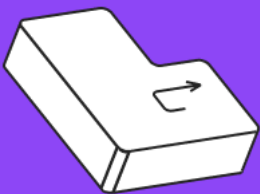




Работа с сетью

Разработка мобильного приложения на основе языка Swift



Оглавление

Введение	3
Термины, используемые в лекции	4
Работа с сетью	5
URL	5
HTTP Request	5
URLSession	6
Маппинг json	11
Codable	12
Decodable и Encodable	14
CodingKeys	14
try - do - catch	15
JSONDecoder	16
Post	18
Alamofire	19
CocoaPods	19
get	21
post	22
AppTransportSecurity	23
Что можно почитать еще?	24
Используемая литература	25

Введение

В ходе прошлой лекции вы познакомились с основами создания интерфейса и уже умеете размещать элементы на экране, знаете, что такое UITableView и UICollectionView. В ходе текущей вы узнаете, как работать с сетью.

На этой лекции вы узнаете:

- Что такое URLSession
- Codable
- Про конструкцию try - do - catch
- Что такое Alamofire

Термины, используемые в лекции

URLSession - API, которое предоставляет Apple. Отвечает за отправку и прием запросов

JSONDecoder - объект, который декодирует экземпляры типа данных из объектов JSON

CocoaPods – программа, написанная на Ruby, которая берет на себя управление сторонними библиотеками, добавленными в проект

Alamofire - библиотека для работы с сетью от стороннего разработчика

Работа с сетью

Мы уже узнали, как создавать свой интерфейс, как расположить элементы на экране. Но чем же их наполнять? Мы же не хотим хардкодить данные, да и например, если мы решим создать приложение, которое показывает текущую погоду, то при захардкоженных данных информация будет всегда одна и такое приложение становится бесполезно. Поэтому нам необходимо получать данные из сети. Для начала познакомимся с тем, что такое url.

URL

URL – это указатель на местонахождение ресурса. Давайте рассмотрим следующий URL: "https://kudago.com/public-api/v1.4/locations/?lang=ru&fields=name,coords" и разберем его, в данном случае:

1. https - схема
2. kudago.com - хост, в данном случае адрес сервера
3. public-api/v1.4/locations - путь для получения городов и их координат
4. lang=ru&fields=name,coords - параметры запроса, в данном случае их два. Первый с именем lang и значением ru, а второй с именем fields и параметром name,coords

Также в url может быть указано следующее:

1. Логин:пароль или токен аутентификации, необходимы для доступа к ресурсу
2. Порт - ресурсы могут находиться на одном хосте, но на разных портах. Обычно для веб-сайтов используется 80 порт по умолчанию.
3. Якорь - указывает конкретное место страницы, которое необходимо установить.

HTTP Request

Теперь мы перейдем к HTTP Request. HTTP – протокол передачи данных, работающих поверх TCP. Можно сказать, это текстовый документ, состоящий из нескольких частей. Обычно взаимодействие с сервером происходит следующим образом:

1. Клиент отправляет запрос на сервер
2. Сервер обрабатывает запрос
3. Сервер отправляет клиенту ответ

Существует два основных типа запроса: get и post. Get - получить данные, post - отправить данные. Далее идет тело запроса. Оно может быть пустым, а может содержать некую информацию, например, массив параметров.

Отдельно стоит сказать про параметры, мы будем часто их использовать. Параметры могут быть переданы или через url, или через тело запроса. Обязательно обращайтесь внимание на документацию, особенно на параметры, которые необходимо передать. В зависимости от них может меняться ответ от сервера.

Теперь вспомним про нашу ссылку: “https://kudago.com/public-api/v1.4/locations/?lang=ru&fields=name,coords”, в начале указано https и именно https мы будем чаще всего использовать. Но что делать, если у нас http? По умолчанию в приложении нельзя делать запросы по http, такие запросы будут вызывать ошибку. Но если все же почему-то очень нужно сделать http-запрос, надо их разрешить. Для этого необходимо открыть файл **info.plist** и добавить в него параметр App Transport Security Settings, а в него – Allow Arbitrary Loads со значением YES.

