



План занятия

1. [Задача](#)
2. [Протоколы](#)
3. [TCP/IP](#)
4. [Формы интеграции](#)
5. [HTTPS](#)
6. [Backend](#)
7. [Итоги](#)
8. [Домашнее задание](#)



ЗАДАЧА



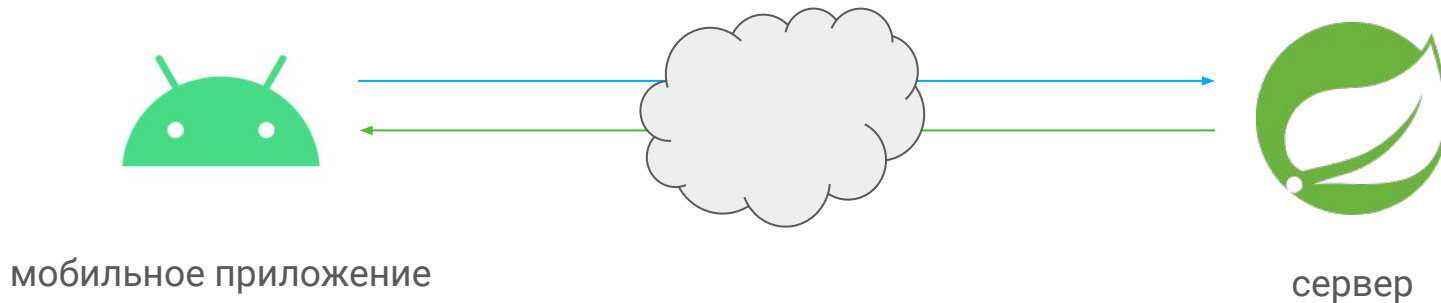
ЗАДАЧА

До этого мы с вами создавали приложения, способные хранить данные только локально и взаимодействовать только с самой системой Android.

Отдельной темой была работа с Push Notifications, которая позволила нам получать информацию извне (обычно Push'и присылает сервер).

ЗАДАЧА

Для реализации большинства приложений этого недостаточно и требуется прямая интеграция с сервером: приложение должно иметь возможность посылать запросы напрямую и получать ответы:



ЗАДАЧА

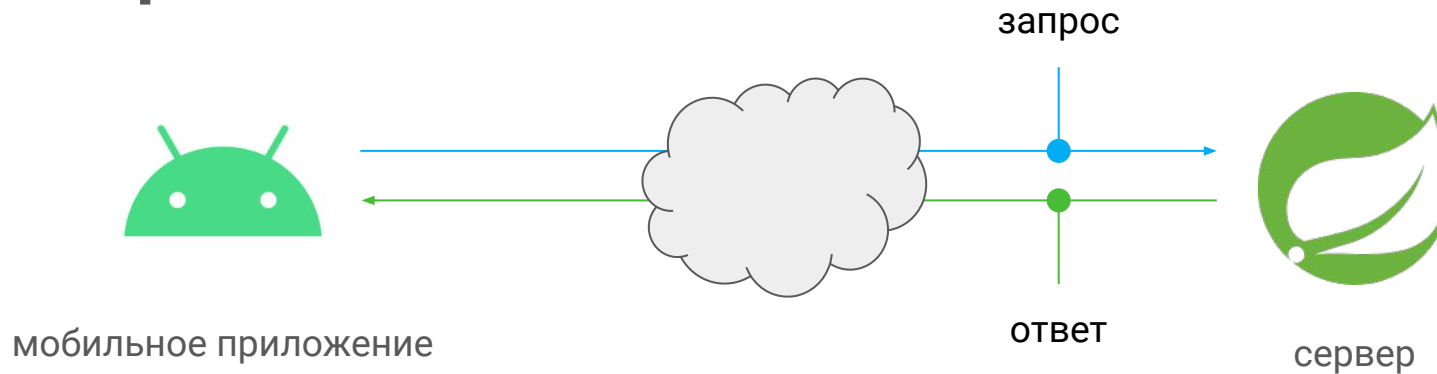
Поэтому сегодня наша задача:

- поговорить о сетевом взаимодействии в целом;
- рассмотреть распространённые схемы интеграции мобильного приложения и сервера.



ПРОТОКОЛЫ

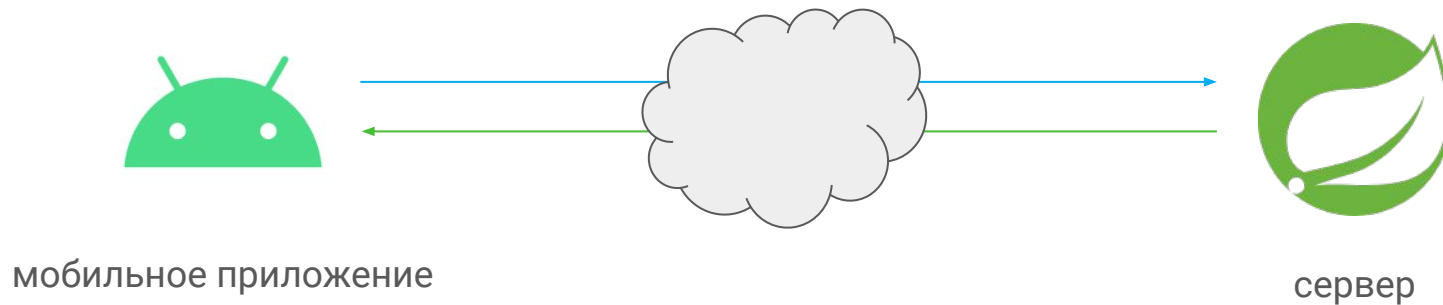
ОБЩАЯ СХЕМА



Сервер — это обычное приложение, которое может быть написано на Kotlin или любом другом языке.

