**Вопросы к экзамену «Технологии разработки web-приложений»**

1. **Сеть Интернет. Определение, основные понятия**.

**Интернет:** Всемирная компьютерная сеть, построенная на основе стека протоколов TCP/IP

**Интернет** – 4 компонента:

* Стек протоколов TCP-IP — это основа Интернета.
* Интернет службы (DNS, DHCP, SMTP, POP3 и т.д.).
* Документация в формате RFC и STD.
* Система организаций, которые поддерживают сеть

1. **Службы Интернет. Определение, основные понятия, примеры.**

**Internet-служба**: другое название Internet-сервис, один из видов Internet-ресурса, имеющий специальное назначение (DNS, WWW, E-mail, FTP, ICQ, Telnet)

**Internet-ресурс**: сущность в сети Internet, имеющая адрес (опубликованная в Internet сущность).

1. **Основные организации управления сетью Интернет.**

IETF (разрабатывает стандарты интернета в RFC)

ICANN (распространяет доменные имена)

IANA (регистрирует MIME)

ISOC (развитие, внедрение и распространение новых интернет-технологий)

 W3C (WWW Consortium, некоммерческая организация для согласования стандартов) — согласовывают всё, что касается веб-программирования (HTTP, CSS, SVG, URI/URL, XML, PNG, JPEG, …). Также они изготавливают стандарты и регистрируют их в IETF

(орг. стандартиз.) **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,…).

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

**RFC**:  жизненный цикл описан в RFC 2026, этапы:  Draft Internet, Proposed Standard,  Draft Standard, Internet Standard (RFC/STD), Historic (вышедшие из употребления). RFC может содержать не только стандарты, но и концепции (Experimental – результаты экспериментов, Informational – информационные, Best Current Practice – опыт применения). (орг. стандартиз.)

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

(орг. управл.)

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

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

**MIME**: **Multipurpose Internet Mail Extensions** - многоцелевые расширения Internet-почты. Используется и как стандарт кодирования Internet-сообщений. (**MIME**- формат данных которые можно пересылать).

(орг. стандартиз.) **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.

1. **Протокол HTTP. Основные свойства, версии и их особенности.**

**HTTP** — протокол прикладного уровня передачи данных, изначально — в виде гипертекстовых документов в формате HTML, в настоящее время используется для передачи произвольных данных.

**Основные свойства**

- версии HTTP/1.1 – действующий (текстовый), HTTP/2 – черновой (бинарный);

- два типа абонентов: клиент и сервер;

- два типа сообщений: request и response;

- на один request всегда один response, и на оборот иначе ошибка;

- TCP-порты: 80, 443 (HTTPS);

- для адресации используется URI или URN.

Версии HTTP:

* 1. 0.9 (methods: GET only, hypertext only, no HTTP headers (cannot transfer other content type files), no status/error codes, no URLs, no versioning)
  2. 1.0 (недействующий, нет постоянного соединения, methods: GET , HEAD , POST , имеет `Content-Type`)
  3. 1.1 (действующий, имеет постоянное соединение, has `Keep-alive`)
  4. 2.0 (бинарный, не распространен)
  5. 3.0 (не распространен)

**Протокол без сохранения состояния** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *Stateless protocol*) — это [протокол передачи данных](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%82%D0%BE%D0%BA%D0%BE%D0%BB_%D0%BF%D0%B5%D1%80%D0%B5%D0%B4%D0%B0%D1%87%D0%B8_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85), который относит каждый запрос к независимой транзакции, которая не связана с предыдущим запросом, то есть общение с сервером состоит из независимых пар [запрос-ответ](https://ru.wikipedia.org/w/index.php?title=%D0%97%D0%B0%D0%BF%D1%80%D0%BE%D1%81-%D0%BE%D1%82%D0%B2%D0%B5%D1%82&action=edit&redlink=1).

Протокол без сохранения состояния не нуждается в сохранении информации о сессии на сервере или статусе о каждом клиенте во время множественных запросов. В противовес этому, протокол, которому необходим учёт о внутреннем состоянии сервера, называется протоколом с сохранением [состояния](https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D1%81%D1%82%D0%BE%D1%8F%D0%BD%D0%B8%D0%B5_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *stateful*).

Примерами протоколов без сохранения состояния являются [Internet Protocol](https://ru.wikipedia.org/wiki/Internet_Protocol) (IP), и [Hypertext Transfer Protocol](https://ru.wikipedia.org/wiki/Hypertext_Transfer_Protocol) (HTTP).

Пример протокола без сохранения состояния — HTTP означает, что каждое сообщение запроса может быть принято в изоляции от других запросов.

1. **Понятия: URL/URI, MIME.**

* **URI: Uniform Resource Identifier** – унифицированный идентификатор ресурса (документ, изображение, файл, служба, электронная почта,…).
* **URL: Uniform Resource Location** - унифицированный локатор ресурса , содержащий местонахождение ресурса и способ обращения (протокол) к ресурса, описывает множество URI.
* **URN: Uniform Resource Name** - унифицированное имя ресурса – URI, имя ресурса, не содержащее месторасположение и способ доступа к ресурсу. В будущем URN должен заменить URL (для решения проблем с перемещением ресурсов в Internet).

