] as? String
            else { return nil }

        self.userId = userId
        self.id = id
        self.title = title
        self.body = body
    }


}
```

Codable

typealias Codable = Encodable & Decodable

- JSONDecoder превращает JSON в нашу модель (класс)
- JSONEncoder превращает инстанс класса в JSON

Codable


```
struct Post: Codable {
    let userId: Int
    let id: Int
    let title: String
    let body: String

    // генерится сама
    private enum CodingKeys: String, CodingKey {
        case userId
        case id
        case title
        case body
    }
}
```

JSONEncoder


```
let post = Post(userId: 1, id: 1, title: "hey", body: "you")

let encoder = JSONEncoder()
encoder.outputFormatting = .prettyPrinted

let data = try encoder.encode(post)
print(String(data: data, encoding: .utf8))
```

JSONDecoder


```
let decoder = JSONDecoder()

do {
    let posts = try decoder.decode([Post].self, from: data)
    print(posts)
} catch let error  {
    print("Parsing Failed \(error.localizedDescription)")
}
```

Сторонние штуки

- ALAMOFIRE
- MOYA
- REST KIT

Ссылки

- https://github.com/tidwall/SwiftWebSocket
- https://github.com/facebook/SocketRocket
- https://github.com/robbiehanson/CocoaAsyncSocket
- https://github.com/IBM-Swift/BlueSocket
- https://swiftbook.ru/post/tutorials/everything-about-codable-in-swift4/ - codable
- https://habr.com/en/post/414221/