1. **Что такое интернет?**

Интернет: Всемирная компьютерная сеть, построенная на основе стека протоколов TCP/IP. Количество пользователей: около 3 млрд.

Источник: Смелов, ПИ лекция 1

1. **4 составляющие интернета**

Это протоколы тсп/айпи, службы, организациии, документация.

1. **Организации(названий хватит)**

**ISOC**: Internet Society – международная организация (офисы США, Швейцария), занимающаяся развитием сети Internet. Состав: 20 тыс. индивидуальных членов, более 100 организаций в 180 странах. ISOC владелец RFC-стандартов. ISOC обеспечивает правовую поддержку и финансирует все другие организации, связанные с деятельностью Internet (IETF, IAB,…).

**IETF**: Internet Engineering Task Force - рабочая группа проектирования Internet. Публикует RFC (Request for Comments – заявка на отзывы = тема для обсуждения). Задачи IETF описаны в RFC 4677.

**IAB**: Internet Architecture Board - совет по архитектуре Internet, одна из комиссий IETF, имеет консультативный статус при ISOC.

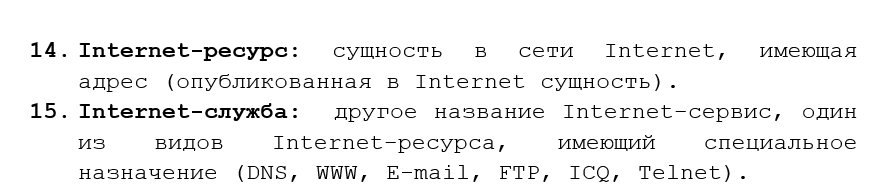
**ICANN**: Internet Corporation for Assigned Names and Numbers – корпорация по управлению доменными именами и IP-адресами.

**IANA**: Internet Assigned Numbers Authority – Администрация адресного пространства Internet. Под контролем ICANN. Кроме того регистрирует типы данных MIME.

**W3C**:World Wide Web Consortium – организация разрабатывающая и внедряющая web-стандарты (HTTP, HTML, URI/URL, CSS, DOM, XML, PNG, SVG,…). Возглавляет Тимоти Джон Бернерс-Ли.

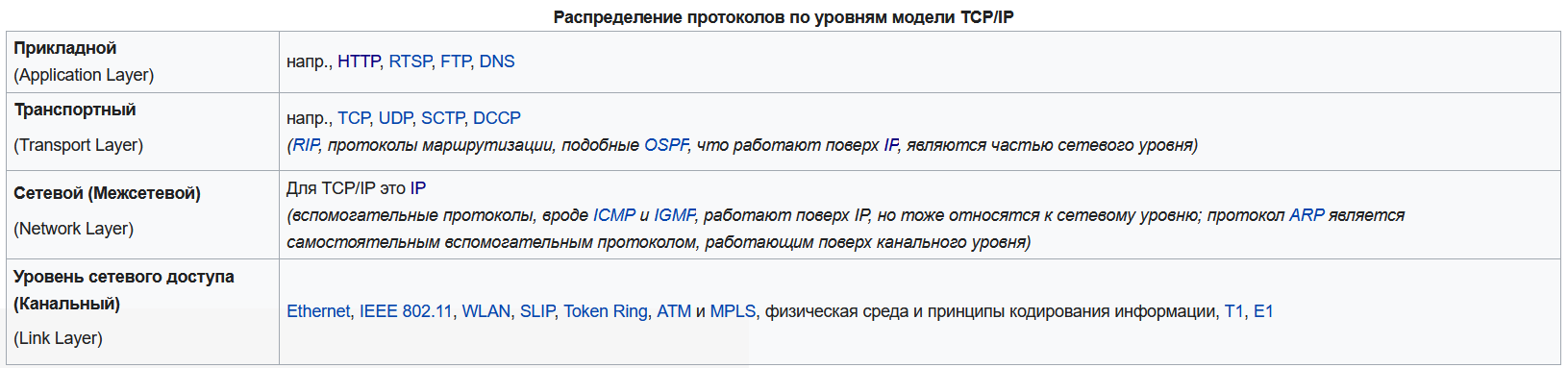
Источник: Смелов, ПИ лекция 1

1. **Примеры служб и что это такое**



Службы (сервисы) Интернет: DNS, E-mail(STMP,POP3, IMAP), IRC(обмен сообщений в реальном времени), FTP, Telnet (управление удаленным компьютером в терминальном режиме), WWW.

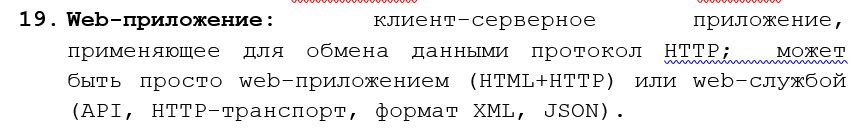
1. **На какому ровне TCP/IP понятие IP?**

