Яндекс

Лекция 6.0

URL Loading System

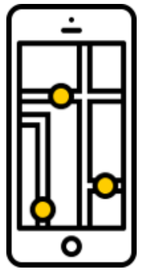
Fetch the world

Сегодня в лекции

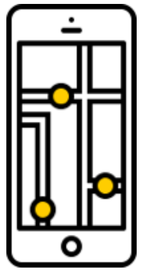
- Зачем iOS-разработчику HTTP?
- Как загрузить картинку
- Как сделать POST-запрос
- Как настроить кэширование
- Как управлять таймаутами и отменять запросы
- Как реализовать аутентификацию

HTTP

Клиент-серверная архитектура



Клиент-серверная архитектура



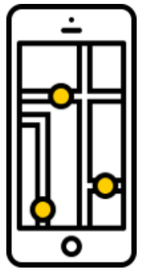
Мне нужен список сообщений!

Клиент-серверная архитектура

Мне нужен список сообщений!

На, держи!

Клиент-серверная архитектура



GET /messages HTTP/1.1

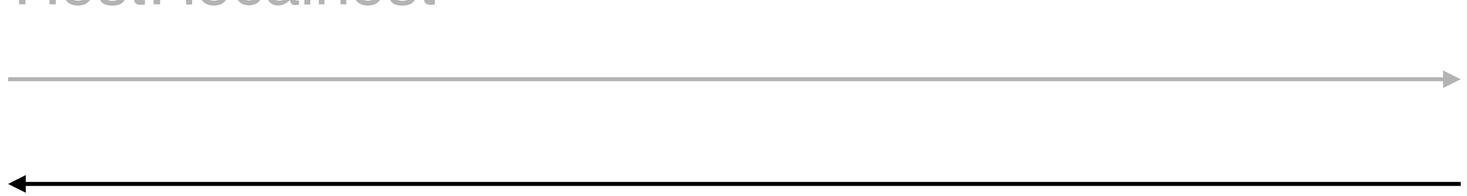
Host: localhost

На, держи!

Клиент-серверная архитектура

GET /messages HTTP/1.1

Host: localhost



HTTP/1.1 200 OK

Content-Type: application/json; charset=utf-8

Content-Length: 16

[‘Hello’, ‘Hi!”]

Структура протокола

GET /messages HTTP/1.1

Host: localhost

HTTP/1.1 200 OK

Content-Type: application/json; charset=utf-8

Content-Length: 16

[‘Hello’, ‘Hi!’]