Сервер работает на компьютере, подключенном к сети интернет и крайне желательно, чтобы сервер был постоянно доступен (т.е. не завершал свою работу) поскольку его ключевая задача — обслуживание запросов клиентов в режиме 24/7.

ОБЩАЯ СХЕМА



Таким образом, получается, что у нас есть два приложения: клиент и сервер, работающих на разных устройствах, которые должны обмениваться данными в рамках взаимодействия.

ОБЩАЯ СХЕМА

Всё это значит, что:


1. Должен быть выбран инструмент (транспорт) для доставки информации (набора байт) от одного приложения к другому.
2. Должны быть выбраны правила интерпретации этих байт (приложения должны понимать друг друга).



ОБЩАЯ СХЕМА

В большинстве случаев*, общение будет проходить по протоколу HTTP с использованием сети Интернет.

А в качестве формата обмена данными будут служить либо файлы (html, изображения, аудио, видео и другие), либо специальные форматы для обмена структурированными данными (например, JSON, который мы с вами проходили).



Примечание*: это не значит, что протокол HTTP — единственный, но он самый часто используемый.

ПРОТОКОЛ

Протокол — это просто правила общения двух сторон. Например, приезжая в аэропорт, перед посадкой в самолёт вы должны пройти предполетный досмотр:



Соответственно, если вы вдруг приедете без документов, то вы не выполните условия протокола, и сотрудники аэропорта (сторона, с которой вы взаимодействуете) вас не пропустят.



ПРОТОКОЛ

В мире приложений всё так же: «договорённости» устанавливаются на множестве уровней: от физического (какие электромагнитные сигналы пересылаются) до уровня приложений (в каком формате и какие данные передаются).



TCP/IP

OSI

Сетевое взаимодействие систем принято описывать с помощью 7-ми уровневой модели OSI:

OSI



Изображение из Wikipedia

OSI

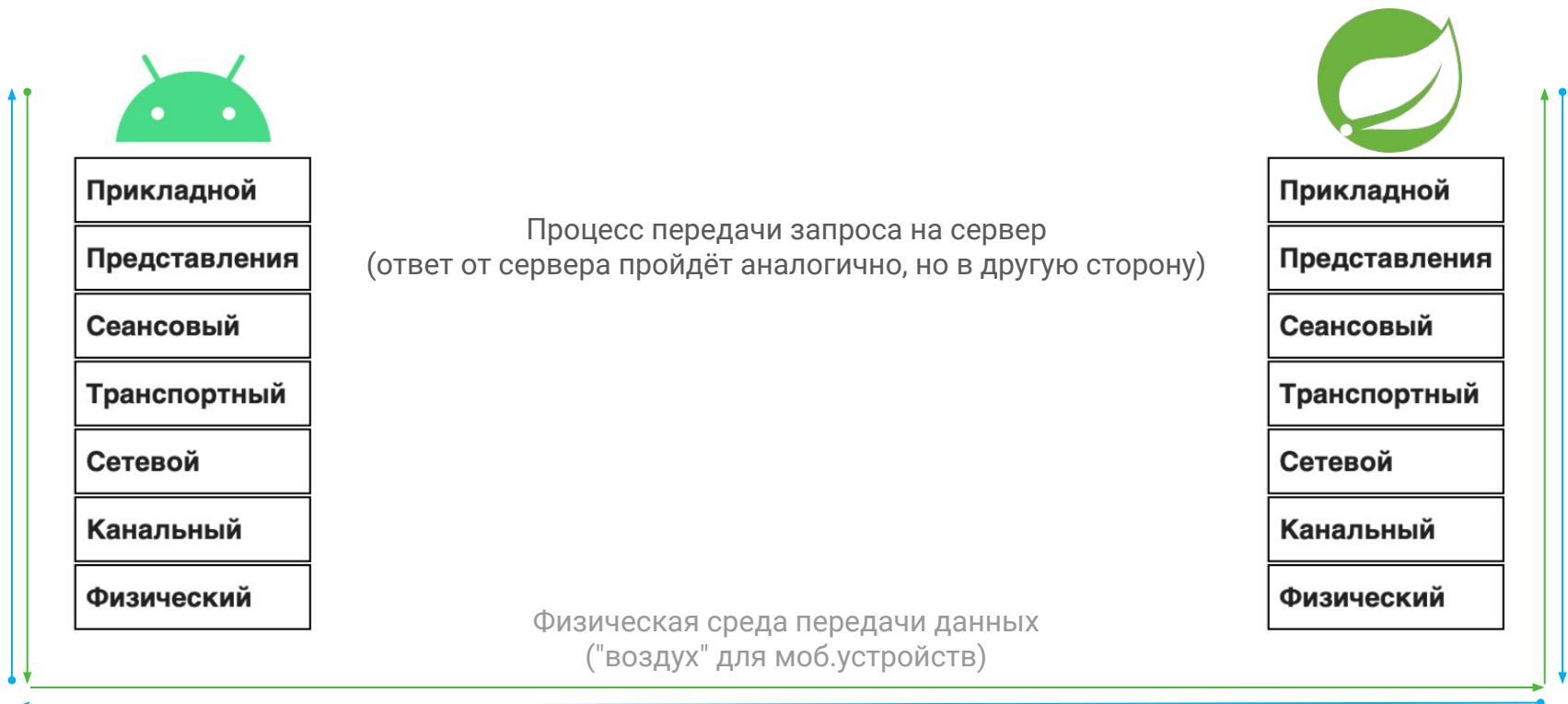
OSI (The Open Systems Interconnection model) — сетевая модель, описывающая, как две системы взаимодействуют между собой.