URLSession

URLSession - api, которое предоставляет Apple. Представляет собой объект, который координирует группу связанных сетевых задач для передачи данных. Отвечает за отправку и прием http запросов.

Каждый объект URLSession содержит экземпляр класса URLSessionConfiguration, он позволяет конфигурировать параметры сессии.

Каждый запрос содержит перечень сетевых задач, который представлен классом URLSessionTask. данный класс содержит несколько подклассов:

1. URLSessionDataTask - для получения данных, является подклассом URLSessionTask
2. URLSessionUploadTask - для выгрузки файлов с устройства на сервер, является подклассом URLSessionDataTask
3. URLSessionDownloadTask - для загрузки файлов с сервера на устройство, является подклассом URLSessionTask

URLSession содержит следующие методы:

- cancel - отмена задания
- resume - возобновление задания

- suspend - приостановка задания

Итак, давайте попробуем получить данные из сети и вывести их в консоль. Создадим функцию `getData()`, внутри которой создадим константу, в которой будет необходимый url и сессию по умолчанию при помощи `URLSession.shared`.

```
1 extension ViewController {
2     func getData() {
3         // url
4         let url = URL(string: "https://kudago.com/public-
    api/v1.4/locations/?lang=ru&fields=name,coords")
5         // Создаем сессию
6         let session = URLSession.shared
7     }
8 }
```

Далее мы создаем задачу для старта загрузки, в замыкании в круглых скобках указываем имена, которые будем использовать для полученных данных от сервера. Внутри замыкания данные преобразуем в json и выводим в консоль.

```
1 func getData() {
2     // url
3     let url = URL(string: "https://kudago.com/public-
    api/v1.4/locations/?lang=ru&fields=name,coords")
4     // Создаем сессию
5     let session = URLSession.shared
6
7     // Создаем задачу
8     let task = session.dataTask(with: url!) { (data, response,
    error) in
9
10        // Преобразуем в json
11        let json = try? JSONSerialization.jsonObject(with: data!,
    options: JSONSerialization.ReadingOptions.allowFragments)
12        // Выводим в консоль
13        print(json) }
14 }
```

В самом конце запустим задачу

```
1 task.resume()
```

Мы еще не рассмотрели что такое `JSONSerialization`. Что же это? `JSONSerialization` - это объект, который выполняет преобразование между JSON и эквивалентными объектами Foundation. После запуска кода в консоль будет выведено следующее:

```
Optional(<__NSArrayI 0x600002722920>({
    coords = {
        lat = "56.838607";
        lon = "60.60551400000001";
    };
    name =
        "\U00415\U0043a\U00430\U00442\U00435\U00440\l
        0438\U0043d\U00431\U00443\U00440\U00433";
},
{
    coords = {
        lat = "56.010569";
        lon = "92.85254500000001";
    };
    name =
        "\U0041a\U00440\U00430\U00441\U0043d\U0043e\l
        044f\U00440\U00441\U0043a";
},
{
    coords = {
        lat = "45.023876999999996";
        lon = "38.970157";
    };
});
```

В итоге мы получаем имя и координаты.

Зачастую в “боевых” приложениях сессии создаются в отдельных классах. Давайте рассмотрим, как создать свою собственную сессию, а не по умолчанию. Для этого сначала мы создаем конфигурацию для сессии при помощи `URLSessionConfiguration`. Выбираем `default`, то есть по умолчанию.

Создать конфигурацию для сессии можно следующими способами:

1. Свойство `default` - конфигурация по умолчанию
2. Свойство `ephemeral` - конфигурация сеанса, в которой не используется постоянное хранилище для кэшей, файлов `cookie` или учетных данных.
3. При помощи функции `background (withIdentifier: String)` -> `URLSessionConfiguration` - она создает объект конфигурации сеанса, который позволяет выполнять скачивание или загрузку HTTP и HTTPS в фоновом режиме.