**URI, URL, URN** –рекомендуется использовать термин URI

**Multipurpose Internet Mail Extensions** - многоцелевые расширения Internet-почты. Используется и как стандарт кодирования Internet-сообщений

1. **Структура HTTP-запроса.**

****

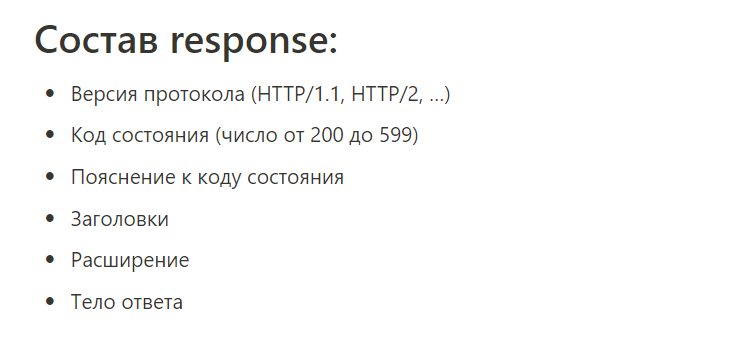
Параметры можно передавать:

- query string

- урлой

- в теле

1. **Структура HTTP-ответа.**



1. **HTTP-заголовки: типы, назначение, примеры.**

* **General**: общие заголовки, используются в запросах и ответах;



* **Request**: используются только в запросах;



* **Response:** используются только в ответах;



* **Entity**: для сущности в ответах и запросах.



1. **Протокол HTTPS: свойства, назначение, применение.**

**HTTPS** *— это расширение HTTP, обеспечивающее защиту передачи данных с использованием* ***TLS*** *(Transport Layer Security). Основные аспекты HTTPS включают:*

*- TLS*

*- Шифронаборы*

*- Сертификаты*

*- Процедура рукопожатия TLS*

**Сертификат** - это электронный документ, который разработан организацией международный институт телекоммуникаций и который гарантирует, что сервер безопасный

**Шифр набор** - это такая совокупность с номер, в котором описывается, как вырабатывается ключ, какое шифрование, какой метод хеширования, какой метод сжатия.

**Набор шифров** – это набор алгоритмов, создающих ключи для шифрования информации между браузером и сервером. Обычно набор шифров включает алгоритм обмена ключами, алгоритм проверки, алгоритм массового шифрования и алгоритм MAC. Несколько алгоритмов в протоколе TLS были модернизированы из SSL по соображениям безопасности.

1. **Протокол TLS: свойства, назначение, применение, процедура рукопожатия.**

**RFC 2246,** новое название **Transport Layer Security (TLS).** Последние обновления RFC 5246, 6176

**Свойства TLS**

1. **Шифрование:**
   * TLS использует симметричные и асимметричные алгоритмы для шифрования данных, передаваемых между клиентом и сервером. Это предотвращает перехват данных третьими сторонами.
2. **Аутентификация:**
   * TLS обеспечивает аутентификацию серверов и клиентов, используя цифровые сертификаты. Это позволяет сторонам проверить подлинность друг друга.
3. **Целостность данных:**
   * TLS использует механизмы целостности (обычно HMAC) для обеспечения того, что данные не были изменены во время передачи.
4. **Обеспечение конфиденциальности:**
   * Шифрование данных гарантирует, что информация остается конфиденциальной и не может быть прочитана третьими лицами.

