# Привет!

# Работа с сетью


# Как вообще работает интернет (очень быстро)

# HTTP

- текстовый протокол
- передача информации в Интернете
- де-факто основной транспорт в текущий момент времени

# HTTP. Методы

- **OPTIONS** - получение информации о веб-сервере
- **GET** - получение содержимого ресурса, параметры идут после «?»
- **HEAD** - тот же GET, но без тела в ответе. Нужен чтоб проверить наличие ресурса или дату его изменения
- **POST** - модификация ресурса
- **PUT** - создание или модификация ресурса, от POST отличается тем, что POST предполагает обработку данных на сервере
- **PATCH** - тот же PUT, но для обновления фрагмента ресурса
- **DELETE** - удаление ресурса
- **TRACE** - проверка изменения запроса промежуточными серверами

# HTTP. Идемпотентность

- это страшное слово означает способность метода вернуть один и тот же результат при многократных запросах с одним и тем же параметром
- идемпотентные методы HTTP: **GET, HEAD, PUT, DELETE**
- идемпотентные методы **можно кешировать** на клиенте (часто это делается фреймворками и непрозрачно для разработчика)
- **POST не кешируется**

# HTTP. Коды состояний

- Информационные (**1xx**)-практически не используются
- Успешно(**2xx**)-запрос удачно выполнен
- Перенаправление(**3xx**)-переадресация на другую страницу
- Ошибка клиента(**4xx**)-что то пошло не так на стороне пользователя
- Ошибка сервера(**5xx**)-что то поломалось на стороне сервера

# HTTPS

- обычный HTTP работающий поверх SSL/TLS
- SSL/TLS - прослойка между транспортным и прикладным уровнями, обеспечивающая шифрованное соединение

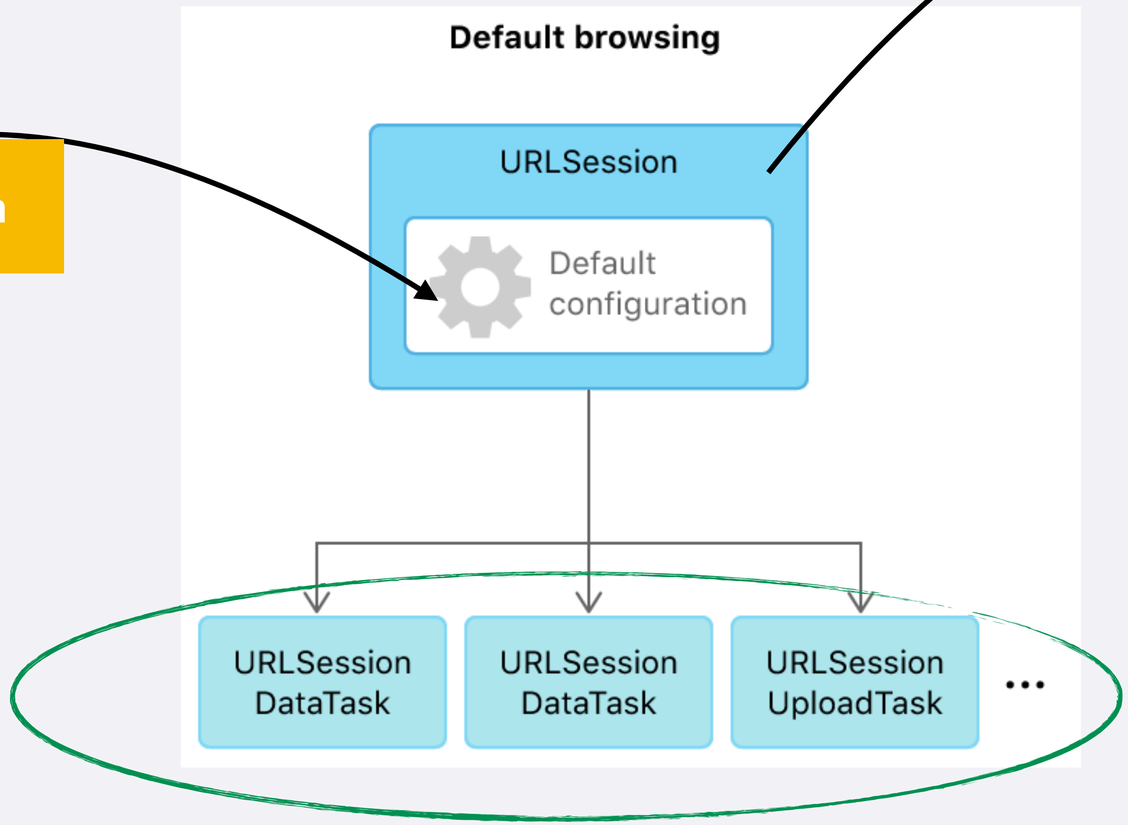
Как в iOS?