Структура протокола

GET /messages HTTP/1.1

Host: localhost

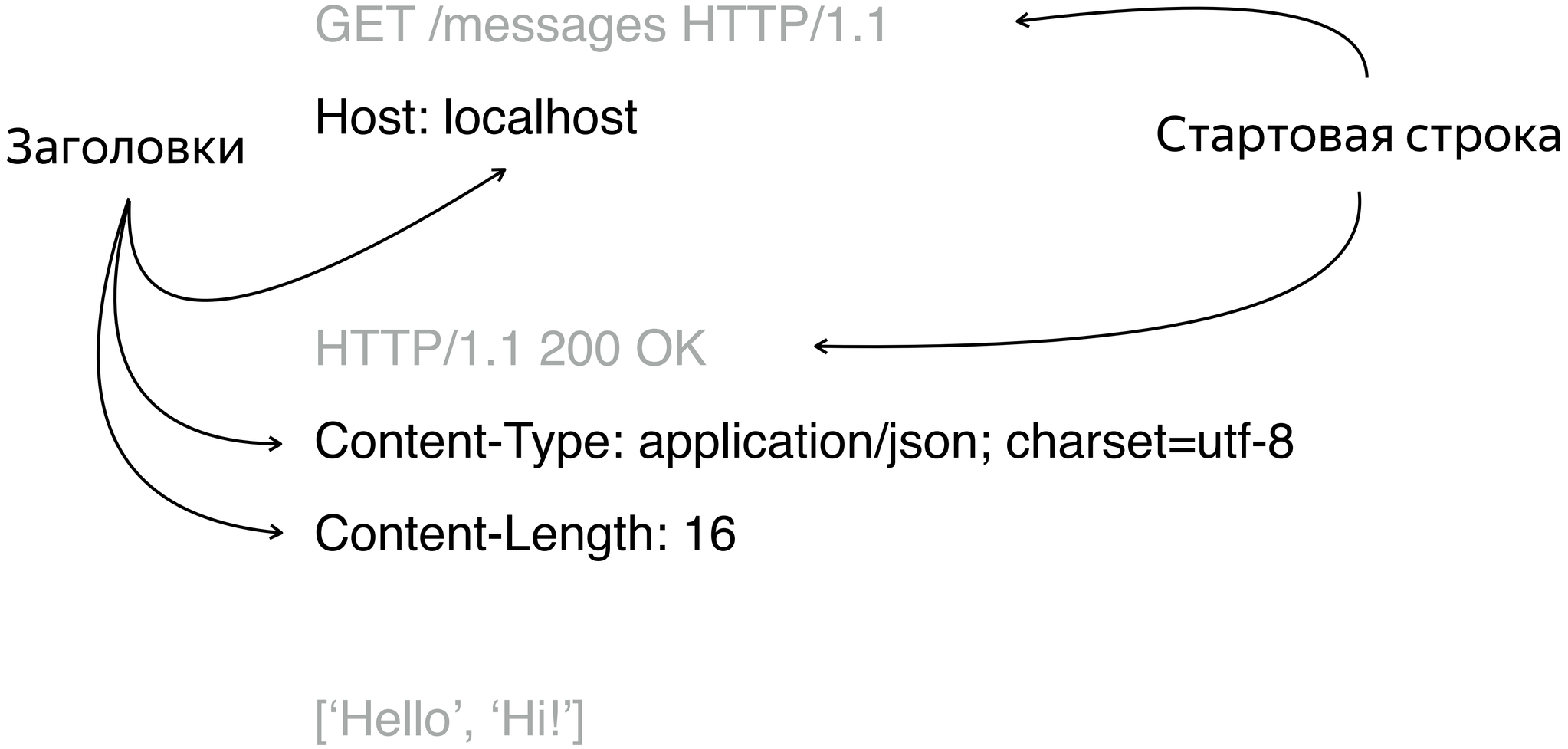
HTTP/1.1 200 OK

Content-Type: application/json; charset=utf-8

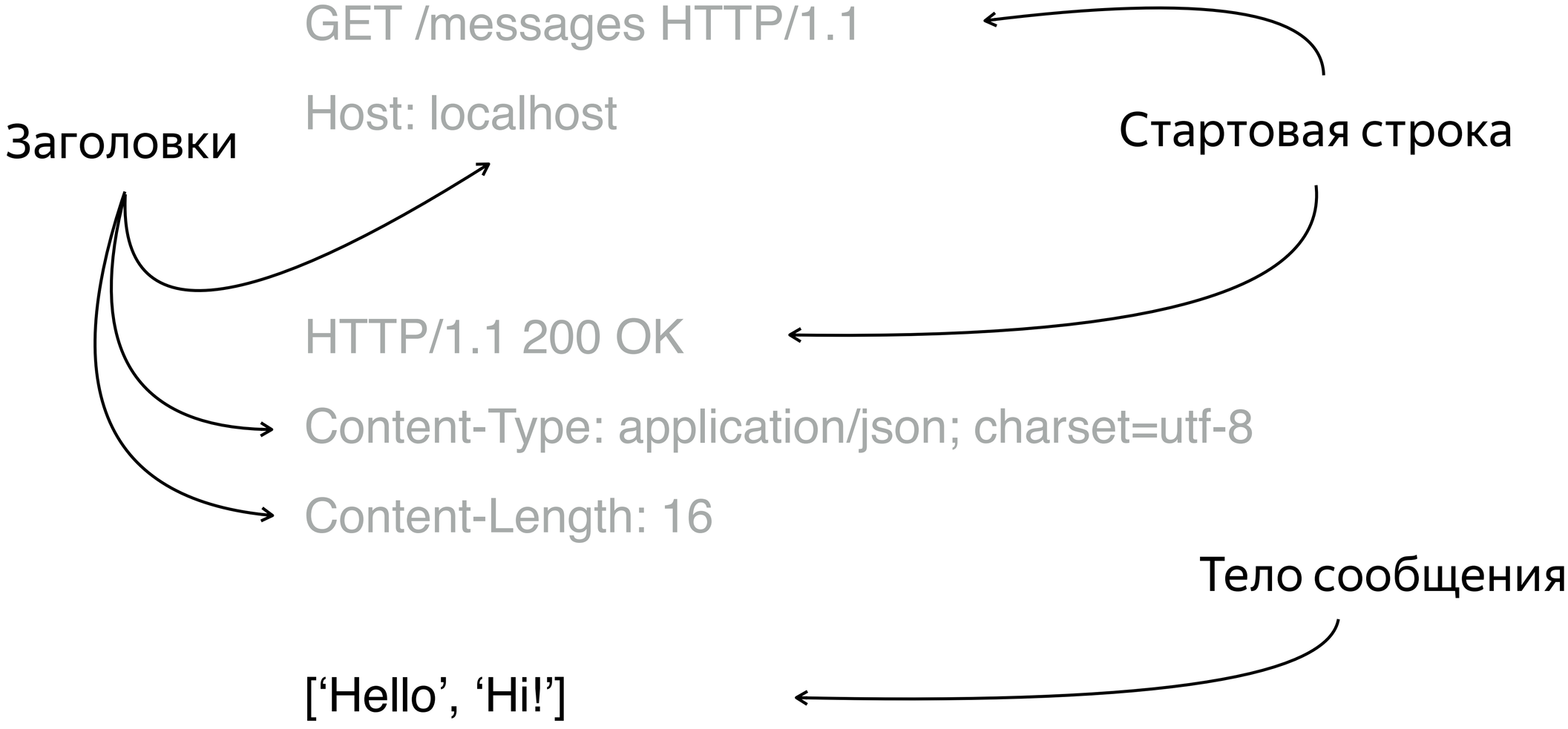
Content-Length: 16

[‘Hello’, ‘Hi!’]

Структура протокола



Структура протокола



Методы запроса и коды ответа

GET /messages HTTP/1.1

Host: localhost

HTTP/1.1 200 OK

Content-Type: application/json; charset=utf-8

Content-Length: 16

[‘Hello’, ‘Hi!’]

Методы запроса и коды ответа

GET – read PUT – update

POST – create DELETE – delete

HTTP/1.1 200 OK

Content-Type: application/json; charset=utf-8

Content-Length: 16

[‘Hello’, ‘Hi!’]

Методы запроса и коды ответа

GET – read PUT – update

POST – create DELETE – delete

200..<300 – Success

300..<400 – Redirection

400..<500 – Client errors

500..<600 – Server errors

Как загрузить картинку?

Работа с URLSession

1. Сформировать URL (URL)
2. Создать HTTP-запрос (URLRequest)
3. Настроить делегата (URLSessionDelegate, ...)
4. Создать конфигурацию (URLSessionConfiguration)
5. Создать сессию (URLSession)
6. Создать задачу (URLSessionTask, ...)

Работа с URLSession

1. Сформировать URL (URL)
2. Создать HTTP-запрос (URLRequest)
3. Настроить делегата (URLSessionDelegate, ...)
4. Создать конфигурацию (URLSessionConfiguration)
5. Создать сессию (URLSession)
6. Создать задачу (URLSessionTask, ...)