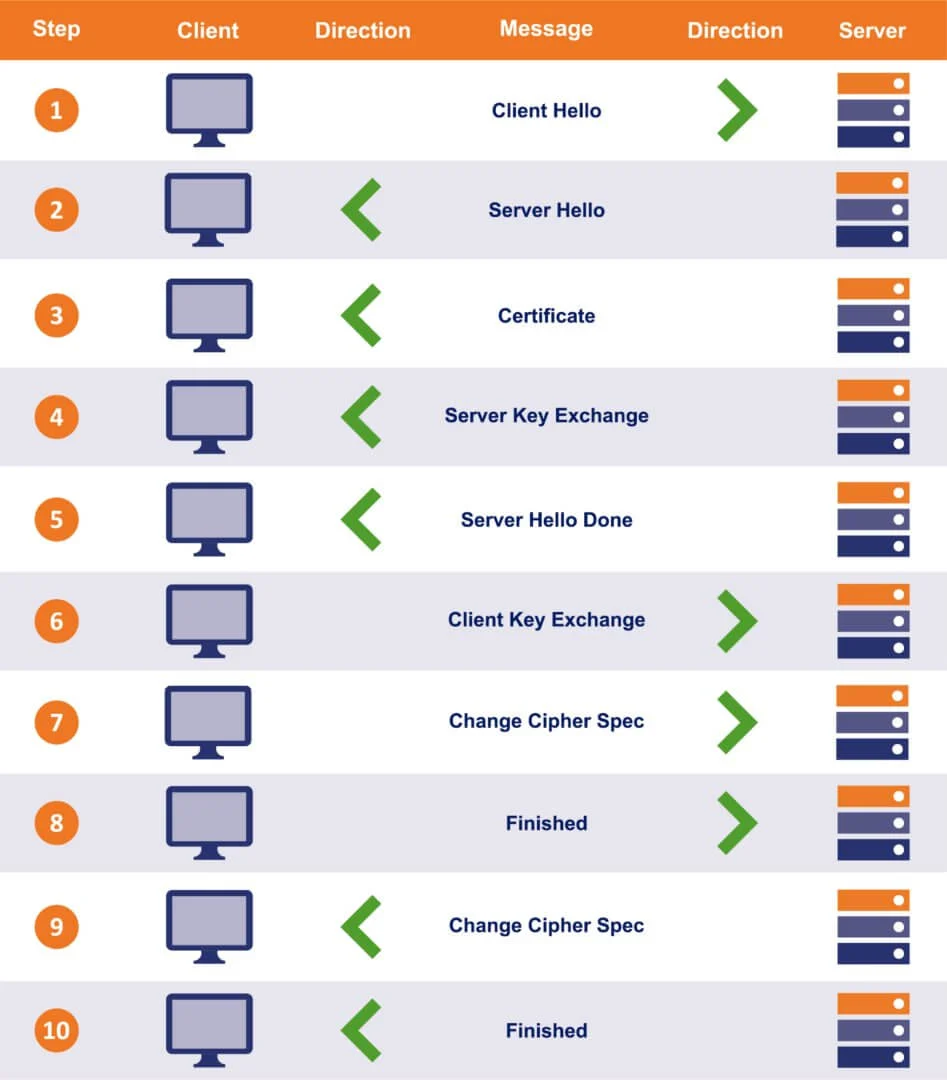
**Назначение TLS**

Протокол TLS (Transport Layer Security) предназначен для обеспечения безопасности данных, передаваемых по сети. Его основные задачи включают:

1. **Защита данных от перехвата и подслушивания.**
2. **Проверка подлинности участников обмена данными.**
3. **Обеспечение целостности передаваемых данных.**

Процедура Рукопожатия TLS (TLS Handshake)

Процедура рукопожатия TLS — это процесс, в ходе которого клиент и сервер устанавливают безопасное соединение. Она состоит из нескольких этапов:



1. **Клиент** выдает запрос серверу (Client Hello).
2. **Сервер** подписывает свой сертификат и высылает клиенту (Server Hello).
3. **Клиент** проверяет сертификат в центре сертификации, которому доверяет.
4. **Клиент** сравнивает данные сертификата с информацией центра сертификации.
5. **Клиент** сообщает серверу, какие ключи шифрования он поддерживает.
6. **Сервер** выбирает подходящую длину ключа.
7. **Клиент** генерирует симметричный ключ, шифрует его открытым ключом.

**Сервер** получает симметричный ключ и расшифровывает его.



Способы вычисления хэша:  
**ECDHE -** Диффи-Хеллман на эллиптических кривых для вычисления общего секрета;

**ECDSA -** аутентификацияданных на этапе установления соединения на основе цифровой подписи на эллиптических кривых;

**AES\_128** – шифрование полезной нагрузки с помощью алгоритма AES с 128-битным ключом в режиме GCM;

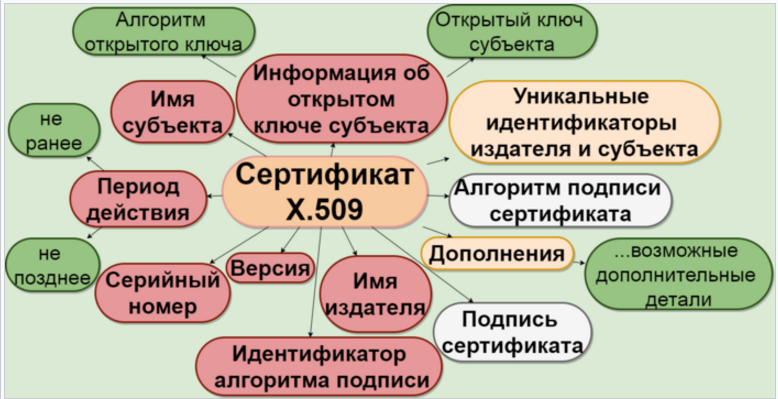
**SHA256** -для хеширования применяется алгоритм SHA с 256-битным ключом

1. **Протокол TLS: X509-сертификаты, назначение.**

**Сертификат X.509:** стандартныйформат хранения и транспортировки атрибутов безопасности; главное – открытый ключ; сертификаты выдают центры сертификации (Certificate Authority, CA). <http://www.thwate.com>, <http://www.Verysign.com>; если это intranet, то можно использовать **Certificate Server Active Directory**; сертификат содержит: имя держателя, адрес, серийный номер сертификата, даты проверки, открытый ключ держателя.

**сертификат содержит**:

* версию
* издателя
* серийный номер
* имя субъекта(кому выдан)
* перечень доменов(имен URI, на которые распространяеца сертификат)
* открытый ключ
* период действия

****

Выдает центр сертификации (СА). <http://www.thwate.com>, <http://www.Verysign.com>. Если это intranet, то можно использовать **Certificate Server Active Directory**. Сертификат содержит: имя держателя, адрес, серийный номер сертификата, даты проверки, открытый ключ держателя

1. **HTTP-аутентификация: Basic - процедура.**

**Заголовки:**

**WWW-Authenticate** — указывает какой метод аутенттификации

[\*\*Realm](https://runebook.dev/ru/docs/http/headers/www-authenticate)\*\* — информирует пользователей о том, какое конкретное имя пользователя/пароль требуется.