****

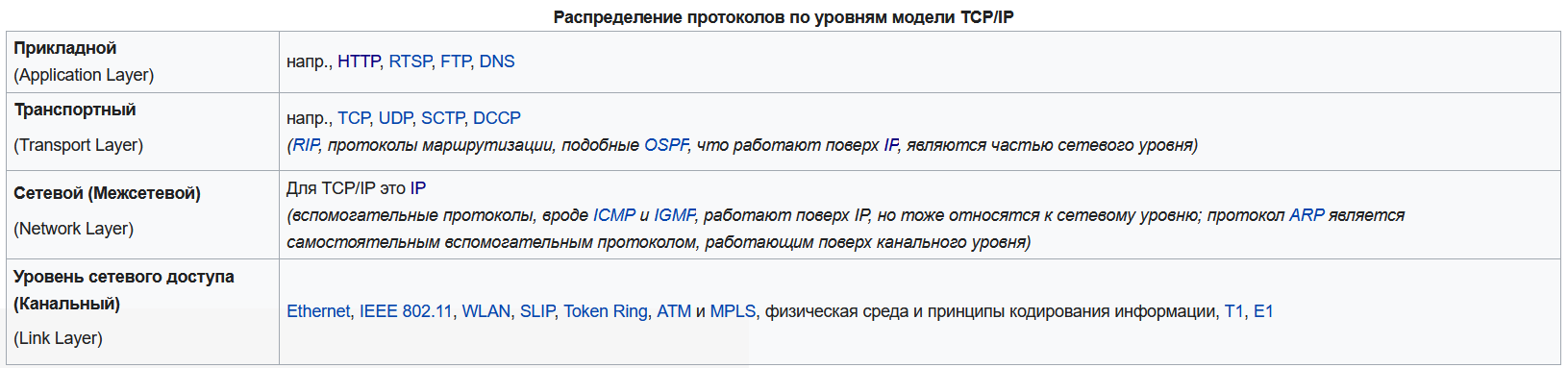
Источник: Википедия

1. **Что такое HTTP?**

Сетевой протокол для передачи данных

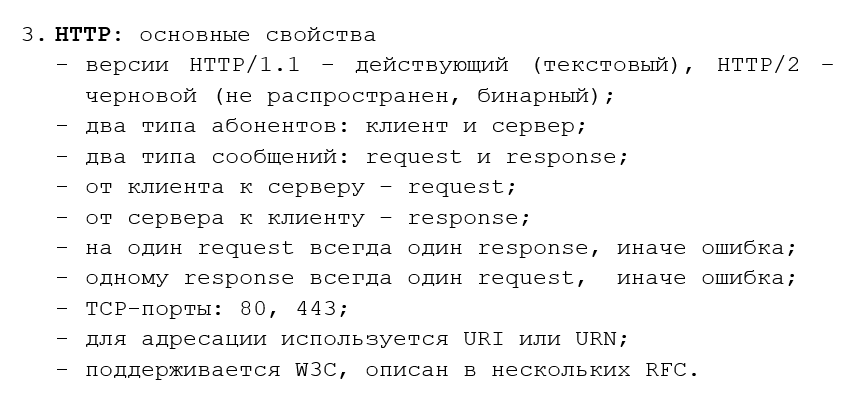


1. **На каком уровне TCP/IP понятие HTTP?**

****

Источник: Википедия

1. **Свойства HTTP**

****

Источник: Смелов, ПИ лекция 2

1. **Порты 80 и 443**

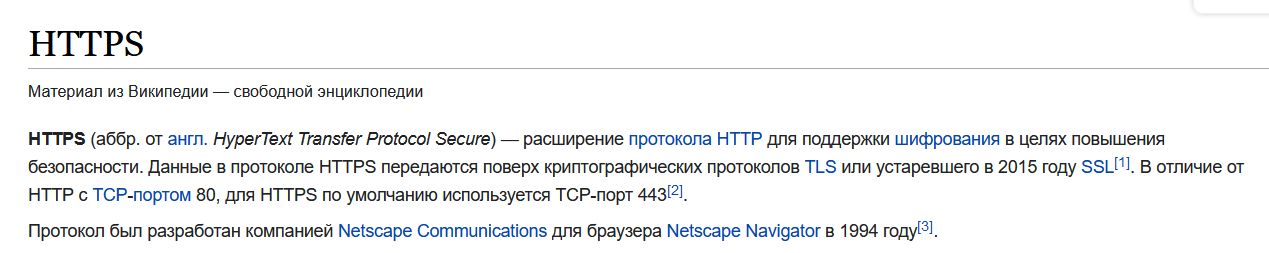
80 – для HTTP

443 – для HTTPS

1. **HTTPS(отличие от HTTP)**



Источник: файл Смелова HTTP\_authen(там опечатка, TLS)



Источник: Википедия

1. HTTPS использует сертификаты TSL/SSL для шифрования.
2. Изза этого хуже производитеность
3. Но лучше доверие и ранжировка у таких сайтов
4. **Документация(RFC)**

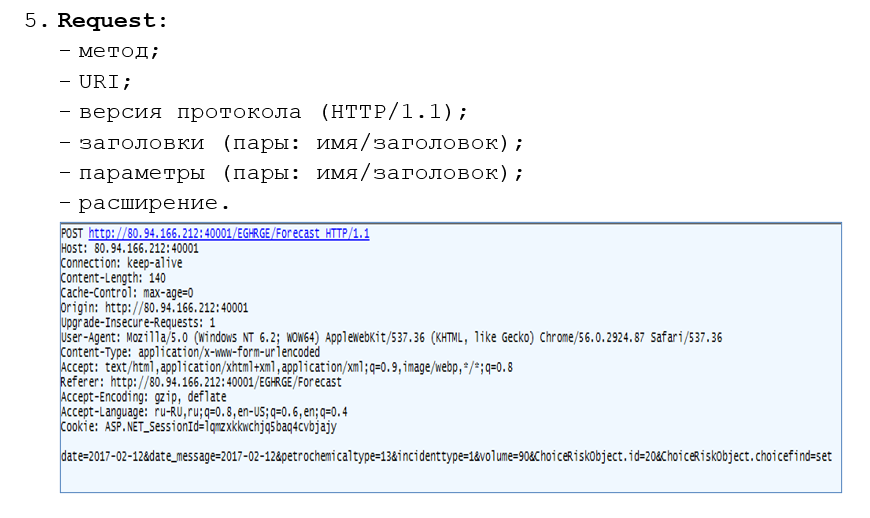
RFC – серия пронумерованных документов, содержащих технические спецификации и стандарты.