Итак, после того, как мы создали константу, в которой хранится конфигурация сессии, нам необходимо создать собственную сессию, для этого мы при инициализации сессии передаем в инициализатор конфигурацию


```

1 func getData() {
2     // url
3     let url = URL(string: "https://kudago.com/public-
api/v1.4/locations/?lang=ru&fields=name,coords")
4     // Создаем конфигурацию
5     let configuration = URLSessionConfiguration.default
6     // Создаем сессию
7     let session = URLSession(configuration: configuration)
8
9     // Создаем задачу
10    let task = session.dataTask(with: url!) { (data, response,
error) in
11
12        // Преобразуем в json
13        let json = try? JSONSerialization.jsonObject(with: data!,
options: JSONSerialization.ReadingOptions.allowFragments)
14        // Выводим в консоль
15        print(json)
16    }
17    task.resume()
18 }

```

После запуска проекта в консоли мы увидим все тоже самое, что и первый раз, но теперь у нас есть своя собственная сессия.

```

Optional(<__NSArrayI 0x600001b46610>(
{
    coords = {
        lat = "56.838607";
        lon = "60.60551400000001";
    };
    name =
        "\U00415\U0043a\U00430\U00442\U00435\U00440\U00438\U0043d\U00431\U00443\U00440\U00433";
},
{
    coords = {
        lat = "56.010569";
        lon = "92.85254500000001";
    };
    name =
        "\U0041a\U00440\U00430\U00441\U0043d\U0043e\U00433";
},
)

```

Теперь давайте получше рассмотрим параметры data, response и error, попробуем их все вывести в консоль

```

1 let task = session.dataTask(with: url!) { (data, response, error) in
2
3         print(data)
4         print(response)
5         print(error)
6     }

```

```

Optional(900 bytes)
Optional(<NSHTTPURLResponse: 0x600003ec1000> {
    URL:
    https://kudago.com/public-api/v1
    .4/locations/?lang=ru&fields=name,coords }
    { Status Code: 200, Headers {
        "Content-Language" =
            (
                ru
            );
        "Content-Length" =
            (
                900
            );
        "Content-Type" =
            (
                "application/json",
                "application/json; charset=UTF-8"
            );
        Date =
            (
                "Thu, 19 Jan 2023 09:06:13 GMT"
            );
        Server =
            (
                ...
            );
        Set-Cookie =
            (
                "prfdcty=msk; Path=/",
                "flavour=full; Path=/",
                "isAuthenticated=0; Path=/",
                "isConfirmed=0; Path=/",
                "isStaff=0; Path=/",
                "sessionId=9xe3cornbl9bvdK8wnsqji0dq2y!fuhz; expires=Sun, 14 Jan 2024 09:06:13 GMT; HttpOnly; Max-Age=31104000; Path=/; SameSite=Lax"
            );
        Vary =
            (
                "Accept-Language, Cookie"
            );
        allow =
            (
                "GET, HEAD, OPTIONS"
            );
    } })
nil

```

print(data) выведет нам Optional(900 bytes), print(response) выведет ответ сервера, а print(error) выведет nil, так как ошибок нет.

Итак:

- data - данные, возвращаемые сервером
- response - объект, предоставляющий метаданные ответа, такие как заголовки HTTP и код состояния. Если делается запрос HTTP или HTTPS, возвращаемый объект фактически является объектом HTTPURLResponse.
- error - объект ошибки, указывающий, почему запрос не удался, или nil, если запрос был успешным.

Попробуем получить ошибку, для этого из url уберем .com и выведем результат в консоль.

```
1 print("Error")
2 print(error?.localizedDescription)
```

После запуска мы увидим ошибку, что не удастся найти сервер с таким именем хоста.

Error

Optional("A server with the specified hostname could not be found.")

Теперь у нас все отлично, мы можем получать данные с сервера, но что нам с ними делать? У константы json тип Any, еще и опционал. Не правда ли было бы классно, если бы вместо Any была какая-то структура или класс?