Формирование URL

```
let url = URL(string: "http://localhost/messages")!
```

Формирование URL

```
var urlComponents = URLComponents()

urlComponents.scheme = "http"
urlComponents.host = "localhost"
urlComponents.path = "/messages"
urlComponents.queryItems = [
    URLQueryItem(name: "user", value: "Vasiliy Pupkin"),
    URLQueryItem(name: "limit", value: "10"),
]

let url = urlComponents.url!
// http://localhost/messages?user=Vasiliy%20Pupkin&limit=10
```

Создание задачи

```
let task = URLSession.shared.dataTask(with: url) {
    data, response, error in

    DispatchQueue.main.async {
        self.label.text = "ok"
    }
}

task.resume()
```

Создание задачи

```
let task = URLSession.shared.dataTask(with: url) {
    data, response, error in

    DispatchQueue.main.async {
        self.label.text = "ok"
    }
}

task.resume()
```

Создание задачи

```
let task = URLSession.shared.dataTask(with: url) {
    data, response, error in

    DispatchQueue.main.async {
        self.label.text = "ok"
    }
}

task.resume()
```

Создание задачи

```
let task = URLSession.shared.dataTask(with: url) {
    data, response, error in

    DispatchQueue.main.async {
        self.label.text = "ok"
    }
}

task.resume()
```

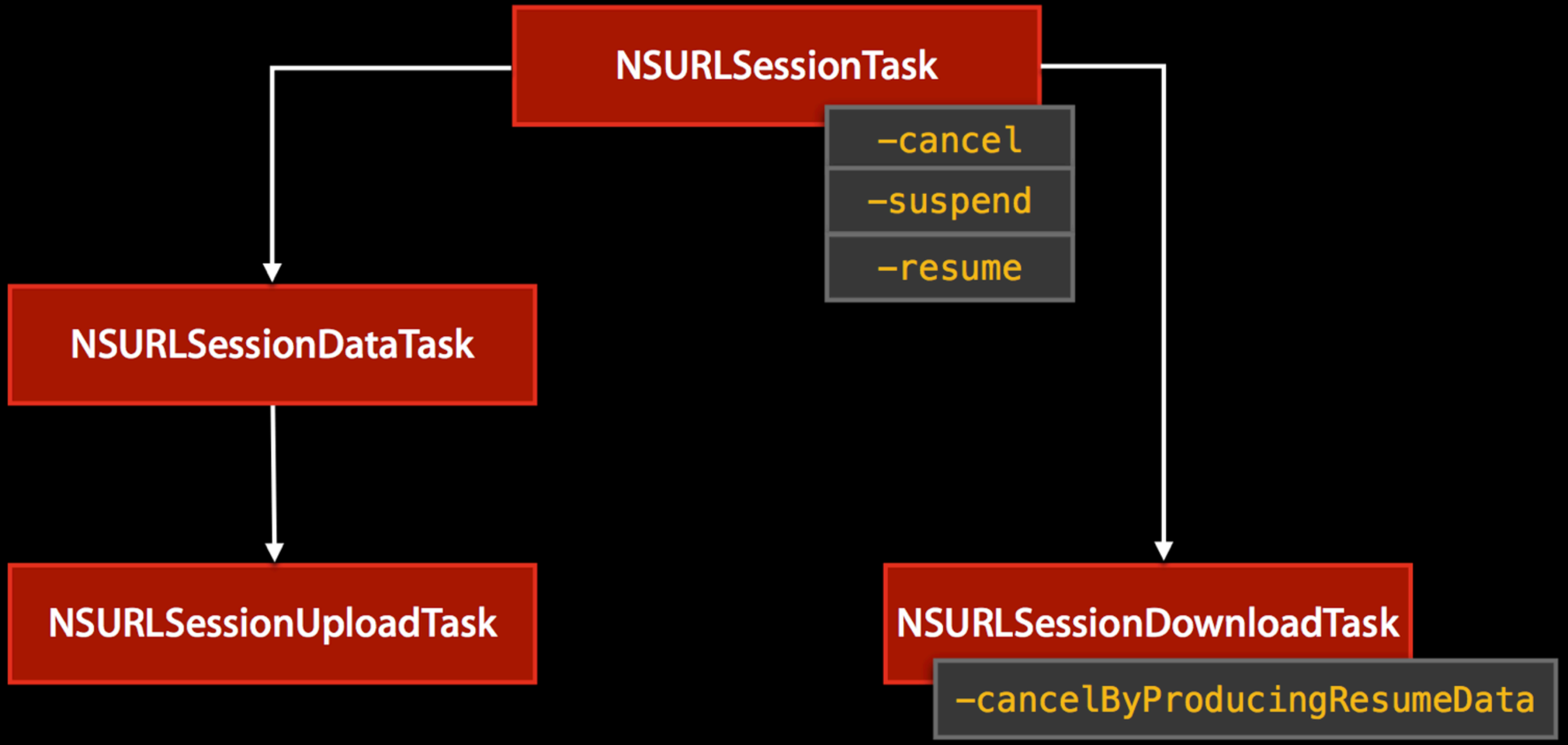
Создание задачи

```
let task = URLSession.shared.dataTask(with: url) {
    data, response, error in

    DispatchQueue.main.async {
        self.label.text = "ok"
    }
}

task.resume()
```