**Authorization** — передача информации для авторизации (в формате Base64)



* + - 1. Клиент отправляет запрос
      2. Сервер отвечает 401-м кодом состояния и заголовком WW-Authenticate
      3. Клиент вводит логин/пароль и отправляет его серверу в base64 кодировке
      4. Сервер принимает его логин/пароль и отвечает успешным 200-м кодом состояния
      5. Клиент дальше пользуется логином и паролем при работе с приложением

**Пример заголовка:**

Authorization: Basic dXNlcm5hbWU6cGFzc3dvcmQ=

1. **HTTP-аутентификация: Digest - процедура.**



1. Клиент запрашивает ресурс у сервера.
2. Сервер отвечает с кодом состояния 401 Unauthorized и заголовком **WWW-Authenticate**, указывая, что требуется Digest аутентификация, и включает nonce (случайный числовой вектор), который используется для создания хэша.
3. Клиент вычисляет хэш-значение, основанное на nonce, имени пользователя, пароле, HTTP-методе и URI, и отправляет это значение в заголовке **Authorization: Digest**.
4. Сервер также вычисляет хэш на своей стороне и сравнивает его с хэшем, присланным клиентом. Если значения совпадают, сервер аутентифицирует клиента.

**Пример:**

Authorization: Digest

username="Mufasa",

realm="testrealm@host.com",

nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",

uri="/dir/index.html",

response="e966c932a9242554e42c8ee200cec7f6",

opaque="5ccc069c403ebaf9f0171e9517f40e41"

1. **HTTP-аутентификация: Forms - процедура.**



1. **Запрос страницы входа (GET /Index):** Клиент (браузер) запрашивает начальную страницу, которая возвращает HTML-форму для входа.
2. **Отправка учетных данных (POST /Login):** Пользователь вводит свои учетные данные и отправляет их на сервер с помощью POST-запроса. Данные логина и пароля включены в тело запроса.
3. **Создание и отправка токена (HTTP/1.1 302 Found):** Если сервер успешно проверяет учетные данные, он создает токен сессии (например, закодированный в base64) и отправляет его обратно клиенту в cookie. Код 302 Found указывает на перенаправление, часто на защищенную страницу или основную страницу после входа.
4. **Доступ к защищенному ресурсу (GET/POST /Index):** После успешной аутентификации клиент использует полученный токен для доступа к защищенным ресурсам. Токен отправляется в заголовке cookie. Сервер проверяет этот токен и, если он действителен, предоставляет доступ к ресурсу.
5. **HTTP-аутентификация: Token – процедура.**

**Token**: битовая последовательность, построенная по определенному принципу.

**Token-аутентификация**: аутентификация, использующая token для идентификации пользователя или авторизации операции пользователя

Схема работы предполагает наличие двух серверов:

* **Identity-provider** - сервер, генерирующий token
* **Service-provider** – сервер, предоставляющий сервис клиенту

Service-provider и Identity-provider должны иметь общий секретный ключ для шифрования/проверки token’а

**Активный клиент** – программный код, который может выполнять любые запросы и обрабатывать любые ответы. Обычно – это пользовательская программная реализация http-клиента. Пассивный клиент – браузер





1. **HTTP: принципы кэширование на стороне клиента.**

(Оно же кэширование на стороне браузера)

Кэширование на стороне браузера — это метод хранения веб-ресурсов на локальном устройстве пользователя (например, HTML, CSS, JavaScript, изображения) для уменьшения времени загрузки страниц и снижения нагрузки на сервер. Это помогает ускорить повторные посещения веб-сайтов, делая их более отзывчивыми

Для этого испоьзуются следующие заголовки

**HTTP-заголовки:**

* **Expires:** Указывает дату и время, после которых ресурс считается устаревшим.

Expires: Wed, 21 Oct 2025 07:28:00 GMT

* **Cache-Control:** Более гибкий и современный способ управления кэшированием. Примеры директив:
  + **max-age=<seconds>**: Определяет максимальное время в секундах, в течение которого ресурс считается свежим.

Cache-Control: max-age=3600

* + **no-cache**: Ресурс кэшируется, но проверяется на сервере перед каждым использованием.

Cache-Control: no-cache

* + **no-store**: Ресурс не должен кэшироваться ни в кэше браузера, ни в промежуточных кэшах.

Cache-Control: no-store

* + **public**: Ресурс может кэшироваться любым кэшем, включая промежуточные кэши.

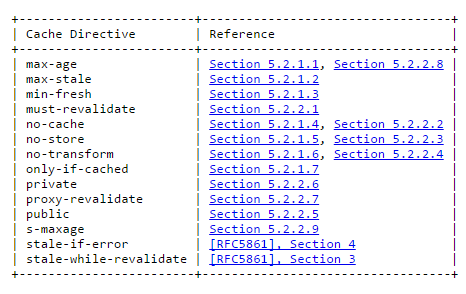
Cache-Control: public

* + **private**: Ресурс может кэшироваться только браузером пользователя.

Cache-Control: private

1. **ETag**:
   * ETag: "abc123" - уникальный идентификатор версии ресурса, который может использоваться для проверки обновлений.
2. **Last-Modified**:
   * Last-Modified: Wed, 30 Nov 2022 12:00:00 GMT - дата и время последнего изменения ресурса на сервере.
3. **If-Modified-Since**:
   * If-Modified-Since: Wed, 30 Nov 2022 12:00:00 GMT - условный заголовок, отправляемый клиентом, чтобы проверить, обновился ли ресурс с момента последнего запроса.