Маппинг json

У нас уже есть json объект, который мы можем преобразовать в необходимую структуру. Давайте рассмотрим, что такое json-структура. Json- структура это пара ключ - значение, очень похоже на словарь, но в данном случае в качестве ключа может быть только строка. Давайте рассмотрим ответ, который нам отдает сервер, для этого мы можем воспользоваться Postman или вбить url в строку в браузере. Мы получим следующее:

```
[{"name":"Екатеринбург","coords":{"lat":56.838606999999996,"lon":60.605514000000001}}, {"name":"Красноярск","coords":{"lat":56.010569,"lon":92.852545}}, {"name":"Краснодар","coords":{"lat":45.023876999999996,"lon":38.970157}}, {"name":"Казань","coords":{"lat":55.7957929999999975,"lon":49.106584999999995}}, {"name":"Минск","coords":{"lat":53.906076999999996,"lon":27.554914}}, {"name":"Москва","coords":{"lat":55.753676,"lon":37.619898999999998}}, {"name":"Нижний Новгород","coords":{"lat":56.326886999999997,"lon":44.005985999999999}}, {"name":"Новосибирск","coords":{"lat":55.030198999999994,"lon":82.92043}}, {"name":"Онлайн","coords":{"lat":null,"lon":null}}, {"name":"Сочи","coords":{"lat":43.581508999999995,"lon":39.722881999999999}}, {"name":"Санкт-Петербург","coords":{"lat":59.939094999999996,"lon":30.315868}}]
```

Далее рассмотрим маленький кусок:

```
{"name":"Екатеринбург","coords":{"lat":56.838606999999996,"lon":60.60551400000001}},
```

В данном случае у нас есть ключ `name` и значение “Екатеринбург”, далее у нас идет ключ `“coords”`, а значение начинается с фигурных скобок. Это означает, что `coords` содержит целый объект. Внутри объекта есть два ключа со своими значениями. Значения представлены в числовом формате, например, `Double`. Итак, мы уже представляем из чего состоит наш `json`, пришло время превратить его в структуру!

Codable

`Codable` позволяет нам преобразовывать `json` в объекты языка `Swift`. Создадим структуру, в которой будут отражены все наши элементы `json`

В полученном `json` у нас есть явно повторяющиеся куски с `name` и `coords`, каждый такой кусок находится в фигурных скобках, а это значит, что один такой кусок может быть одной структурой. Создадим структуру `Town`

```
1 struct Town {  
2  
3 }
```

Внутри куска есть `name` и `coords`, добавим их в структуру.

```
1 struct Town {  
2     var name  
3     var coords  
4 }
```

Теперь нам необходимо определиться с типами для этих двух переменных. Из `json` видно, что в `name` хранится название города, значит у `name` будет тип `String`

```
1 struct Town {  
2     var name: String  
3     var coords  
4 }
```

Теперь посмотрим на `coords`, после идет фигурные скобки, а значит то, что внутри `coords` мы должны вынести в отдельную структуру, для этого создадим структуру `Coordinate`

```
1 struct Coordinate {  
2  
3 }
```

Внутри этой структуры расположим два свойства: lat и lon с типом Double.

```
1 struct Coordinate {  
2     var lat: Double  
3     var lon: Double  
4 }
```

И теперь укажем эту структуру в качестве типа для coords

```
1 struct Town {  
2     var name: String  
3     var coords: Coordinate  
4 }
```

Также укажем, что обе структуры соответствуют Codable. Если это указать только для Town, то в строке coords: Coordinate выскочит ошибка

```
1 struct Town: Codable { 2 ✖ Type 'Town' does not conform t...  
2     var name: String  
3     var coords: Coordinate  
4 }  
5  
6 struct Coordinate {  
7     var lat: Double  
8     var lon: Double  
9 }
```

Итак, обе структуры должны выглядеть следующим образом:

```

1 struct Town: Codable {
2     var name: String
3     var coords: Coordinate
4 }
5
6 struct Coordinate: Codable {
7     var lat: Double
8     var lon: Double
9 }

```

Decodable и Encodable

Если посмотреть на ошибку, которую мы получили, когда не указали Codable для Coordinate, то мы увидим, что там ни слова про Codable, только про Encodable и Decodable. Это потому, что на самом деле Codable - это псевдоним для Decodable и Encodable, то есть, когда мы указываем, что структура должна соответствовать Codable, мы имеем в виду, что она должна соответствовать и Decodable и Encodable.

Encodable - это тип, который может кодировать себя во внешнее представление.

Decodable - это тип, который может декодировать себя из внешнего представления.

То есть, если мы будем использовать структуру только для декодировки ответа полученного от сервера, то мы можем вместо Codable указать Decodable. А если подразумевается, что структура будет использоваться только для кодировки и отправки на сервер, то можем указать только Encodable.