Эта модель определяет ряд уровней, каждый из которых отвечает за определённую функциональность:

- прикладной — данные в формате, понятном приложениям;
- представления — кодировка данных, сжатие, шифрование;
- сеансовый — сессия;
- транспортный — передача данных между конечными точками;
- сетевой — логическая адресация;
- канальный — физическая адресация;
- физический — передача по физическим каналам связи.

OSI

Когда мы передаём данные из нашего приложения по сети другому приложению, то данные последовательно проходят через все уровни, передаются получателю и запускается обратный процесс:



OSI

Q: Что значит «проходит через все уровни»?

A: Дело в том, что приложения оперируют термином «данные». Например, в нашем приложении – это посты. Но когда это всё передаётся по сети, все эти данные на каждом уровне разбиваются на «кусочки», и к каждому кусочку добавляются мета-данные соответствующего уровня.

Это как с бумажными письмами:

- вы пишете адрес и запаковываете само письмо;
- сотрудник почты клеит марки;
- и т.д.



OSI vs TCP/IP

Несмотря на то, что OSI это «эталонная модель», по которой принято описывать системы, на практике она не реализуется, а используется стек (набор) протоколов TCP/IP.

OSI vs TCP/IP



Изображение из Wikipedia



ТСР/IP


Сегодня нас будут интересовать только IP, ТСР и HTTP. Потому что нижние уровни определяют физическую передачу данных, и большую часть из них реализует оборудование (сетевая карта/WiFi-адаптер/4-5G модуль).



RFC

RFC (Request For Comments) – специальный тип документов, которые описывают стандарты, протоколы, форматы и т.д.

На все три протокола, которые мы будем рассматривать, есть свои RFC*.



Примечание*: RFC читать достаточно полезно, поскольку это первоисточник.



IP

RFC: <https://tools.ietf.org/html/rfc791>.

The internet protocol provides for transmitting blocks of data called datagrams from sources to destinations, where sources and destinations are **hosts identified by fixed length addresses**.

IP используется в сетях для передачи данных между хостами, которые идентифицируются адресами фиксированной длины.

Для нас хост – это просто машина (компьютер, моб.устройство или что-то ещё) у которой есть IP-адрес.

IP-АДРЕС

Есть две версии протокола, определяющие адреса разной длины:

- IPv4
- IPv6

Когда вы в командной строке набираете `ipconfig` (Windows), `id addr show` (Mac/Linux), то увидите адрес:

```
DNS-суффикс подключения . . . . . :  
Локальный IPv6-адрес канала . . . : fe80::254b:dc96:4072:a27c%3  
IPv4-адрес. . . . . : 192.168.0.102  
Маска подсети . . . . . : 255.255.255.0  
Основной шлюз. . . . . : 192.168.0.1
```

СПЕЦИАЛЬНЫЕ АДРЕСА

- **127.0.0.1 (::1)** – **localhost** (ip-адрес вашего хоста, даже если он не подключен к сети).
- **0.0.0.0 (::0)** – wildcard (означает, что будем принимать данные со всех ip-адресов, которые есть у нашего хоста).

Q: Почему для нас это важно?

A: Потому что в рамках курса вы будете запускать сервер локально (на вашем компьютере) и обращаться к нему из эмулятора/мобильного устройства.

ANDROID EMULATOR

Android Emulator дополнительно определяет [ряд фиксированных адресов](#):

- 10.0.2.2 – псевдоним для 127.0.0.1 вашего компьютера (на котором запущен сам эмулятор, 127.0.0.1 внутри эмулятора указывает на сам эмулятор).
- 10.0.2.15 – адрес самого эмулятора (в сети с хостом).

IP

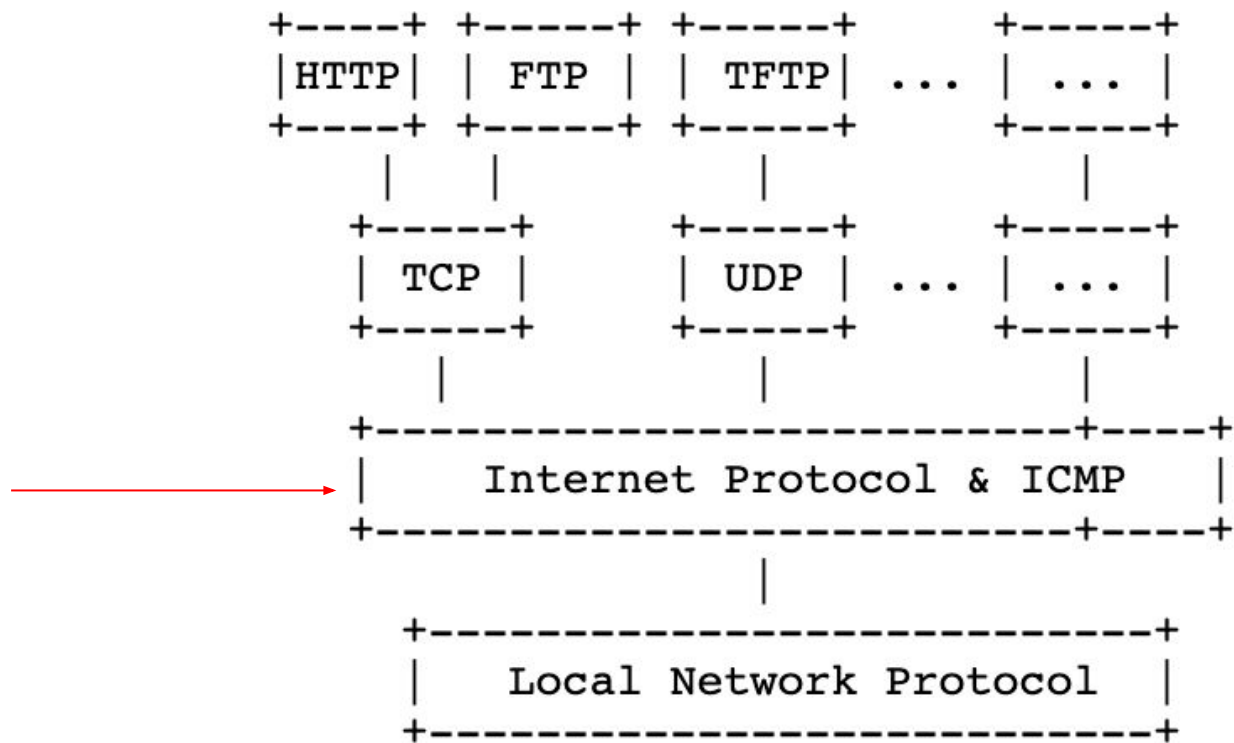
Про IP нам достаточно знать две вещи:

- IP обеспечивает адресацию (умеет* передавать данные по тому адресу, который вы укажете).
- IP обеспечивает пересылку данных, но не гарантирует доставку.

Под термином удалённый подразумевается, что он «удалён» от нас физически (т.е. находится где-то в другом месте).

Примечание*: то, что умеет, не значит, что передаст, поскольку удалённый хост может быть недоступен.

IP





TCP

RFC: <https://tools.ietf.org/html/rfc793>.

The TCP is intended to provide a **reliable process-to-process communication service** in a multinetwork environment.

Т.е. TCP обеспечивает надёжную* передачу данных от одного процесса к другому (даже при условии что процессы работают на разных хостах).

Примечание*: надёжную не значит, что ваши данные гарантированно будут доставлены.



HOST & PORT

Поскольку процессов на нашем хосте может быть много, то в рамках TCP определено ещё понятие порта — это просто целое число, которое операционная система выделяет процессу для сетевого взаимодействия.

Благодаря связке из IP-адреса и номера порта два процесса на разных хостах могут «найти друг друга».

КЛИЕНТ-СЕРВЕРНАЯ МОДЕЛЬ

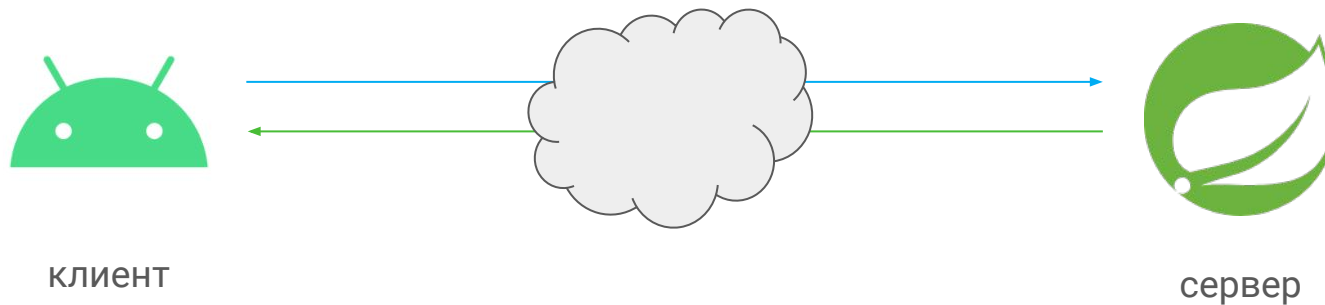
Если мы посмотрим на нашу схему, то увидим, что компоненты в рамках неё не равнозначны:

- Android приложение может быть в любой момент закрыто, отключено от сети и т.д.;
- сервер же должен работать «непрерывно», в противном случае пользователи очень расстроятся;
- взаимодействие инициируется клиентом (он посылает запросы на сервер в ответ на действия пользователя), сам сервер не может открыть приложение на вашем устройстве*.

Примечание*: именно для того, чтобы сервер мог по своей инициативе присылать данные устройству и были придуманы Push'и.

КЛИЕНТ-СЕРВЕРНАЯ МОДЕЛЬ

Такую модель взаимодействия называют клиент-сервер:



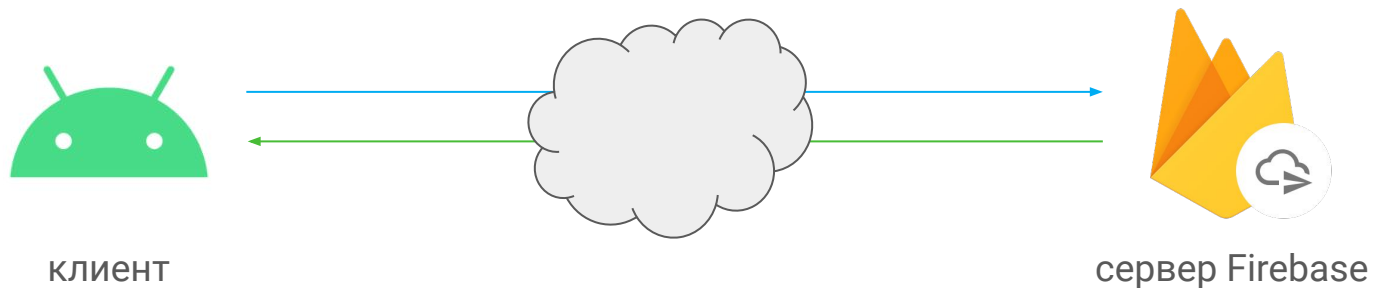
Задача клиента: подготовить и отправить запрос, получить ответ и обработать его.

Задача сервера: принять и обработать запрос, подготовить и отправить ответ.

Первым в этой модели всегда начинает клиент.