NSURLSessionTask



Обработка ошибок

```
if let error = error {
    print(error.localizedDescription)
    return
}

if let response = response as? HTTPURLResponse {
    switch response.statusCode {
    case 200..<300: break
    default:
        print("Status: \(response.statusCode)")
    }
}
```

Обработка ошибок: ошибки соединения

```
if let error = error {
    print(error.localizedDescription)
    return
}

if let response = response as? HTTPURLResponse {
    switch response.statusCode {
    case 200..<300: break
    default:
        print("Status: \(response.statusCode)")
    }
}
```

Обработка ошибок: ошибки приложения

```
if let error = error {
    print(error.localizedDescription)
    return
}

if let response = response as? HTTPURLResponse {
    switch response.statusCode {
    case 200..<300: break
    default:
        print("Status: \(response.statusCode)")
    }
}
```

Application Transport Security

- Политика безопасности, требующая работать с сетевыми ресурсами по защищённому соединению (HTTPS)
- Запрещает устаревшие и небезопасные протоколы и алгоритмы шифрования
- Обязательна к соблюдению (почти)
- Гибко настраивается

Конфигурация ATS в plist

NSAppTransportSecurity

- NSAllowsArbitraryLoads
- NSAllowsArbitraryLoadsForMedia
- NSAllowsArbitraryLoadsInWebContent
- NSAllowsLocalNetworking
- NSExceptionDomains

It's a DEMO time

HTTP GET

Как сделать POST-запрос?

HTTP-методы

GET – чтение

POST – создание

PUT – изменение

DELETE – удаление

OPTIONS, HEAD, PATCH, …

Создание ресурса

POST http://localhost/api/message

Host: localhost

Тело запроса

POST http://localhost/api/message

Host: localhost

Content-Type: text/plain

Hello world

Тело запроса

POST http://localhost/api/message

Host: localhost

Content-Type: application/x-www-form-urlencoded

text=Hello%20world

Тело запроса

POST http://localhost/api/message

Host: localhost

Content-Type: application/json

{"message": "Hello world"}

Работа с URLSession

1. Сформировать URL (URL)
2. Создать HTTP-запрос (URLRequest)
3. Настроить делегата (URLSessionDelegate, ...)
4. Создать конфигурацию (URLSessionConfiguration)
5. Создать сессию (URLSession)
6. Создать задачу (URLSessionTask, ...)

Работа с URLSession

1. Сформировать URL (URL)
2. Создать HTTP-запрос (URLRequest)
3. Настроить делегата (URLSessionDelegate, ...)
4. Создать конфигурацию (URLSessionConfiguration)
5. Создать сессию (URLSession)
6. Создать задачу (URLSessionTask, ...)