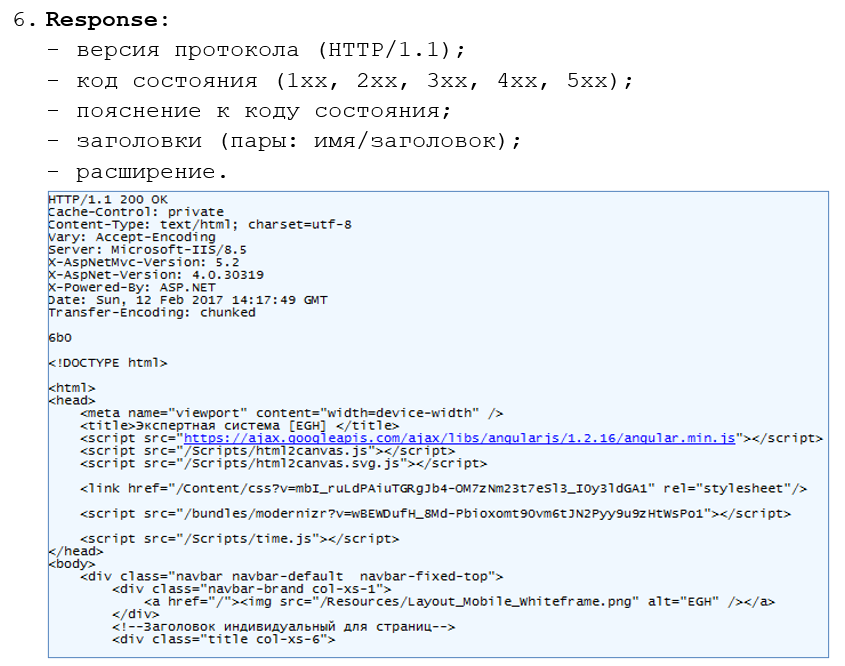
1. **Отличие версий HTTP(1.1 и 2.0)**

HTTP 1.1 – использует текст(текстовый), полудуплексный

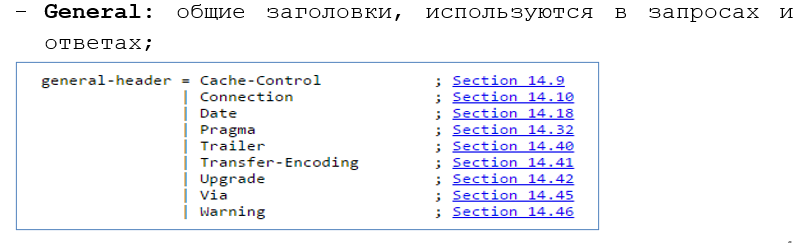
HTTP 2.0 – бинарный, дуплексный

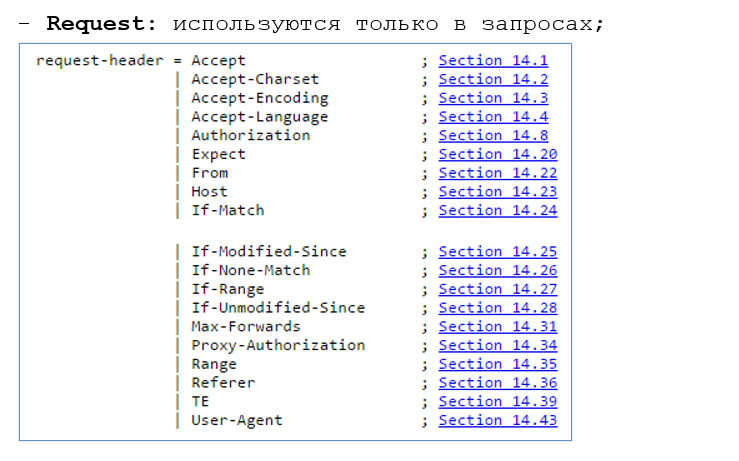
1. Мультипликирование (несколько запросов обрабатываются одновременно, не блокируя остальные)
2. Сжатие заголовков через HPACK (меньше места, быстрее передача)
3. Бинарный фомат -> быстрее передача
4. **Структура HTTP запросов и ответов**