1. Только https

# URLSession (NSURLSession)

**URLSessionConfiguration**

- Конфигурация сессии




**Delegate**

- Если делегата нет, вызывается блок
- Если есть, блок не вызывается
- Обычно делегат не нужен

- Это задачи на работу с сетью. Например, скачать файл с пёсиком
- Подклассы URLSessionTask

Сбербанк Онлайн

Devteam

# URLSession (NSURLSession)

```
let config = URLSessionConfiguration.default
let session = URLSession(configuration: config)
```

- Типы
  - .default - стандартная
  - .ephemeral - инкогнито. Не сохраняет на диск кэши, пароли или любые другие данные. Хранит их в оперативке.
  - .background - позволяет загружать и скачивать данные даже при выключенном приложении.

- NSURLSession.sharedSession - для базовых запросов. Нет конфигурации.

# NSURLSessionConfiguration

```
let config = URLSessionConfiguration.default
let session = URLSession(configuration: config)
```

После создания конфигурации её уже не получится изменить

- Определяет таймауты, политику кэширования, требования к соединению, дополнительные HTTP заголовки, ждать ли появления доступа к сети и многое другое
- Типы
  - .default - стандартная
  - .ephemeral - инкогнито. Не сохраняет на диск кэши, пароли или любые другие данные. Хранит их в оперативке.
  - .background - позволяет загружать и скачивать данные даже при выключенном приложении.
- NSURLSession.sharedSession - для базовых запросов. Нет конфигурации.

# URL (NSURL)

```
let url = URL(string: "https://developer.apple.com/documentation")
```

# URLSessionTask

- Четыре типа
  - URLSessionDataTask - запрашивает у ресурса данные, хранит ответ сервера и полученные данные в памяти как  NSData
  - URLSessionUploadTask - загружает данные на сервер.
  - URLSessionDownloadTask - загружает данные в файл на диске.
  - URLSessionStreamTask - делает соединение к порту сервиса.

# URLSessionTask

- Создание задачи на работу с интернетом:

```
let url = URL(string: "https://developer.apple.com/documentation")!
let task = session.dataTask(with: url) { (data: Data?, response: URLResponse?, error: Error?) in
    // то, что будем делать после выполнения задачи
}
```

- Использование:

```
task.resume() // запускаем задачу на загрузку
```

```
task.cancel() // отменяет задачу.
task.suspend() // ставит задачу на паузу.
```

# URLSessionTask

- Можно создавать таск не через URL, а с помощью URLRequest.
- Так больше гибкости и можно заменять настройки из URLSessionConfiguration

```
let request = URLRequest(url: URL,
                         cachePolicy: URLRequest.CachePolicy,
                         timeoutInterval: TimeInterval)

let task = session.dataTask(with: url) { (data: Data?, response: URLResponse?, error: Error?) in
    // то, что будем делать после выполнения задачи
}
```

# Что в итоге получилось:

```
let config = URLSessionConfiguration.default
let session = URLSession(configuration: config)

let url = URL(string: "https://developer.apple.com/documentation")!
let request = URLRequest(url: url,
                         cachePolicy: .useProtocolCachePolicy,
                         timeoutInterval: 120) // в секундах

let task = session.dataTask(with: request) { (data: Data?, response: URLResponse?, error: Error?) in
    // то, что будем делать после выполнения задачи
}

task.resume() // запускаем задачу на загрузку
```