Создание HTTP-запроса

```
var request = URLRequest(url: url)

request.httpMethod = "POST"

request.setValue("application/json",
                 forHTTPHeaderField: "Content-Type")

request.httpBody = try! JSONSerialization.data(
                   withJSONObject: ["message": "Hello"])
```

Создание задачи из запроса

```
let url = URL(string: "http://localhost:8080/message")!
var request = URLRequest(url: url)

let task = session.dataTask(with: request) {
    data, response, error in

    if error != nil {
        print("ok")
    }
}

task.resume()
```

It's a DEMO time

HTTP POST

Как настроить кэширование?

Работа с URLSession

1. Сформировать URL (URL)
2. Создать HTTP-запрос (URLRequest)
3. Настроить делегата (URLSessionDelegate, ...)
4. Создать конфигурацию (URLSessionConfiguration)
5. Создать сессию (URLSession)
6. Создать задачу (URLSessionTask, ...)

Работа с URLSession

1. Сформировать URL (URL)
2. Создать HTTP-запрос (URLRequest)
3. Настроить делегата (URLSessionDelegate, ...)
4. Создать конфигурацию (URLSessionConfiguration)
5. Создать сессию (URLSession)
6. Создать задачу (URLSessionTask, ...)

URLSessionConfiguration

- Cache, Cookies, Credential stores
- Cell usage, network service type
- Number of connections
- Resource and network timeouts
- TLS protocols
- HTTP proxies, cookies, pipelining, headers
- Protocol handlers

Типовые конфигурации

- URLSessionConfiguration.default
  - Настройки по умолчанию
- URLSessionConfiguration.ephemeral
  - Cache, Credentials и Cookie хранятся в памяти
- URLSessionConfiguration.background(withIdentifier: String)
  - Для запросов в фоновом режиме

Создание конфигурации

```
let config = URLSessionConfiguration.default

config.urlCache = URLCache(memoryCapacity: 100 * 1024 * 1024,
                           diskCapacity: 100 * 1024 * 1024,
                           diskPath: nil)

config.requestCachePolicy = .returnCacheDataElseLoad
```

Политики кэширования

- useProtocolCachePolicy
  - Согласно HTTP-заголовкам
- reloadIgnoringLocalCacheData
  - Только сеть
- returnCacheDataElseLoad
  - Кэш, потом сеть
- returnCacheDataDontLoad
  - Только кэш

Создание сессии

```
lazy var session: URLSession = {
    let configuration = URLSessionConfiguration.default

    configuration.requestCachePolicy =
                                .returnCacheDataElseLoad

    return URLSession(configuration: configuration)
}()
```

It's a DEMO time

HTTP Caching

Как управлять таймаутами и отменять запросы?

It's a DEMO time

Timeouts

Как реализовать аутентификацию?

Basic Authorization

Basic Authorization

GET /secret HTTP/1.1

Host: localhost

Basic Authorization

GET /secret HTTP/1.1

Host: localhost

HTTP/1.1 401 Unauthorized

WWW-Authenticate: Basic realm="example"

Basic Authorization

GET /secret HTTP/1.1

Host: localhost

HTTP/1.1 401 Unauthorized

WWW-Authenticate: Basic realm="example"

GET /secret HTTP/1.1

Host: localhost

Authorization: Basic YWRtaW46cGFzc3dvcmQ=

Basic Authorization

```
GET /secret HTTP/1.1
Host: localhost
```

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Basic realm="example"
```

```
GET /secret HTTP/1.1
Host: localhost
Authorization: Basic YWRtaW46cGFzc3dvcmQ=
```

```
HTTP/1.1 200 OK
```

Работа с URLSession

1. Сформировать URL (URL)
2. Создать HTTP-запрос (URLRequest)
3. Настроить делегата (URLSessionDelegate, ...)
4. Создать конфигурацию (URLSessionConfiguration)
5. Создать сессию (URLSession)
6. Создать задачу (URLSessionTask, ...)

Работа с URLSession

1. Сформировать URL (URL)
2. Создать HTTP-запрос (URLRequest)
3. Настроить делегата (URLSessionDelegate, ...)
4. Создать конфигурацию (URLSessionConfiguration)
5. Создать сессию (URLSession)
6. Создать задачу (URLSessionTask, ...)

Настройка делегата