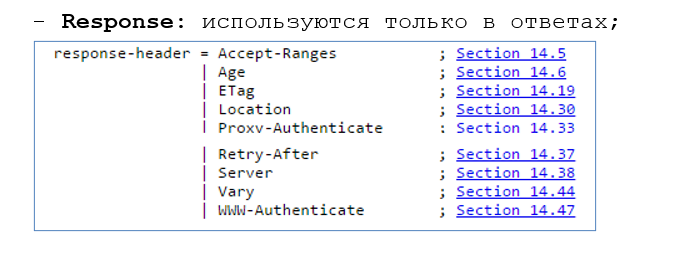
****

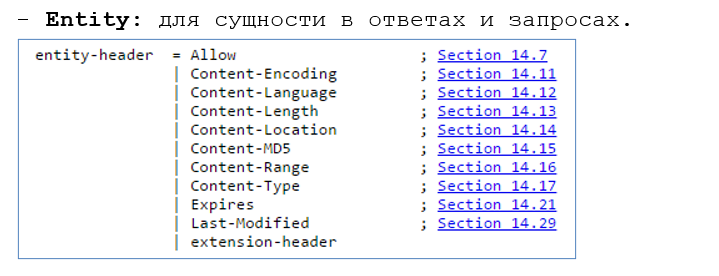
****

1. **Виды заголовков HTTP(могут потребовать по 1 примеру каждого)**

****

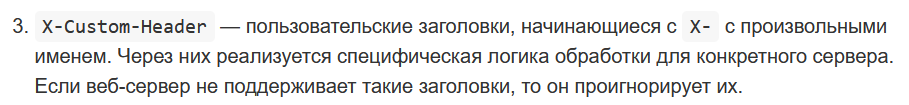
****

****

****

Источник: Смелов, ПИ лекция 2

Хоть в лекции и написано про 4 типа, но Смелов считает что их 5. Пятым является пользовательский тип

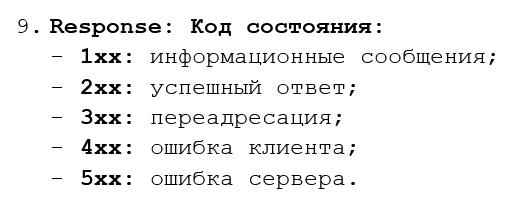


А ещё где-то в этих скриншотах в его лекции ловушка – приведя в пример какой-нибудь заголовок Смелов может сказать, что он на самом деле другого типа и не засчитать ответ – нужно будет приводить ещё. Не знаю сколько таких, может и один, у меня записан только Host.

Источник: слова Смелова,

Хабр https://habr.com/ru/companies/avito/articles/710660/

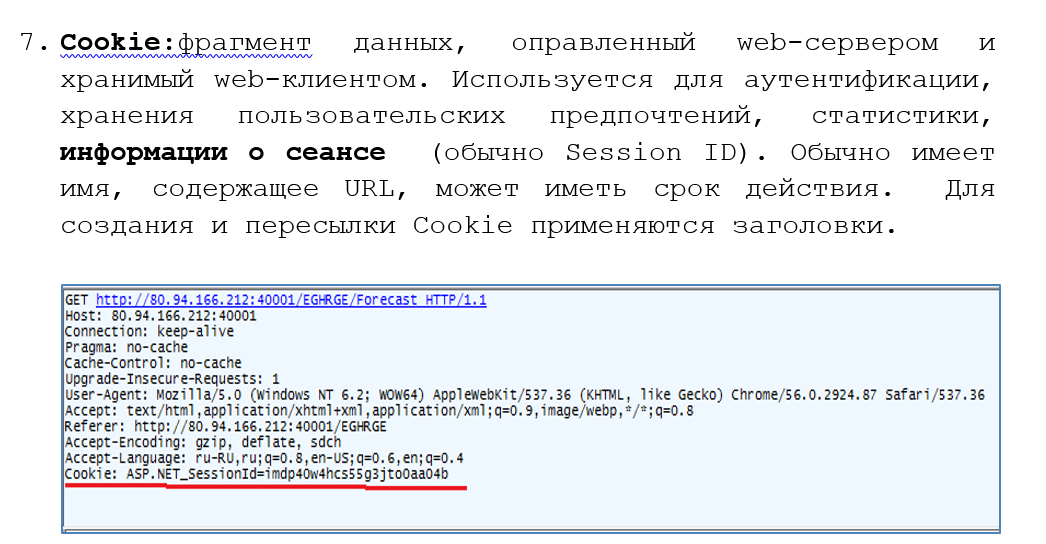
1. **Серии статусов HTTP-ответов(могут потребовать примеры)**