CodingKeys

Обратите внимание, что имена для переменных структуры должны полностью совпадать с именами ключа json. Но что делать, если необходимо использовать другое имя? Здесь на помощь приходит CodingKeys. CodingKey - это тип, который можно использовать в качестве ключа для кодирования и декодирования. Например, в коде мы хотим, чтобы переменная в коде называлась townName, а не name. Тогда мы создаем две переменных: townName и coords

```

1 struct Town: Codable {
2     var townName: String
3     var coords: Coordinate
4 }

```

Затем мы создаем enum с именем CodingKeys. Это перечисление должно обязательно соответствовать CodingKey. Внутри перечисления необходимо указать кейсы с именами, которые мы ранее создали для переменных.

Затем мы также указываем String после перечисления, чтобы иметь возможность задать строковое значение. После кейса указываем имя из json, которое соответствует имени, которое мы выбрали

```
1 struct Town: Codable {
2     var townName: String
3     var coords: Coordinate
4
5     enum CodingKeys: String, CodingKey {
6         case townName = "name"
7         case coords
8     }
9 }
```

Теперь вернемся к функции getData(), в которой мы запрашиваем данные.

try - do - catch

Перед тем, как преобразовать json в нашу структуру нам необходимо узнать, что же такое try, указанное перед JSONSerialization и do/catch

Когда мы пишем код, мы можем увидеть методы с ключевым словом throws. Это указывает на то, что метод может вызвать ошибку, и рекомендуется корректно обработать ее, а также показать пользователю соответствующее сообщение об ошибке. Если вы не поймаете ошибку, ваше приложение может упасть.

Давайте разберемся с ключевыми словами:

- do — это ключевое слово запускает блок кода, содержащий метод, который потенциально может вызвать ошибку.
- try — используем это ключевое слово перед вызывающим методом. То есть строка try <метод> может звучать так: попытаться выполнить такой-то метод
- catch — если метод с try не работает и вызывает ошибку, выполнение попадет в этот блок catch. Здесь можно написать код, который будет выводить сообщение об ошибке

Когда мы используем try мы обязательно должны использовать и do/catch. Однако у нас еще есть try! и try? с которыми использовать do/catch не нужно

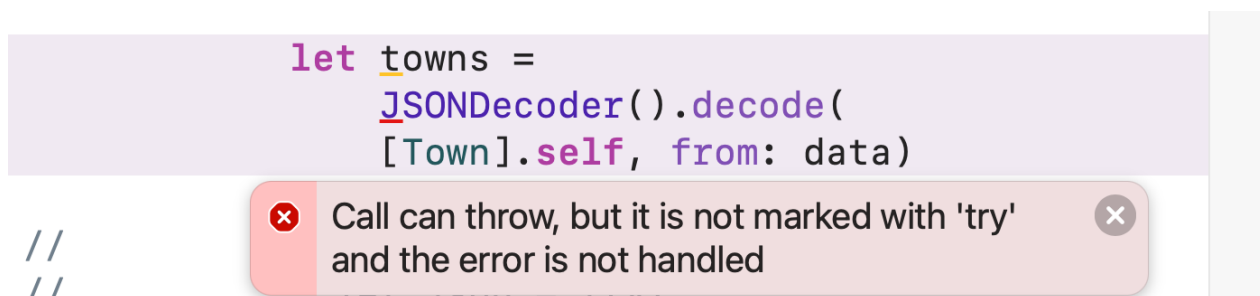
Мы можем использовать ключевое слово `try`? вместо `try`, тогда в случае ошибки мы просто получим `nil`

Если же мы полностью уверены, что ошибки не возникнет, то можно использовать `try!`

JSONDecoder

Теперь мы можем перейти к преобразованию `json` в структуру. Для преобразования мы используем `JSONDecoder`. `JSONDecoder` - объект, который декодирует экземпляры типа данных из объектов `JSON`.