PUSH'и

На самом деле, с пушами тоже работает клиент-серверная модель. Только клиентом тут является сам Android, который периодически посылает запросы на сервер Firebase, опрашивая последний о наличии уведомлений:





DNS

Клиент должен знать, куда отправлять запрос (а именно IP и порт). Но зачастую используется не IP, а человекопонятный адрес, например, `netology.ru` вместо `104.22.49.171`.

За это отвечает специальный сервис, который называется DNS (Domain Name System), который и позволяет переводить человекопонятные доменные имена в IP-адреса (он есть и в традиционных ОС, и в Android).

СОБИРАЕМ ВСЁ ВМЕСТЕ

Итак, что мы получили:

1. У нас есть клиенты и сервера.
2. Сервер должен быть запущен на определённом хосте на фиксированном порту, чтобы клиент мог к нему обратиться.
3. Сервер должен непрерывно ожидать запросов от клиентов.

Давайте попробуем запустить свой небольшой сервер*.

Примечание*: мы будем предоставлять вам готовые сервера для ваших ДЗ.

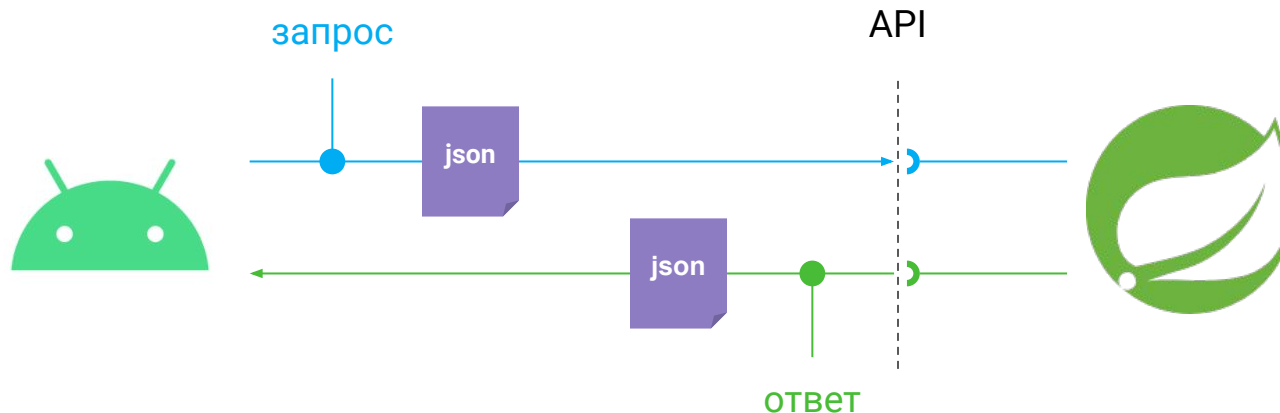


ФОРМЫ ИНТЕГРАЦИИ

HTTP

Доминирующей формой интеграции является HTTP, а если быть точнее, два варианта:

1. REST.
2. JSON over HTTP.



Несмотря на это, существуют и более продвинутые варианты вроде GraphQL, gRPC, WebSockets, и даже написание собственных протоколов поверх TCP.

HTTP

- 1.1: <https://tools.ietf.org/html/rfc2616>.
- 2.0: <https://tools.ietf.org/html/rfc7540>.

HTTP (HyperText Transfer Protocol) — протокол для создания hypermedia систем. Под hypermedia понимается как гипертекст (HTML-страницы со ссылками), так и различные мультимедиа форматы.

Этим его применение не ограничивается, и по факту, он используется в качестве удобного «транспорта» для данных уровня приложения.

HTTP

На сегодняшний день есть две версии протокола:

- 1.1 — текстовая (значит запросы и ответы можно **интерпретировать как строки** в человеко-понятном формате).
- 2 — бинарная (значит запросы и ответы нельзя интерпретировать как строки).

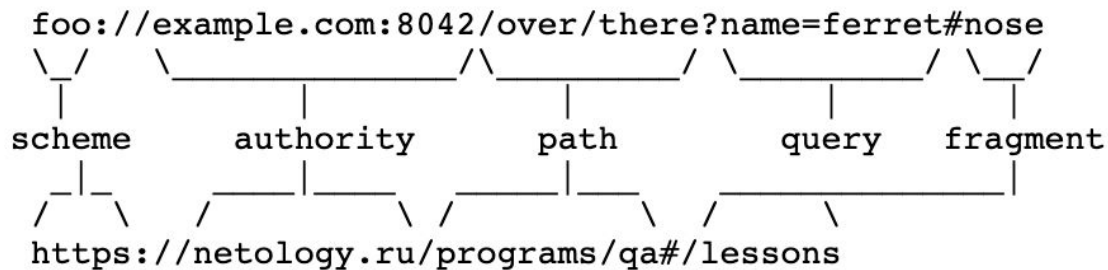
Рассмотрим для примера версию 1.1.

URI

В рамках HTTP запросы делаются на определённый адрес, который представляет из себя URI:

URI = scheme ":" hier-part ["?" query] ["#" fragment]

Иерархическая часть состоит из authority и path.



Authority может включать в себя логин и пароль пользователя, хост (или домен) и порт. Логин и пароль указываются достаточно редко. Порт также не указывается, если используется [общепринятый](#) (80 для http, 443 для https).

HTTP

В рамках протокола определяется понятие «сообщения» (message) как единицы передачи информации.

Сообщение может быть либо запросом, либо ответом:

4.1 Message Types

HTTP messages consist of requests from client to server and responses from server to client.