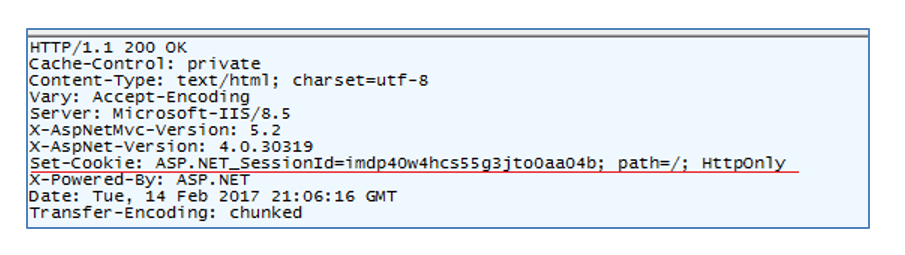




Источник: Смелов, ПИ лекция 2

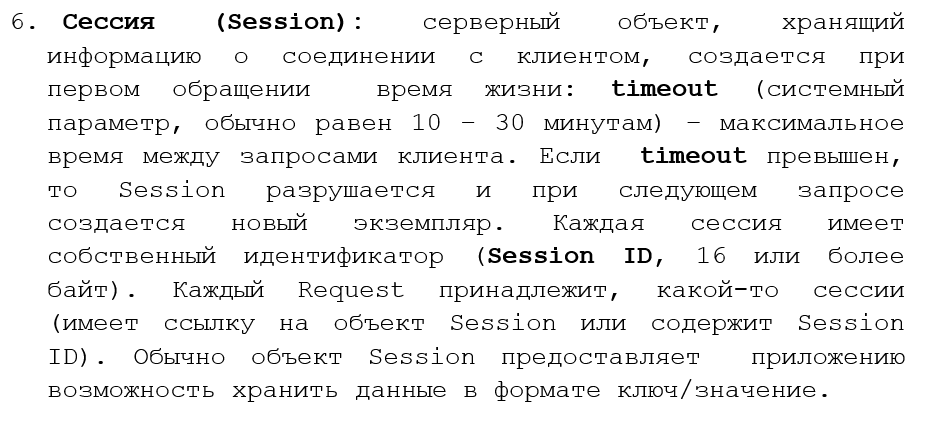
1. **Куки**





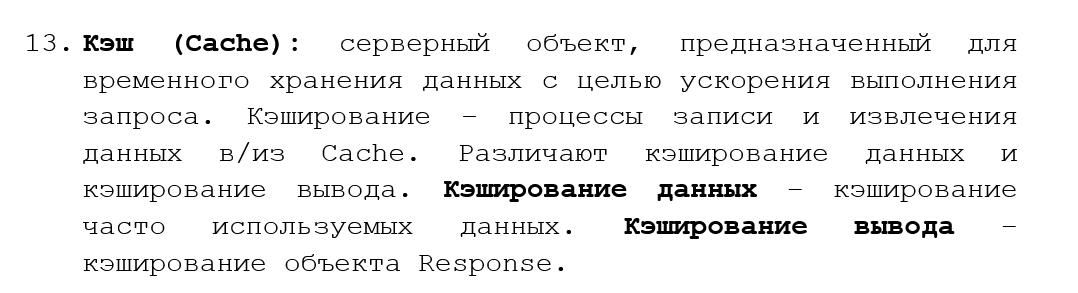
Источник: Смелов, ПИ лекция 3

1. **Сессия**



Источник: Смелов ПИ лекция 3

1. **Кэширование**



Источник: Смелов, ПИ лекция 3

1. **Дуплекс, полудуплекс, симплекс, подписка**

Что такое симплекс?

Симплекс означает, что при передаче данных поддерживается только-односторонняя передача. В практических приложениях есть принтеры, радиостанции, мониторы и т. д. Только принимать сигналы или команды, не посылать сигналы.

Что такое полудуплекс?

Полу-дуплекс означает, что передача данных поддерживает двунаправленную передачу, но не может выполнять двунаправленную передачу одновременно. В то же время один конец может только отправлять или получать.

Что такое дуплекс?

Дуплекс означает, что данные передаются в двух направлениях одновременно, что представляет собой комбинацию двух симплексных коммуникаций, требующих, чтобы передающее устройство и принимающее устройство имели независимые возможности приема и отправки одновременно.

Что такое подписка?

В модели "издатель-подписчик" данные передаются от одного источника (издателя) множеству получателей (подписчиков).

1. **Handshake(рукопожатие)**

### Этапы TLS-рукопожатия:

#### 1. **ClientHello** (Клиент отправляет запрос):

* Браузер отправляет на сервер сообщение **ClientHello**, которое содержит:
  + Версию поддерживаемого протокола TLS (например, TLS 1.3).
  + Список поддерживаемых шифров (например, AES, RSA).
  + Случайное число, сгенерированное клиентом (которое используется для создания ключей сессии).
  + Другие параметры, такие как поддерживаемые расширения (например, поддержка SNI).

#### Пример (содержимое ClientHello):

yaml

Копировать код

ClientHello:

Версия TLS: 1.3

Поддерживаемые шифры: AES, RSA

Случайное число клиента: a3f2b4c1d8e5...

Расширения: SNI = example.com

#### 2. **ServerHello** (Сервер отвечает):

* Сервер отправляет ответ **ServerHello**. В нем содержится:
  + Версия TLS, которая будет использоваться (например, TLS 1.3, согласованная сторонами).
  + Выбранный шифр для связи (например, AES-256).
  + Случайное число сервера.
  + Цифровой сертификат сервера, который подтверждает его подлинность (содержит публичный ключ сервера и информацию о домене).
  + (Опционально) запрос на сертификат клиента, если сервер требует аутентификацию клиента.

#### Пример (содержимое ServerHello):

yaml

Копировать код

ServerHello:

Версия TLS: 1.3

Выбранный шифр: AES-256

Случайное число сервера: f6d4c3a8b1e9...

Сертификат сервера: [сертификат SSL/TLS для example.com]

#### 3. **Проверка сертификата**:

* **Клиент** проверяет сертификат сервера:
  + Проверяет его подпись (например, подписан ли сертификат доверенным центром сертификации).
  + Убедится, что сертификат принадлежит домену example.com.
  + Проверяет срок действия сертификата (не просрочен ли он).

Если сертификат проходит проверку, клиент продолжает процесс рукопожатия.

#### 4. **Генерация Pre-Master Secret**:

* Клиент генерирует предварительный ключ сессии (**Pre-Master Secret**).
* Этот ключ клиент шифрует с помощью **публичного ключа сервера**, который он получил из сертификата сервера.
* Затем клиент отправляет этот зашифрованный ключ на сервер.

#### Пример (зашифрованный Pre-Master Secret):

java

Копировать код

Pre-Master Secret (зашифрованный): [ZGc9UDhsbkxoSVYr3E2+...]