Для начала создадим константу, в которой будет храниться результат декодировки. Мы сразу увидим ошибку, что метод может упасть, это означает, что нам необходимо использовать `try`



Чтобы добавить `try` создадим еще и блок `do/catch`. Сначала мы указываем `do`, затем создаем переменную, в которой декодируем `json`. Затем выводим полученные данные. В блоке `catch` мы выводим ошибку, если она будет

```
1 do {
2     let towns = try JSONDecoder().decode([Town].self, from: data)
3     print(towns)
4 } catch {
5     print(error)
6 }
```

После запуска проекта в консоль будет выведена ошибка о том, что ожидается `Double`, а получено `nil`.

```
valueNotFound(Swift.Double,
Swift.DecodingError.Context(codingPath:
[_JSONKey(stringValue: "Index 8",
intValue: 8), CodingKeys(stringValue:
"coords", intValue: nil),
CodingKeys(stringValue: "lat", intValue:
nil)], debugDescription: "Expected Double
value but found null instead.",
underlyingError: nil))
```

Давайте внимательно рассмотрим `json`, который получаем. Мы увидим, что есть координаты, в которых приходит не `Double`, а `null`


```
[{"name":"Екатеринбург","coords":{"lat":56.838606999999996,"lon":60.605514000000001}}, {"name":"Красноярск","coords":{"lat":56.010569,"lon":92.852545}}, {"name":"Краснодар","coords":{"lat":45.023876999999996,"lon":38.970157}}, {"name":"Казань","coords":{"lat":55.7957929999999975,"lon":49.106584999999995}}, {"name":"Минск","coords":{"lat":53.906076999999996,"lon":27.554914}}, {"name":"Москва","coords":{"lat":55.753676,"lon":37.619898999999998}}, {"name":"Нижний Новгород","coords":{"lat":56.326886999999997,"lon":44.005985999999999}}, {"name":"Новосибирск","coords":{"lat":55.030198999999994,"lon":82.92043}}, {"name":"Онлайн","coords":{"lat":null,"lon":null}}, {"name":"Сочи","coords":{"lat":43.581508999999995,"lon":39.722881999999999}}, {"name":"Санкт-Петербург","coords":{"lat":59.9390949999999966,"lon":30.315868}}]
```

Конкретно нас интересует следующий кусок:

```
{"name":"Онлайн","coords":{"lat":null,"lon":null}}.
```

Чтобы это поправить нам необходимо указать, что lat и lon могут быть опциональными, для этого после Double укажем знак вопроса. Это можно сделать и когда вместо значения может прийти nil, и когда пара ключ-значение может вообще не прийти.

```
1 struct Coordinate: Codable {
2     var lat: Double?
3     var lon: Double?
4 }
```

Запустим проект и увидим, что в консоль выводятся необходимые нам данные.

```
[TestProject.Town(townName:
    "Екатеринбург", coords:
    TestProject.Coordinate(lat:
    Optional(56.838606999999996),
    lon:
    Optional(60.605514000000001))),
TestProject.Town(townName:
    "Красноярск", coords:
    TestProject.Coordinate(lat:
    Optional(56.010569), lon:
    Optional(92.852545))),
TestProject.Town(townName:
```

Также обратите внимание, что мы можем не вызывать task.resume, как делаем это сейчас

```

1 func getData() {
2     let url = URL(string: "https://kudago.com/public-
    api/v1.4/locations/?lang=ru&fields=name,coords")
3     let configuration = URLSessionConfiguration.default
4     let session = URLSession(configuration: configuration)
5
6     let task = session.dataTask(with: url!) { (data, response,
    error) in
7         guard let data = data else {
8             return
9         }
10        do {
11            let towns = try JSONDecoder().decode([Town].self, from:
    data)
12            print(towns)
13        } catch {
14            print(error)
15        }
16    }
17    task.resume()
18 }

```

А указать resume после фигурной скобки, то есть после закрытия фигурной скобки замыкания:

```

1 func getData() {
2     let url = URL(string: "https://kudago.com/public-
    api/v1.4/locations/?lang=ru&fields=name,coords")
3     let configuration = URLSessionConfiguration.default
4     let session = URLSession(configuration: configuration)
5
6     let _ = session.dataTask(with: url!) { (data, response, error)
    in
7         guard let data = data else {
8             return
9         }
10        do {
11            let towns = try JSONDecoder().decode([Town].self, from:
    data)
12            print(towns)
13        } catch {
14            print(error)
15        }
16    }.resume()
17 }