HTTP-message = Request | Response ; HTTP/1.1 messages

5 Request

A request message from a client to a server includes, within the first line of that message, the method to be applied to the resource, the identifier of the resource, and the protocol version in use.

```
Request      = Request-Line           ; Section 5.1
               *(( general-header     ; Section 4.5
                 | request-header      ; Section 5.3
                 | entity-header ) CRLF) ; Section 7.1
               CRLF
               [ message-body ]       ; Section 4.3
```

6 Response

After receiving and interpreting a request message, a server responds with an HTTP response message.

```
Response     = Status-Line           ; Section 6.1
               *(( general-header     ; Section 4.5
                 | response-header     ; Section 6.2
                 | entity-header ) CRLF) ; Section 7.1
               CRLF
               [ message-body ]       ; Section 7.2
```

СООБЩЕНИЕ

Сообщение содержит:

1. Строку запроса (или строку ответа).
2. Заголовки (служебные данные).
3. Тело (не все запросы).

И в любых частях сообщения можно передавать данные.

HTTP

В рамках HTTP также определяются:

1. Методы (GET, POST, PUT, DELETE) определяют логическое назначение действия + ограничения на запрос (например, у GET тело запроса должно быть пустое).
2. Статус-коды ответов:
 - 1xx — информационные
 - 2xx — успешно
 - 3xx — перенаправление
 - 4xx — ошибки клиента
 - 5xx — ошибки сервера

▼ General

Request URL: https://netology.ru/

Request Method: GET

Status Code:  200

Remote Address: 104.22.48.171:443

1

2

МЕТОДЫ

- GET — получение ресурса (не имеет тела).
- POST — создание/обновление ресурса.
- PUT — обновление ресурса.
- PATCH — обновление ресурса (чаще всего — частичное).
- DELETE — удаление ресурса.

Как вы видите, правила не строгие, поэтому интерпретируются достаточно вольно и смысловая нагрузка сильно зависит от разработчиков backend'a.

REST

Сам термин REST (Representational State Transfer) был введен Roy Fielding в [своей диссертации](#) в 2000 году.

В современной интерпретации это выглядит так:

- HTTP как транспорт для передачи сообщений;
- статус коды как сигнал успешности выполнения запросов;
- HTTP-методы в качестве «осмысленных» операций;
- группировка данных в наборы ресурсов с доступным перечнем операций для них;
- stateless — клиент должен передавать все необходимые данные для выполнения запроса;
- json как формат передачи для большинства запросов (исключение — бинарные данные, например, картинки).



REST

Мы говорим в «современной», потому что изначальный смысл термина (описанный в оригинальной работе) «потерялся», и под REST сейчас понимают именно те пункты, которые мы описали.

GITHUB API

Пример Github:

List organization repositories ⓘ

```
GET /orgs/:org/repos
```

Response

```
Status: 200 OK
Link: <https://api.github.com/resource?page=2>; rel="next",
      <https://api.github.com/resource?page=5>; rel="last"
```

```
[
  {
    "id": 1296269,
    "node_id": "MDEwOlJlcG9zaXRvcnkxMjk2MjY5",
    "name": "Hello-World",
    "full_name": "octocat/Hello-World",
    ...
  }
]
```

Create an organization repository ⓘ

POST /orgs/:org/repos

Example

```
{
  "name": "Hello-World",
  "description": "This is your first repository",
  "homepage": "https://github.com",
  "private": false,
  "has_issues": true,
  "has_projects": true,
  "has_wiki": true
}
```

Response

Status: 201 Created 
Location: <https://api.github.com/repos/octocat/Hello-World>

```
{
  "id": 1296269,
  "node_id": "MDEwOlJlcG9zaXRvcnkxMjk2MjY5",
  "name": "Hello-World",
  "full_name": "octocat/Hello-World",
  ...
}
```


ОБЩАЯ КОНЦЕПЦИЯ

Общая концепция в идеальном случае выглядит так:

- **GET** /resources — список ресурсов типа resource (в GitHub — orgs).
- **GET** /resources/:id — детали по конкретному ресурсу.
- **POST** /resources — создание нового ресурса.
- **PATCH/PUT** /resources/:id — обновление ресурса.
- **DELETE** /resources — удаление всех ресурсов (реализуется редко).
- **DELETE** /resources/:id — удаление конкретного ресурса.

Где :id это placeholder: заменяемая на конкретное значение часть пути (необязательно число), например: /resources/netology. Часто её (эту часть) называют path parameter (или path param).

ОБЩАЯ КОНЦЕПЦИЯ

Тогда:

- **GET** /resources/:id/subresources (в GitHub — /orgs/:org/repos).
- **GET** /resources/:id/subresources/:sid.
- **POST** /resources/:id/subresources.
- **PATCH/PUT** /resources/:id/subresources/:sid.
- **DELETE** /resources/:id/subresources/:sid.

ПРОБЛЕМЫ

И это только двухуровневая организация. А теперь пойдём дальше: в каждом репо есть issue, коммиты, релизы и прочее.

А в issue есть ответы (URL'ы становятся уже гигантскими):

[/orgs/:orgId/repos/:repoId/issues/:issueId/comments/:commentId.](#)

Кроме того, непонятно, какой URL должен быть на то, чтобы получить топ репозиторийев (самые популярные репозитории)?

Поэтому такая «чистая» схема встречается разве что в книжках.

GITHUB API

Get a repository ⓘ

```
GET /repos/:owner/:repo ← he /orgs/:org/repos/:repo
```

Response

The `parent` and `source` objects are present when the repository is a fork.

`parent` is the repository this repository was forked from, `source` is the ultimate source for the network.

Status: 200 OK



```
{
  "id": 1296269,
  "node_id": "MDEwOlJlcG9zaXRvcnkxMjk2MjY5",
  "name": "Hello-World",
  "full_name": "octocat/Hello-World",
  ...
}
```

JSON OVER HTTP

Поэтому некоторые разработчики предпочитают реализовывать API, похожее на вызов методов:

Wall

Методы для работы с записями на стене.