**HTTPS не кэшируется**

****











1. **HTML:стандарты, BOM, DOM, CSS, HTML 5 API.**

**Стандарты**

HTML 4

XHTML 1.0.

**XHTML 1.1.**

* Удаление устаревших элементов и атрибутов
* Поддержка модульного подхода для включения только необходимых элементов.

**HTML 5.**

* Введение новых семантических элементов: <header>, <footer>, <article>, <section>, <nav>, <aside> и другие.
* Улучшенная поддержка мультимедиа: <audio>, <video>, <track> и <source>.

**BOM, DOM**

**Модель DOM (Document Object Model)** DOM представляет собой объектную модель, которую браузеры используют для представления и взаимодействия с HTML-документами. DOM позволяет JavaScript манипулировать содержимым, структурой и стилями документов.

\*\***Модель BOM (Browser Object Model)** \*\*BOM предоставляет интерфейсы для взаимодействия с браузером вне контекста HTML-документа. BOM включает объекты, такие как **window**, **navigator**, **screen**, **location**, и **history**.

[BOM (Browser Object Model)](https://learn.javascript.ru/browser-environment#bom-browser-object-model)

Объектная модель браузера (Browser Object Model, BOM) – это дополнительные объекты, предоставляемые браузером (окружением), чтобы работать со всем, кроме документа.

**CSS**

Каскадная таблица стилей.

**CSS (Cascading Style Sheets)** – это язык стилей, используемый для описания внешнего вида и форматирования документов, написанных на языках разметки, таких как HTML. Основная задача CSS – отделить содержимое документа от его представления, что позволяет улучшить управление стилями

**Типы селекторов**:

**Элемент (Типовой селектор)**: применяет стиль ко всем элементам указанного типа.

p { color: blue; }

​

**Идентификатор (ID-селектор)**: применяет стиль к элементу с конкретным идентификатором.

#header { background-color: grey; }

​

**Класс (Классовый селектор)**: применяет стиль к элементам с указанным классом.

.highlight { background-color: yellow; }

​

**Атрибутный селектор**: применяет стиль к элементам с указанными атрибутами.

input[type="text"] { border: 1px solid black; }

​

**Комбинированные селекторы**: позволяют применять стиль к элементам, соответствующим нескольким условиям.

div p { color: red; }

**HTML 5 API**

**Canvas API**:

* Позволяет рисовать графику, анимацию и обрабатывать изображения непосредственно в HTML-странице с использованием элемента <canvas>.

**Web Storage API**:

* Включает localStorage и sessionStorage для хранения данных в браузере.
* localStorage сохраняет данные без срока действия, а sessionStorage - только на время текущей сессии.

**Geolocation API**:

* Позволяет определять географическое положение устройства.

**Web Workers API**:

* Позволяет выполнять скрипты в фоновом потоке, не блокируя основной поток пользовательского интерфейса.

**File API**:

* Позволяет работать с файлами, выбранными пользователем через элемент <input type="file">.

**IndexedDB API**:

* База данных в браузере для хранения больших объемов структурированных данных.

**WebRTC API**:

* Позволяет осуществлять аудио- и видеосвязь в реальном времени прямо в браузере.

**Notifications API**:

* Позволяет веб-приложениям отображать уведомления на рабочем столе пользователя.

1. **AJAX: асинхронный запрос, объекты XMLHTTPREQUEST и FETCH, форматы передачи данных.**

**Asynchronous JavaScript and XML** – асинхронный JavaScript and XML – методология (подход) построения динамических приложений, при которых не осуществляется полная перезагрузка html-страниц. AJAX: XMLHTTPRequest, DOM, формат: XML и JSON.

**Асинхронные запросы:** запрос, при котором поток, выдавший http-запрос, не блокируется до поступления запроса; для обработки ответа применяется функция обратного вызова

**Синхронный http-запрос:** запрос, при котором поток, выдавший http-запрос, блокируется до поступления запроса

**XMLHTTPRequest**: Это объект, предоставляемый браузерами для выполнения асинхронных HTTP-запросов к серверу без перезагрузки страницы. Он был изначально представлен в Internet Explorer 5 и стал стандартом в JavaScript

**Fetch API**: Это современный API для выполнения сетевых запросов в браузере. Он предоставляет более гибкий и мощный способ выполнения запросов по сравнению с XMLHttpRequest

**Форматы:** JSON, XML

**Порядок создания запроса**:

* Создать XMLHttpRequest

let xhr = new XMLHttpRequest();

* Инициализировать его (async – true, sync – false (default false))

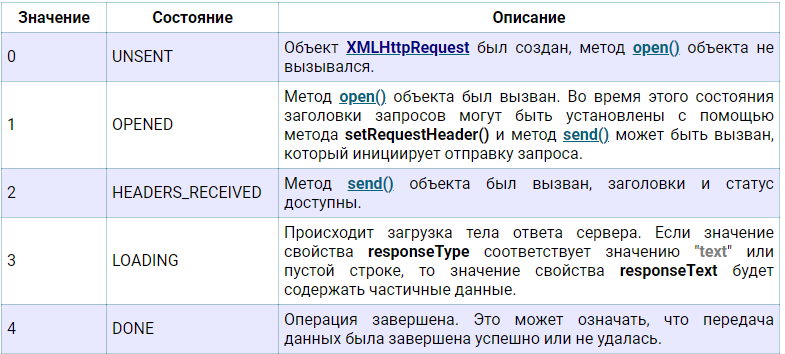
xhr.open(method, URL, [async, user, password])