```
extension ViewController: URLSessionDataDelegate {
    func urlSession(
        _ session: URLSession,
        task: URLSessionTask,
        didReceive challenge: URLAuthenticationChallenge,
        completionHandler: (AuthChallengeDisposition,
                            URLCredential?) -> Void) {

        completionHandler(.useCredential, credential)
    }
}
```

Настройка делегата

```
extension ViewController: URLSessionDataDelegate {
    func urlSession(
        _ session: URLSession,
        task: URLSessionTask,
        didReceive challenge: URLAuthenticationChallenge,
        completionHandler: (AuthChallengeDisposition,
                            URLCredential?) -> Void) {

        completionHandler(.useCredential, credential)
    }
}
```

Настройка делегата

```
extension ViewController: URLSessionDataDelegate {
    func urlSession(
        _ session: URLSession,
        task: URLSessionTask,
        didReceive challenge: URLAuthenticationChallenge,
        completionHandler: (AuthChallengeDisposition,
                            URLCredential?) -> Void) {

        completionHandler(.useCredential, credential)
    }
}
```

completionHandler(AuthChallengeDisposition)

- performDefaultHandling
  - Сделать вид, что ничего не происходит
- cancelAuthenticationChallenge
  - Отказаться от этой идеи
- useCredential
  - Указать реквизиты доступа

Создание сессии

```
extension ViewController: URLSessionDataDelegate {
    lazy var session: URLSession = {
        return URLSession(configuration: .default,
                          delegate: self,
                          delegateQueue: .main)
    }()
}
```

Создание задачи

```
let url = URL(string: "http://localhost:8080/secret")!
let task = session.dataTask(with: url)

task.resume()
```

It's a DEMO time

HTTP Auth

Сегодня мы узнали

1. Как сформировать URL и создать HTTP-запрос
2. Какие возможности предоставляет конфигурация сессии
3. Как работать с кэшом и авторизацией

Вопросы?

Задача

Общие требования (Вариант 1 - Простой)

- Реализовать получение списка постов из VK, иметь возможность выбрать и добавить один из них в заметки
- Реализовать постинг заметки на стену в VK (через HTTP (!))
- Усложнений нет

Общие требования (Вариант 2 - Сложный)

〉 Требуется реализовать загрузку и сохранение заметок на сервер.

〉 Базовая задача - реализовать все необходимые запросы к бэкенду. Вызывать их через созданные в предыдущей лекции операции. Обработать ошибки. Авторизация через Bearer.

〉 Усложнение 1: Реализовать авторизацию через Яндекс.Паспорт и OAuth.

〉 Усложненение 2: В случае ошибок бэкенда изменяющие операции повторить через определенный промежуток времени до тех пор, пока не выполнится. Обеспечить правильную последовательность повтора.

Базовая задача

〉 Базовая задача - реализовать все необходимые запросы к бэкенду. Вызывать их через созданные в предыдущей лекции операции. Обработать ошибки. Авторизация через Bearer.

〉 Описание бэкенда тут - https://yadi.sk/i/yKp7dUYX3JGcMv

〉 Bearer токены в личку в телеграме. Бэкенд написан мной и при необходимости снова поднимается на notes.mrdekk.ru

Усложение 1 (*)

〉 Реализовать авторизацию через Яндекс.Паспорт и OAuth.

〉 Описание API и OAuth Паспорта - https://tech.yandex.ru/oauth/

〉 OAuth токен должен быть получен средствами паспорта.

〉 Бэкенд умеет ходить в паспорт и проверять токен

Усложение 2 (**)

- В случае ошибок бэкенда изменяющие операции повторить через определенный промежуток времени до тех пор, пока не выполнится. Обеспечить правильную последовательность повтора.
- Бэкенд периодически стреляет учебными 500'ками

Контакты

Малых Денис

iOS Developer

 mrdekk@yandex.ru

 @mrdekk

 mrdekk

Спасибо за внимание