# Скачали. И что с этим делать?

# Первый вариант (старый) - разобрать пришедший JSON

```
let task = session.dataTask(with: request) { (data: Data?, response: URLResponse?, error: Error?) in
    // то, что будем делать после выполнения задачи
    let json = try? JSONSerialization.jsonObject(with: data!, options: []) as? [String: AnyObject]
    if let json = json {
        // обрабатываем json
        print(json)
        return
    }

    // иначе обрабатываем ошибку
}

task.resume()
```

# Второй вариант - Decodable

```
struct Post: Decodable {
    let id: String
    let title: String
    let subtitle: String?
}
```

```
do {
    let posts = try JSONDecoder().decode([Post].self, from: json.data(using: .utf8)!)
} catch {
    print(error)
}
```

Сбербанк Онлайн

Devteam

# Decodable со своими ключами

```
struct FirebaseCategoryDTO: Decodable {
    var name: String?
    var type: String?

    init(name: String) {
        self.name = name
    }

    enum CodingKeys: String, CodingKey {
        case name = "name"
        case type = "type"
    }

    init(from decoder: Decoder) throws {
        do {
            let container = try decoder.container(keyedBy: CodingKeys.self)

            name = try? container.decode(String.self, forKey: .name)
            type = try? container.decode(String.self, forKey: .type)
        } catch {
            assertionFailure("failed to decode CategoryDTO: \(error)")
        }
    }

}
```

# Тогда для того, чтобы закодировать - Encodable

```
struct FirebaseCategoryDTO: Encodable {
    var name: String?
    var type: String?

    init(name: String) {
        self.name = name
    }

    enum CodingKeys: String, CodingKey {
        case name = "name"
        case type = "type"
    }

    func encode(to encoder: Encoder) throws {
        var container = encoder.container(keyedBy: CodingKeys.self)
        try container.encode(name, forKey: .name)
        try container.encode(type, forKey: .type)
    }
```

```
let userData = try JSONEncoder().encode(user)
```

Сбербанк Онлайн

Devteam

# В реальном мире используйте ещё и DTO

```
struct RealCategory {

    var name: String
    var type: String // энам, что угодно

    // когда
    var dto: FirebaseCategoryDTO {
        var dto = FirebaseCategoryDTO(name: name)
        dto.type = type
        return dto
    }

    init(dto: FirebaseCategoryDTO) {
        name = dto.name ?? ""
        type = dto.type ?? "Неизвестный"
    }
}
```

Сбербанк Онлайн

Devteam

# В MVC

- Выносим во отдельный сервис.
  Сервис - как свойство у контроллера.
- Можно сделать отдельный сервис по загрузке, например, аватарок

# Загрузка для TableView и CollectionView

- TableViewDelegate (CollectionViewDelegate)
- Останавливаем загрузку картинок, которые уже не нужны
- Когда видим, что тэйбл вью останавливается - начинаем
- Спиннер

```
@available(iOS 2.0, *)
optional func scrollViewWillBeginDecelerating(_ scrollView:
    UIScrollView) // called on finger up as we are moving

@available(iOS 2.0, *)
optional func scrollViewDidEndDecelerating(_ scrollView:
    UIScrollView) // called when scroll view grinds to a halt
```

```
// called on start of dragging (may require some time and or
    distance to move)
@available(iOS 2.0, *)
optional func scrollViewWillBeginDragging(_ scrollView:
    UIScrollView)
```

Сбербанк Онлайн

Devteam

# Полезное

- Postman - тренироваться отправлять запросы
- Firebase realtime database - SAAS
    - Используйте REST API, а не iOS фрэймворк

# Что почитать?

https://developer.apple.com/documentation/foundation/NSURLSession - про работу с сетью. Там снизу много гайдов

https://www.youtube.com/watch?v=PsLzEAsphbM&list=PLrCZzMib1e9pg7ZLlOhmGSlmkMf8yEOLZ - хороший и понятный видос о том, как работают все
эти заголовки