<code>closeComments</code>	 Выключает комментирование записи
<code>createComment</code>	 Добавляет комментарий к записи на стене.
<code>delete</code>	Удаляет запись со стены.
<code>deleteComment</code>	Удаляет комментарий к записи на стене.
<code>edit</code>	Редактирует запись на стене.
<code>editComment</code>	Редактирует комментарий на стене.
<code>get</code>	Возвращает список записей со стены пользователя или сообщества.
<code>getById</code>	Возвращает список записей со стен пользователей или сообществ по их идентификаторам.



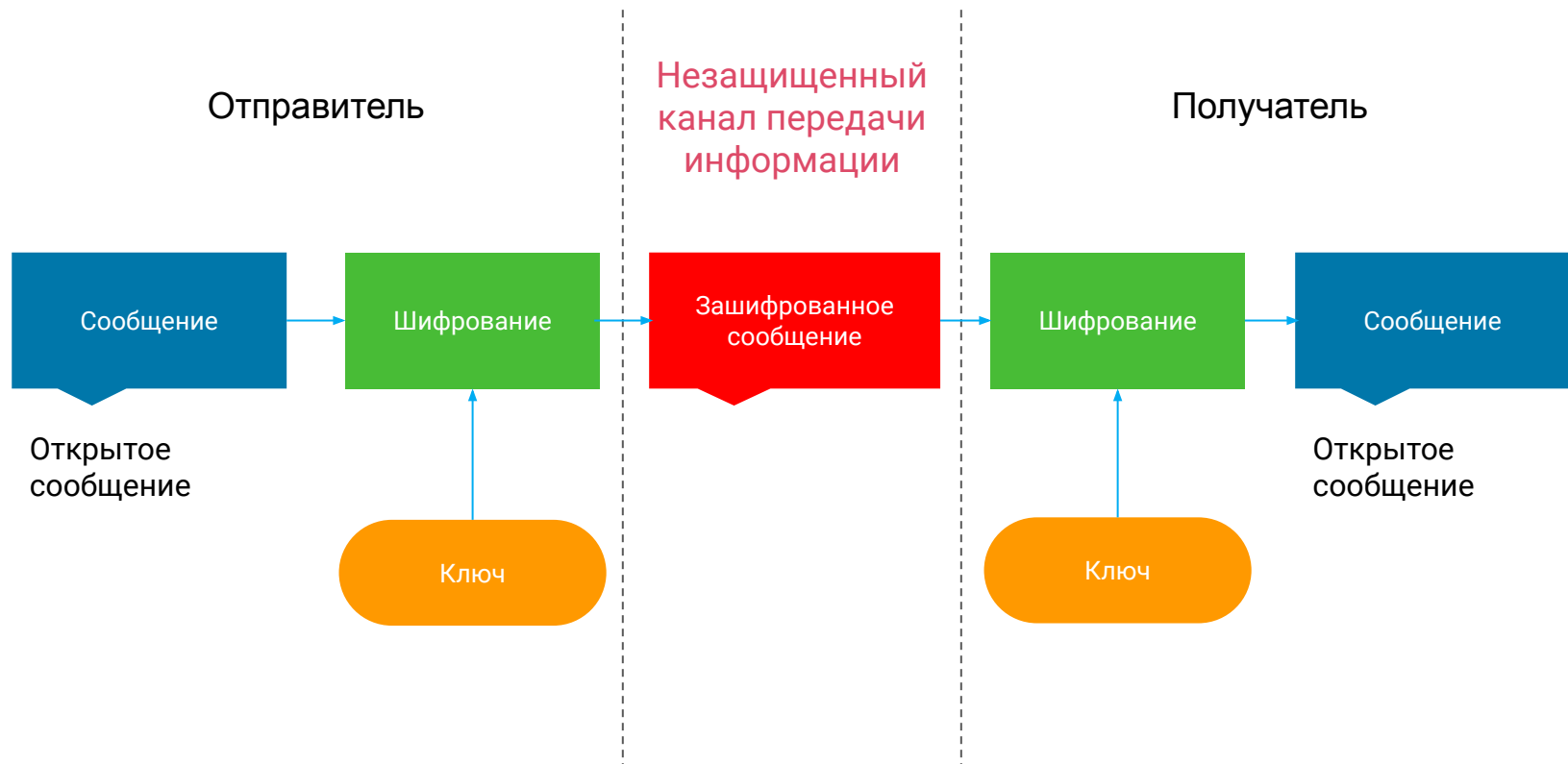
HTTPS

HTTPS

Данные, передаваемые по HTTP, передаются в открытом виде — это значит, что любое промежуточное устройство в виде «открытого текста» (пусть даже там бинарные данные) видит всё, что передаётся.

Это не очень хорошо, особенно при передаче чувствительных данных: личных данных, платёжных данных, персональных фото и видео.