* Послать запрос

xhr.send([body])

* Слушать события на xhr, чтобы получить ответ.

onload, onprogress, onerror, onReadyStateChange





1. **Протокол WebSocket: назначение, применение.**

**WebSockets** — это протокол, который обеспечивает двустороннюю(дуплексную) связь между клиентом и сервером через одно соединение TCP (устанавливает постоянное соединение). Он предназначен для приложений, которые требуют постоянного обмена данными, таких как чаты, игры, финансовые приложения, и реального времени обновление данных.

**Принципы работы WebSockets**

1. **Установка соединения**:
   * Инициация начинается с HTTP-запроса от клиента к серверу с использованием метода **GET** и заголовка **Upgrade: websocket**.
   * Сервер отвечает подтверждением (статус 101 Switching Protocols), что устанавливает WebSocket соединение.
2. **Двусторонняя связь**:
   * После установки, клиент и сервер могут свободно обмениваться сообщениями в обе стороны без необходимости повторной отправки заголовков HTTP.
3. **Постоянное соединение**:
   * Соединение остается открытым, что позволяет непрерывный обмен данными, снижая задержки и накладные расходы на установку новых соединений.
4. **Формат данных**:
   * Сообщения WebSocket могут быть как текстовыми, так и бинарными, что позволяет гибкость в типах передаваемых данных.

### ****Применения WebSockets****

WebSockets находят применение в:

* **Чат-приложениях**: Обеспечивают моментальный обмен сообщениями между пользователями.
* **Реального времени обновлениях**: Например, в приложениях для отслеживания фондовых рынков или спортивных событий.
* **Онлайн-играх**: Обеспечивают низкую задержку и высокий уровень взаимодействия между игроками.

1. **Web-приложение: определение, назначение, применение.**

**Web-приложение:** клиент-серверное приложение, применяющее для обмена данными протокол HTTP; может быть просто web-приложением (HTML+HTTP) или web-службой (API, HTTP-транспорт, формат XML, JSON





-



-----



-------

**Web ресурс приложения:** сущность, расположенная на стороне сервера и имеющая URL/URI, к которой можно сделать http-запрос и получить http-ответ. Одно web-приложение представлено одним или более ресурсов.

**Web ресурсы приложения: статические** - отправляются клиенту без изменения (html-страницы, рисунки, видео-файлы, …), **динамические** – динамически (программно) формируются на сервере и отправляются клиенту (сервлеты, JSP, http-обработчики, aspx-страницы,…). Ресурс может быть статическим относительно сервера и динамическим относительно клиента (html-страницы с JavaScript).

1. **Web-сервер: основные объекты.**

Сессия, Куки, Конфиг. Файл, Контекст, Слушатели, Кэш,Фильтры

(пиздец 21-27 вопросы в одном)

1. **Web-сервер: объекты Request и Response.**

**Запрос(Request):** серверныйобъект, который образуется в результате обработки сервером http-запроса, поступающего от клиента и передается серверному программному коду для обработки. Содержит: всю информацию из http-запроса: метод, коллекция заголовков, коллекция параметров, поток данных … Обычно объект Request предоставляет возможность хранить данные в формате ключ/значение

****

**Ответ(Response):** серверныйобъект, который автоматически формируется сервером, при получении http-запроса (одновременно с объектом Request), заполняется данными серверными программным кодом, преобразуется в http-ответ и отправляется клиенту. Содержит: всю информацию, которая должна быть помещена в http-ответ: статус, коллекция заголовков, поток данных



1. **Web-сервер: объект Sesson и его жизненный цикл.**

**Объект сессии это способ сохранять состояния**

**Сессия (Session):** серверный объект, хранящий информацию о соединении с клиентом, создается при первом обращении время жизни: **timeout** (системный параметр, обычно равен 10 – 30 минутам) – максимальное время между запросами клиента. Если **timeout** превышен, то Session разрушается и при следующем запросе создается новый экземпляр. Каждая сессия имеет собственный идентификатор (**Session ID**, 16 или более байт). Каждый Requestпринадлежит, какой-то сессии (имеет ссылку на объект Session или содержит Session ID). Обычно объект Session предоставляет приложению возможность хранить данные в формате ключ/значение

**жизненный цикл**

создается при первом обращении, умирает в конце срока действия



1. **Web-сервер: объект Filter/Middleware его жизненный цикл.**

**Фильтр (Filter)**: серверный объект – препроцессор запроса, предназначен для предварительной обработки объекта Request. К одному ресурсу может быть построена цепочка фильтров, последний в цепочке – ресурс. Фильтр может прервать цепочку и сам сформировать ответ клиенту. Один и тот же фильтр может быть применен к нескольким ресурсам. В качестве параметров фильтр получает объекты Request и Response, которые от передает дальше по цепочке или обрывает цепочку и заполняет объект Response



1. **Web-сервер: объект Context его жизненный цикл.**

**Контекст web-приложения**: серверный объект, предназначенный для хранения информации об одом web-приложении, общий для всех файлов. Как правило, формируется сразу при загрузке web-сервера, основные данные (параметры приложения) копируются из конфигурационного файла приложения, общий для всех сессий приложения,  Обычно контекст предоставляет возможность хранить данные в формате ключ/значение.