#### 5. **Создание симметричных ключей**:

* Сервер расшифровывает Pre-Master Secret с помощью своего **приватного ключа**.
* Оба участника (клиент и сервер) теперь используют Pre-Master Secret и сгенерированные случайные числа (от клиента и сервера) для создания симметричных ключей шифрования.
* Эти ключи будут использоваться для шифрования и дешифрования всех последующих данных, передаваемых между клиентом и сервером.

#### 6. **Заключительные шаги (Finished)**:

* Клиент отправляет сообщение **Finished**, зашифрованное симметричным ключом, подтверждая, что он готов к началу защищенного обмена данными.
* Сервер отвечает своим сообщением **Finished**, также зашифрованным симметричным ключом.
* Теперь обе стороны готовы к безопасной передаче данных.

#### Пример (заключительный обмен):

arduino

Копировать код

Client Finished: [зашифрованное сообщение клиента]

Server Finished: [зашифрованное сообщение сервера]

#### 7. **Начало защищенного обмена данными**:

* Теперь все данные, передаваемые между клиентом и сервером, зашифрованы с использованием согласованных симметричных ключей.
* Браузер начинает безопасно отправлять HTTP-запросы (например, получение страницы example.com), а сервер отвечает зашифрованными данными.

### Итог:

* **ClientHello** — клиент предлагает параметры соединения.
* **ServerHello** — сервер выбирает параметры и отправляет сертификат.
* **Проверка сертификата** — клиент проверяет подлинность сервера.
* **Обмен ключами** — клиент и сервер создают симметричные ключи на основе Pre-Master Secret.
* **Заключение** — начинается защищенный обмен данными.

1. **Механизм обмена ключами**

Есть открытый ключ – он для шифрования

Есть закрытый – он для дешифрования

Версия для чайников(её пересказывать не нужно, только суть):

У нас есть два персонажа: **Боб** (сервер) и **Алиса** (браузер пользователя). Боб передает Алисе **навесной замок и удостоверение личности**. Открыть этот замок может только Боб, так как ключ есть только у него.

Замок = публичный (открытый) ключ.

Ключ от замка = приватный (закрытый) ключ.

Удостоверение личности = сертификат, подписанный центром сертификации

Получив замок и сертификат, Алиса сверяется с центром сертификации, убеждаясь, что это действительно Боб, надевает замок на ящик, в котором содержится некая секретная информация, и посылает обратно Бобу. Аналогично, браузер шифрует сообщение публичным ключом и отдает серверу.

Ящик с замком попадет к Бобу, тот откроет замок своим ключом и узнает секретную информацию. Так, сервер расшифровывает сообщение закрытым ключом, который есть только у него.

Если надо подробнее гуглите сами, нормальный ответ на 3 страницы выйдет.

1. **Long pooling(лонг пулинг)**

Лонг-пуллинг (Long Polling) — это техника общения между клиентом и сервером в веб-приложениях, которая используется для асинхронного получения данных от сервера. Лонг-пуллинг позволяет имитировать "пуш"-уведомления (мгновенную отправку данных от сервера к клиенту), даже если сервер не поддерживает постоянные соединения, как в WebSocket.

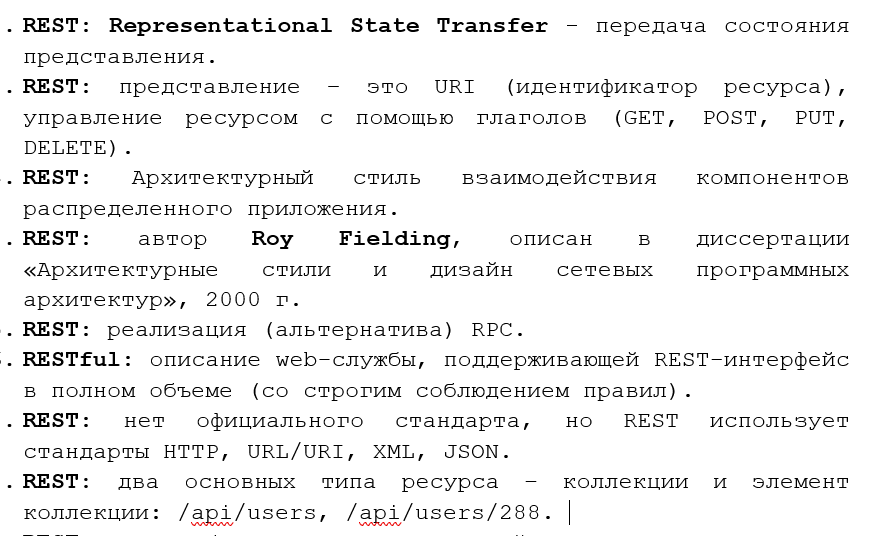
Как работает лонг-пуллинг:

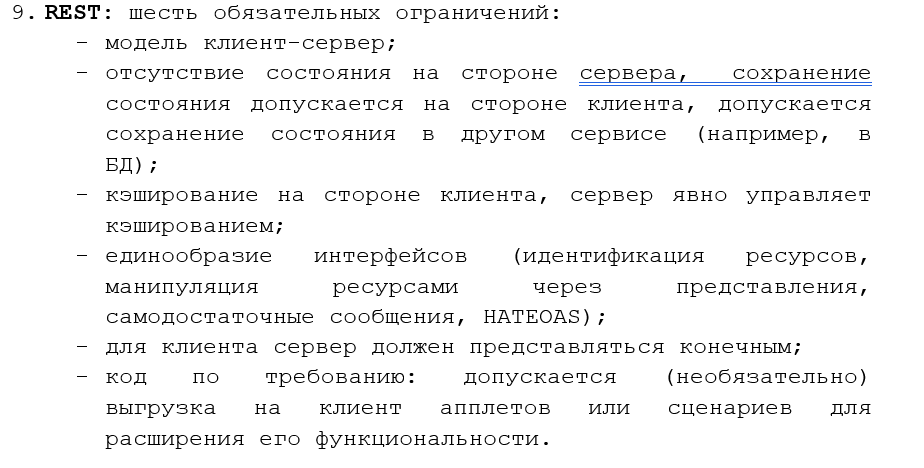
1. Запрос от клиента:
   * Клиент (например, браузер) отправляет HTTP-запрос на сервер.
2. Ожидание на сервере:
   * Сервер не отвечает немедленно на этот запрос. Он удерживает запрос в ожидании, пока не появятся новые данные, которые клиенту нужно получить.
3. Ответ сервера:
   * Когда новые данные становятся доступны, сервер отвечает на запрос клиента, отправляя эти данные.
4. Повторный запрос:
   * Как только клиент получает ответ, он немедленно отправляет новый запрос на сервер, чтобы продолжить ожидание новых данных.

Особенности лонг-пуллинга:

* В отличие от обычного HTTP-запроса, который сразу получает ответ от сервера, лонг-пуллинг подразумевает длительное ожидание ответа. Сервер "держит" соединение открытым до тех пор, пока не будут готовы данные.
* Если в течение определённого времени данных нет, сервер может отправить "пустой" ответ, и клиент снова отправит запрос.
* Это позволяет поддерживать иллюзию постоянного соединения между клиентом и сервером, даже если фактически каждый запрос и ответ — это отдельная HTTP-транзакция.

1. **REST и RESTfull**

****

****

Источник: Смелов, лекция ПWS 02XX(да, так и называется)

Многие штуки здесь стоит учить наизусть – Смелова может не волновать понимаете вы там что это или не понимаете, только пересказ его слов. Даже если он примет высока вероятность уйти не туда и получить порцию новых вопросов к которым вы не готовились.

Это только первая версия ответов на лабу, наверняка всплывет что-то ещё. Или где-то ответ неполный или неверный или новые вопросы, короче вы поняли.

Чат гпт про рест

**REST** и **RESTful** — это термины, связанные с архитектурным стилем проектирования веб-сервисов, но между ними есть небольшие различия. Давай разберем каждый из них по отдельности:

### 24) ****REST (Representational State Transfer)****:

REST — это **архитектурный стиль**, используемый для создания распределённых систем, особенно веб-сервисов. Он был предложен Роем Филдингом в его диссертации в 2000 году и основывается на наборе принципов и ограничений, которые позволяют создавать простые, масштабируемые и эффективные веб-API.

#### Основные принципы REST:

1. **Клиент-серверная архитектура**:
   * Клиент и сервер должны быть четко разделены. Клиент отправляет запросы, сервер отвечает. Это позволяет легко масштабировать систему и упрощает разработку и обслуживание.
2. **Безсостоя́тельность (stateless)**:
   * Каждый запрос от клиента к серверу должен быть независимым и содержать всю необходимую информацию для выполнения запроса. Сервер не хранит состояния клиента между запросами.
3. **Кэширование**:
   * Ресурсы на стороне клиента или прокси-сервера могут быть кэшируемыми, что снижает нагрузку на сервер и улучшает производительность.
4. **Единообразный интерфейс (uniform interface)**:
   * Все взаимодействие происходит через унифицированный набор операций, например, стандартные HTTP-методы (GET, POST, PUT, DELETE и т.д.). Ресурсы идентифицируются с помощью URI.
5. **Слои (layered system)**:
   * Архитектура должна поддерживать разделение на слои. Например, клиент может взаимодействовать с промежуточными серверами (прокси или шлюзы), не зная о существовании промежуточных слоев.
6. **Представление ресурсов**:
   * Ресурсы передаются в формате, отделённом от их внутренней реализации на сервере (например, в виде JSON, XML, HTML). Это позволяет гибко работать с данными.

### 2) ****RESTful (или RESTful API)****:

**RESTful** — это термин, который описывает веб-сервис или API, **следующий принципам REST**. То есть, если API или веб-сервис следует REST-ограничениям и принципам, он считается **RESTful**.

#### Основные характеристики RESTful API:

* **Использование HTTP-методов** для взаимодействия с ресурсами:
  + **GET** — для получения ресурса.
  + **POST** — для создания нового ресурса.
  + **PUT** — для обновления существующего ресурса.
  + **DELETE** — для удаления ресурса.
* **Идентификация ресурсов через URI**:
  + Например, если ресурс — это пользователь, его URI может быть /users/{id}. Запрос к /users/123 может вернуть данные о пользователе с идентификатором 123.
* **Использование форматов данных**:
  + Ресурсы передаются в формате, например, **JSON** или **XML**, который наиболее популярен для обмена данными между клиентом и сервером.
* **Статус-коды HTTP**:
  + RESTful API возвращает стандартные HTTP-статус-коды для обозначения результата операции (например, 200 OK, 404 Not Found, 201 Created и т.д.).
* **Безсостоя́тельность**:
  + Каждый запрос не зависит от предыдущих, и сервер не сохраняет состояние клиента между запросами.