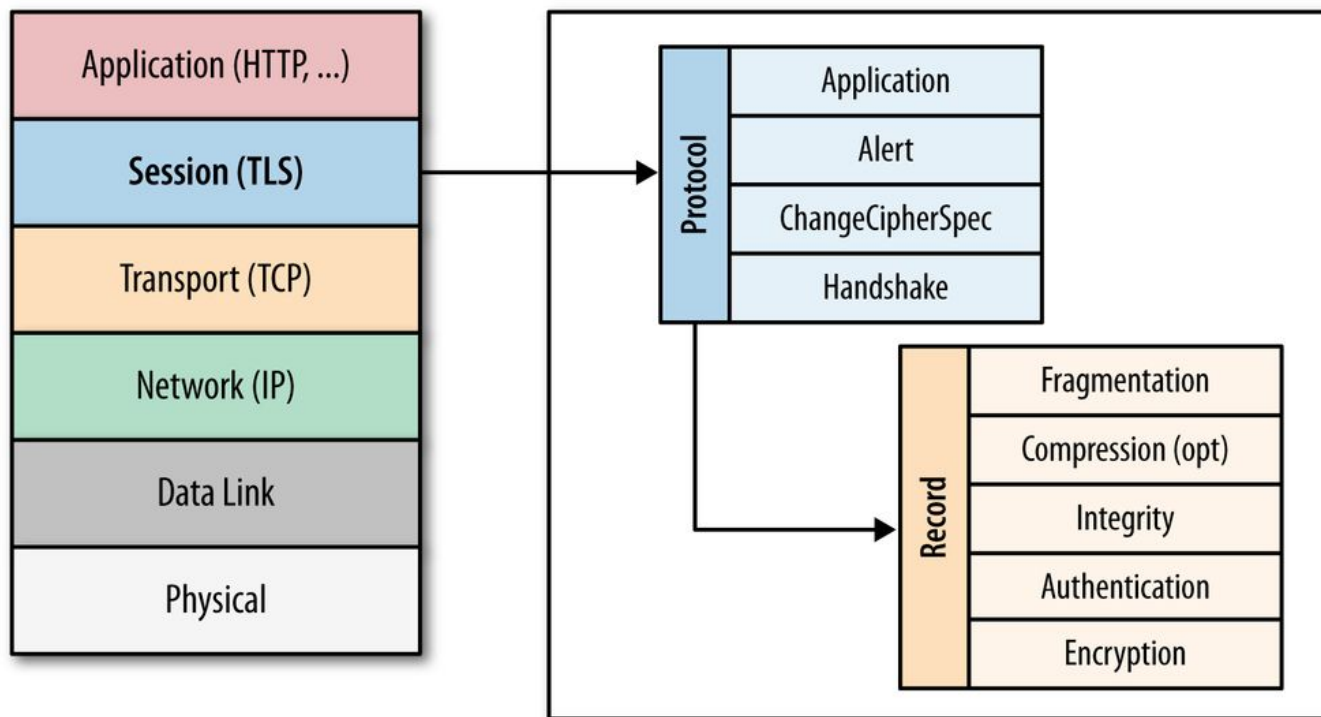
ШИФРОВАНИЕ



Ключ — это набор байт, который хранится в секрете, с помощью которого выполняется криптографическое преобразование.

TLS

TLS — специальный протокол, которым можно обеспечить шифрование (в том числе HTTP-трафика).



Изображение из книги High Performance Browser Networking by Ilya Grigorik



TLS

TLS (Transport Layer Security) — протокол, предоставляющий конфиденциальность и целостность данных при коммуникациях между двумя приложениями.

В рамках TLS две взаимодействующие стороны генерируют ключ, который будут использовать для шифрования, и шифруют все данные с помощью него.

HTTPS (безопасная версия HTTP) использует TLS.

TLS

Ключевая идея: существуют специальные центры (доверенные), которые занимаются тем, что выпускают сертификаты и подтверждают их подлинность.

Владельцы сайтов устанавливают сертификаты на своих веб-сервисах, что позволяет:

1. Удостоверять подлинность веб-сайта (можно проверить сертификат у соответствующего центра).
2. Шифровать трафик между клиентом и веб-сервисом.



TLS

Мы специально на первых лекциях будем эмулировать ситуацию, при которой у нас нет сертификата (посмотрим, как настраивать Android для этих сценариев).

А затем будем использовать самоподписанный, сгенерированный нами сертификат.

Это типичные «истории» на этапе разработки, поэтому вам нужно уметь с ними работать.



BACKEND

СЕРВЕР NMEDIA

Сервер представлен в виде обычного Gradle-проекта, написанного на фреймворке Spring.

Он поддерживает ключевые необходимые нам операции:

Получение списка сообщений

► **GET** <http://localhost:9999/api/posts>

Создание нового сообщения (id = 0)

► **POST** <http://localhost:9999/api/posts>
Content-Type: application/json

```
{
  "id": 0,
  "author": "Netology",
  "content": "First Post",
  "published": 0,
  "likedByMe": false,
  "likes": 0
}
```

СЕРВЕР NMEDIA

Получение сообщения по id

- ▶ GET <http://localhost:9999/api/posts/1>

Обновление сообщения (id != 0)

- ▶ POST <http://localhost:9999/api/posts>
Content-Type: application/json

```
{  
  "id": 1,  
  "author": "Netology",  
  "content": "First Post (UPDATED)",  
  "published": 0,  
  "likedByMe": false,  
  "likes": 0  
}
```

Удаление сообщения по id

- ▶ DELETE <http://localhost:9999/api/posts/1>

Конечно же, для конкретных ДЗ API будет расширяться, например, появятся лайки и т. д.



СЕРВЕР NMEDIA

Сейчас мобильные разработчики часто пишут «заглушки» либо демо-сервера для своих приложений (т.к. production сервера не всегда готовы к моменту разработки).

Поскольку мы предоставляем вам уже готовые сервера, в наш курс не входит рассмотрение вопросов разработки серверной части.



POSTMAN

Вы можете использовать [Postman](#) или любой подобный инструмент для отправки запросов.

Демонстрация: запуск сервера (запускать можно через IDEA либо через `./gradlew bootRun` Linux/Mac `.\gradlew bootRun` Windows в командной строке) и отправка запросов через Postman.

Рекомендация: запросы Postman удобно сохранять в коллекцию.



ИТОГИ



ИТОГИ

Сегодня мы посмотрели теоретические вопросы организации API, которые нам предстоит использовать.