```

Post

Мы уже научились работать с get запросами, давайте рассмотрим, что необходимо делать в случае с post запросом. Чтобы отправить post запрос нам необходимо

создать `URLRequest`. Мы передаем в инициализатор `url`, а затем устанавливаем `httpMethod`. Пример кода представлен ниже.

```
1 let request = URLRequest(url: url)
2 request.httpMethod = "POST"
```

Также для `URLRequest` можно задать, например, `httpBody` - это данные, отправляемые в виде тела сообщения запроса. Также не забудьте, что теперь в `dataTask` необходимо передать не `url`, а `request`

```
1 let _ = session.dataTask(with: request)
```

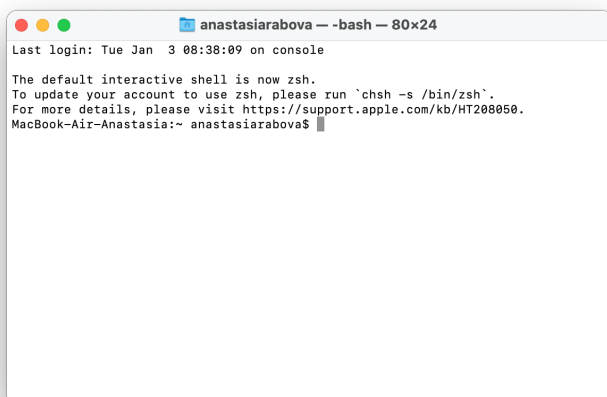
Alamofire

Мы уже научились создавать запросы при помощи `URLSession`. Теперь рассмотрим часто используемую библиотеку для работы с сетью от стороннего разработчика - `Alamofire`.

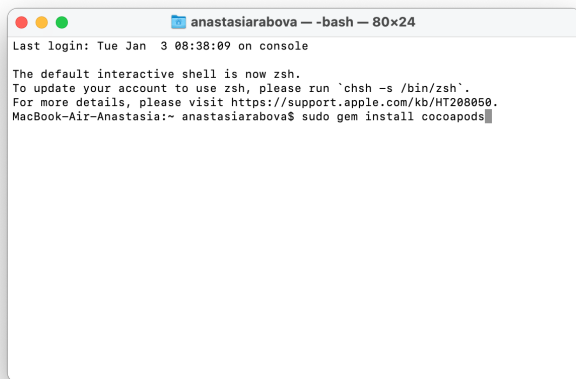
CocoaPods

Для добавления библиотеки в наш проект воспользуемся `CocoaPods`. `CocoaPods` – программа, написанная на Ruby, которая берет на себя управление сторонними библиотеками, добавленными в проект.

Установка `CocoaPods` происходит из терминала. Для начала необходимо открыть терминал.



Теперь вводим команду `sudo gem install cocoapods`, нажимаем `enter` и вводим пароль.



```
anastasiarabova -- -bash -- 80x24
Last login: Tue Jan 3 08:38:09 on console

The default interactive shell is now zsh.
To update your account to use zsh, please run 'chsh -s /bin/zsh'.
For more details, please visit https://support.apple.com/kb/HT208050.
MacBook-Air-Anastasia:~ anastasiarabova$ sudo gem install cocoapods
```

После завершения установки необходимо перейти в папку с проектом, для этого используем команду `cd`, после которой указываем путь к папке

```
MacBook-Air-Anastasia:~ anastasiarabova$ cd Desktop/TestProject
MacBook-Air-Anastasia:TestProject anastasiarabova$
```

Далее вводим команду `pod init`

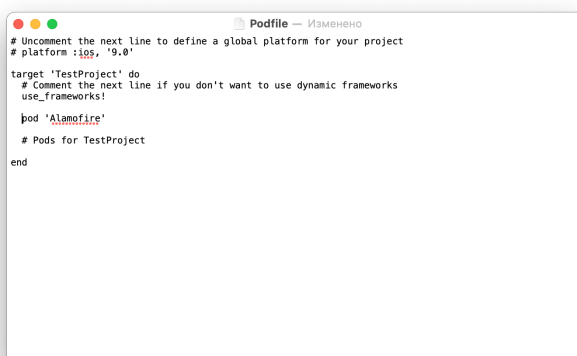
```
MacBook-Air-Anastasia:~ anastasiarabova$ cd Desktop/TestProject
MacBook-Air-Anastasia:TestProject anastasiarabova$ pod init
```

Теперь в папке проекта у нас появился файл `Podfile`, в нем перечислены зависимости, которые используются в проекте.