### Разница между ****REST**** и ****RESTful****:

* **REST** — это **принципы** или **архитектурный стиль**, на котором строится API. Это не конкретная реализация, а набор правил и подходов, которым нужно следовать для создания распределённых систем.
* **RESTful** — это термин, используемый для описания **реализации** или API, которое **следует** этим принципам. Когда говорят о **RESTful API**, подразумевают, что API реализовано с соблюдением принципов REST.

### 25) ****Что означает, что HTTP является stateless (безсостоя́тельным)?****

HTTP — это **stateless протокол** (безсостоя́тельный), что означает, что каждый запрос от клиента к серверу обрабатывается **независимо** от предыдущих запросов. Сервер не сохраняет информацию о предыдущих взаимодействиях с клиентом.

Каждый HTTP-запрос передает всю необходимую информацию для выполнения операции, и сервер не "помнит" состояния клиента между запросами. Например, если ты отправляешь два запроса подряд на сервер, сервер не связывает их друг с другом — каждый запрос рассматривается как отдельная транзакция.

**Пример**: Если ты посещаешь интернет-магазин и добавляешь товар в корзину, сервер не "помнит", что ты уже добавил товар, если не используется механизм хранения состояния (например, с помощью cookies или сессий).

### 2) ****Чем stateless отличается от идемпотентности?****

**Stateless** и **идемпотентность** — это два разных свойства, хотя они могут применяться к одному и тому же протоколу (например, HTTP).

#### **Stateless (безсостоя́тельность)**:

* Как мы уже обсудили, это характеристика протокола, означающая, что сервер не сохраняет информацию о состоянии клиента между запросами. Каждый запрос независим от предыдущих.
* В **stateless** системе клиент каждый раз отправляет всю необходимую информацию в запросе, чтобы сервер мог его обработать, без зависимости от предыдущих действий.

#### **Идемпотентность**:

* **Идемпотентность** — это свойство определённых операций, которое означает, что **повторение одной и той же операции несколько раз приведёт к тому же результату**, что и выполнение её один раз.

**Пример идемпотентной операции**:

* HTTP-метод **GET** — идемпотентный, потому что если ты отправишь несколько одинаковых запросов на получение ресурса (например, страницу сайта), результат всегда будет одинаковым: сервер вернет содержимое этой страницы.
* HTTP-метод **PUT** — также идемпотентный, так как если ты отправляешь запрос на обновление ресурса с одинаковыми данными несколько раз, конечное состояние ресурса останется тем же, как если бы запрос был выполнен только один раз.

**Пример неидемпотентной операции**:

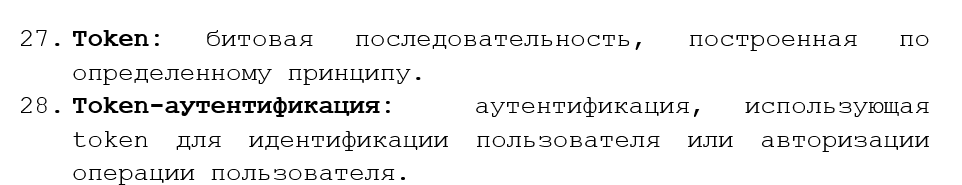
* HTTP-метод **POST** не является идемпотентным. Если ты отправишь несколько запросов POST на создание ресурса, это может привести к созданию нескольких одинаковых записей, потому что POST обычно используется для создания новых данных на сервере.

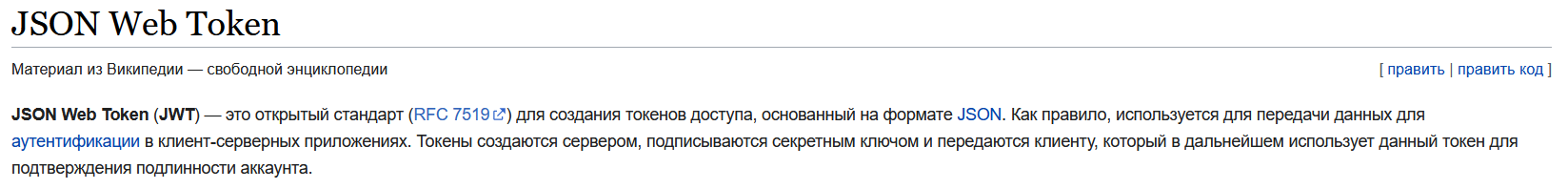
### Разница:

* **Stateless** описывает состояние системы (сервер не сохраняет информацию о прошлых запросах).
* **Идемпотентность** описывает поведение конкретной операции (результат одного и того же запроса, выполненного несколько раз, будет одинаковым).

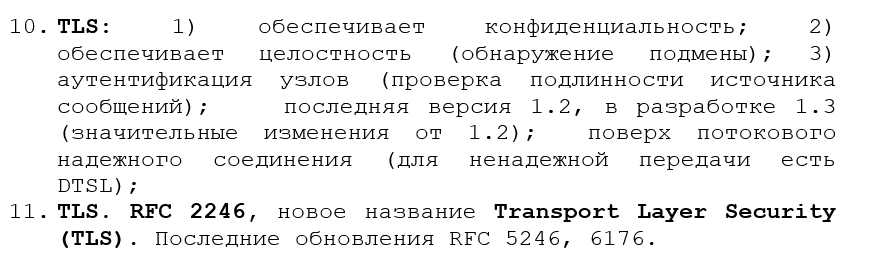
Идемпотентные операции могут существовать в **stateless** системах, но они могут также применяться и в системах с сохранением состояния.

Доп инфа



Что такое TLS



Как реализовать постояяное соединене?