Контекст сервера хранит информацию между сессиями.

**жизненный цикл**

резидентный — находится в оперативной памяти (на протяжении работы сервера)



1. **Web-сервер: принципы кэширования.**

**Кэш (Cache):** серверный объект, предназначенный для временного хранения данных с целью ускорения выполнения запроса. Кэширование – процессы записи и извлечения данных в/из Cache. Различают кэширование данных и кэширование вывода. **Кэширование данных** – кэширование часто используемых данных. **Кэширование вывода** – кэширование объекта Response.



1. **Web-сервер: объект connection pool и его жизненный цикл.**

**Пул соединений с базой данных:** несколько предварительно и постоянно открытых соединений с  сервером СУБД, которые используют приложения.  Выбор подключения из пула по Open, возврат в пул Close. Если все подключения пула заняты, запрос на соединение ставится в очередь. Применение пула позволяет увеличить производительность за счет отсутствия процесса подключения к серверу. Альтернатива: держать постоянное открытое соединение для каждого подключения (в общем случае для web-приложения не приемлемо) или открывать/закрывать соединение при каждом запросе (большие накладные расходы на установку соединения).

**Объект Connection Pool** (пул соединений) - это механизм, который используется в веб-серверах для управления соединениями с ресурсами. Его основное назначение - уменьшить накладные расходы на установку и разрыв соединений, что повышает производительность и эффективность работы приложения.

**Жизненный цикл Connection Pool:**

1. **Инициализация**: На этом этапе создается пул соединений, обычно в начале работы сервера или при первом обращении к базе данных. Количество соединений в пуле определяется параметрами конфигурации и требованиями приложения.
2. **Запрос соединения**: Когда приложение нуждается в выполнении операции, требующей соединения с базой данных (например, выполнение SQL-запроса), оно запрашивает свободное соединение из пула.
3. **Выдача соединения**: Пул соединений предоставляет приложению доступное соединение из своего пула. Если все соединения заняты, пул может создать новое соединение, если это допускается настройками, или ждать, пока не освободится соединение.
4. **Использование соединения**: После получения соединения приложение выполняет необходимые операции, такие как выполнение запросов к базе данных.
5. **Возврат соединения в пул**: После завершения операции приложение возвращает соединение обратно в пул, чтобы оно могло быть использовано другими запросами.
6. **Разрыв соединения**: В некоторых случаях соединение может быть разорвано из-за истечения тайм-аута, ошибок сети или других причин. Пул соединений может периодически проверять соединения на активность и пересоздавать их при необходимости.
7. **Завершение работы**: При завершении работы сервера или приложения пул соединений освобождает все ресурсы, связанные со собой, включая активные соединения, и завершает свою работу.
8. **Web-сервер: объект listener и его жизненный цикл.**

**Слушатели событий (Lister):** серверные объекты – для обработки событий жизненного цикла web-приложения.

Событие start сервера/завершение сервера, образование/уничтожение сессии и другие.(авторизация, аутнификация и т.д.)

У сервера есть набор событий которые разрешают вклинится своим кодом в них.

Загружаются вместе с сервером, информация записывается в контекст.

**жизненный цикл**

Создётся при инициализации сервера

Умирает при выключении сервера

Хранится в контексте



1. **Web-браузер: структура, принципы функционирования, основные объекты.**

**Браузер** прикладное программное средство, предназначенное:

* формирования и выполнения http-запросов;
* получения и обработки http-ответов;
* отображения компьютерных файлов;
* интерпретации js-файлов.

**Браузерное  программирование** – разработка приложений, работающих в рамках браузера.

**BP: User** **Interface** – модуль пользовательского интерфейса, позволяющий пользователю управлять работой браузера (вводить URI, движение вперед/назад по истории, закладки и пр.)

**BP: Browser** **Engine** – модуль управления браузером: управление закладками, скачивание, проверка орфографии, поиск на странице, …).

**BP: Render** **Engine** – модуль отображения контента: WebKit, Blink, Gecko. Стандарты HTML, CSS, XML, DOM

**BP: Data Persistence** – модуль обеспечивающий работу с хранилищем: Local storage, Session Storage, Cookies, WebSQL, IndexedDB, File System, AppCache, Service Workers.

**BP: JS Engine** – модуль-интерпретатор JavaScript: V8,Gecko, SpiderMonkey, Rhino, Tamarin, Chakra, Carakan. Стандарт JS и Browser API.

**BP: Networking** – модуль взаимодействия с сетью.

**BP:** программисту браузер представляется в виде модели **Browser** **Object** **Model**(BOM), модели **Document** **Object** **Model** (DOM) и **Browser** **API**.