Podfile

Открываем файл и указываем `pod 'Alamofire'`



```
Podfile — 130x40
# Uncomment the next line to define a global platform for your project
# platform :ios, '9.0'

target 'TestProject' do
  # Comment the next line if you don't want to use dynamic frameworks
  use_frameworks!

  pod 'Alamofire'

  # Pods for TestProject
end
```

После добавления зависимости в файл, необходимо сохранить изменения и закрыть файл. Затем вводим в терминал команду `pod update` для обновления зависимостей. Обратите внимание, что команду необходимо запускать, находясь в папке проекта.

```
MacBook-Air-Anastasia:TestProject anastasiarabova$ pod update
```

После установки, в папке проекта, у нас появится новый файл, теперь проект необходимо запускать именно через него.



TestProject.xcworkspace

Чтобы работать с необходимой нам библиотекой, необходимо ее импортировать в файлах, где она будет использоваться. Для этого вверху файла указываем ключевое слово `import`, после которого необходимо указать нужную нам библиотеку, в данном случае `Alamofire`

```
import UIKit
import Alamofire|
```

get

Теперь напишем несколько запросов при помощи `Alamofire`. Напишем функцию `getDataWithAlamofire`. Для `get` запроса сначала указываем `AF`, затем `request`, в котором передаем строку с `url`, далее указываем `responseDecodable`, в котором указываем тип, в который декодируем. В нашем случае это массив `Town`. Затем передается замыкание, в котором мы прописываем действия, которые необходимо выполнить с ответом.

```

1 func getDataWithAlamofire() {
2     AF.request("https://kudago.com/public-api/v1.4/locations/?
   lang=ru&fields=name,coords").responseDecodable(of: [Town].self) { response
   in
3         print(response)
4     }
5 }

```

В логах мы можем увидеть ответ от сервера:

```

success([TestProject.Town(townName:
    "Екатеринбург", coords:
    TestProject.Coordinate(lat:
    Optional(56.838606999999996),
    lon:
    Optional(60.605514000000001))),
    TestProject.Town(townName:
    "Красноярск", coords:
    TestProject.Coordinate(lat:
    Optional(56.010569), lon:
    Optional(92.852545))),
    TestProject.Town(townName:
    "Краснодар", coords:
    TestProject.Coordinate(lat:
    Optional(45.023876999999996),

```

В Alamofire есть несколько основных функций:

- `.request`: Любой другой запрос HTTP, несвязанный с передачей файлов.
- `.download`: Загрузка файлов или возобновление загрузки, находящейся в процессе.
- `.upload`: Выгрузка (uploading) файлов с помощью многокомпонентных, потоковых, файловых методов и методов данных.

post

Теперь рассмотрим, как сделать post запрос. Для этого достаточно указать `method`:

`.post` после `request`, например

```
1 AF.request("<url>", method: .post) ... код ...
```

AppTransportSecurity

По умолчанию мы не можем сделать HTTP запрос в приложении, только HTTPS. Но если все-таки очень нужно сделать именно HTTP запрос - можно обойти данное ограничение. Для этого в info.plist нам необходимо добавить параметр App Transport Security Settings, уже в которой необходимо добавить Allow Arbitrary Loads со значением YES.

▼ App Transport Security Settings	⬆	Dictionary	(1 item)
Allow Arbitrary Loads	⬆	Boolean	YES

Что можно почитать еще?

1. [Alamofire](#)
2. [AppTransportSecurity](#)
3. [URLSession](#)

Используемая литература

1. <https://developer.apple.com/documentation/foundation/urlrequest>
2. <https://developer.apple.com/documentation/foundation/jsondecoder>
3. <https://developer.apple.com/documentation/swift/codable>
4. <https://developer.apple.com/documentation/foundation/urlsession>