1. **Web-браузер: назначение и возможности IndexDB API.**

**IndexDB**/HTML5 - программный интерфейс для работы с встроенной в браузер NoSQL БД

**IndexDB** СУБД, встроенная в web-browser, NoSQL, система индекированных хранилищ, формат данных JSON, JavaScript API, асинхронная работа, транзакционная модель, поддерживается курсор, использует события DOM об уведомлении javascrpt-приложения (error/succes)

**Возможности**:  
JS API, проверка браузера

создание и модификация БД

запись в БД

читать из БД по идентификатору

удалить из БД по идентификатору

читать из БД с помощью курсора без индекса

читать из БД по индексу с помощью курсора, режимы (IDBKeyRange

1. **Протокол WebDAV: назначение, принцип применения.**

**WebDAV**: Web Distributed Authoring and Versioning – расширение протокола HTTP/HTTPS, поддерживающее совместную работу по управление файлами на удаленных web-северах; применяется для создания сетевой файловой системы; в системах документооборота (document management system).

**WebDAV**: **GET** – скачать файл.

**WebDAV**: **PUT** – загрузить файл на сервер.

**WebDAV**: **DELETE** – удалить серверный объект.

**WebDAV RFC 5618: PROPFIND** – получить свойства серверного объекта в XML-формате.

**WebDAV RFC 5618: PROPPATCH** – изменить свойства серверного объекта.

**WebDAV RFC 5618: MKCOL** – создать папку на сервере.

**WebDAV RFC 5618: COPY** – копировать на сервере.

**WebDAV RFC 5618: MOVE** – переместить на сервере.

**WebDAV RFC 5618: LOCK** – заблокировать  серверный объект.

**WebDAV RFC 5618: UNLOCK** – разблокировать серверный объект.

**Назначение**

* **Управление документами**:
  + WebDAV позволяет пользователям создавать, изменять и удалять документы на сервере, обеспечивая удаленный доступ к файлам.

**Принцип применения**

1. **WebRTC: назначение, перечень протоколов.**

**WebRTC (Web Real-Time Communication)** — это набор технологий, позволяющих реализовывать потоковую передачу аудио, видео и данных в реальном времени между браузерами и другими устройствами через Интернет без использования плагинов или сторонних программ.

WebRTC - основан на 2 протоколах RTP и RTCP.

**Назначение WebRTC:**

1. **Видеосвязь и аудиосвязь**: Позволяет организовывать видео- и аудио-конференции прямо в браузере без дополнительных программ.
2. **Потоковая передача данных**: Позволяет обмениваться данными между браузерами в реальном времени для различных сценариев, таких как игры, стриминг и другие веб-приложения.

**Перечень протоколов, используемых в WebRTC:**

1. **ICE (Interactive Connectivity Establishment)**: Протокол, используемый для обнаружения и выбора наиболее эффективного пути для установления соединения между устройствами.
2. **STUN (Session Traversal Utilities for NAT)**: Протокол, позволяющий определить публичный IP-адрес и порт клиента, находящегося за NAT.
3. **TURN (Traversal Using Relays around NAT)**: Протокол, используемый для обеспечения связности передачи данных между клиентами, находящимися за различными типами NAT.
4. **DTLS (Datagram Transport Layer Security)**: Протокол, обеспечивающий безопасную передачу данных в реальном времени посредством шифрования и аутентификации.
5. **SRTP (Secure Real-time Transport Protocol)**: Протокол, обеспечивающий безопасную передачу аудио- и видео-данных в реальном времени.

**НОВЫЕ СУКА ВОПРОСЫ УРА:**

**36. Сервер NGINX: назначение и применение.**

NGINX — это высокопроизводительный HTTP-сервер и обратный прокси-сервер, а также почтовый (IMAP/POP3) прокси-сервер. Он был создан для решения проблем, связанных с масштабированием и производительностью, и стал одним из самых популярных серверов в мире.

**Назначение:**

HTTP-сервер: NGINX может работать как полноценный веб-сервер, обслуживая статические файлы, обрабатывая динамические запросы через CGI, FastCGI, SCGI, uWSGI и другие протоколы

forward proxy(прямой прокси, кэширование, скрыть ip-адрес клиента), reverse proxy (обратный прокси, балансировка нагрузки

обслуживание статических запросов

обратное проксирование

**Обратный прокси-сервер**:

* NGINX часто используется как обратный прокси-сервер для распределения входящих запросов между несколькими бэкенд-серверами, что позволяет балансировать нагрузку и повышать отказоустойчивость.

**Балансировка нагрузки**:

NGINX поддерживает различные методы балансировки нагрузки, включая round-robin, least connections, IP hash и другие. Это позволяет эффективно распределять трафик между серверами

**Кэширование**:

* NGINX может кэшировать ответы от бэкенд-серверов, что значительно снижает нагрузку на серверы приложений и ускоряет доставку контента.

**Почтовый прокси-сервер**:

NGINX может работать как почтовый прокси для IMAP, POP3 и SMTP, обеспечивая балансировку нагрузки и безопасность

**Применение NGINX**

* **Обслуживание статического контента**: NGINX отлично справляется с доставкой статических файлов, таких как изображения, видео и другие медиафайлы.
* **Обратное проксирование и балансировка нагрузки**: Обеспечивает равномерное распределение запросов между несколькими серверами для повышения производительности и отказоустойчивости.
* **Кэширование**: Уменьшает нагрузку на серверы приложений и ускоряет доставку контента пользователям.
* **Безопасность**: Позволяет использовать SSL/TLS терминаторы для шифрования трафика, а также настраивать защиту от DDoS-атак и других угроз.
* **Мультипротокольная поддержка**: Может работать с различными протоколами и модулями для расширения функциональности.

1. **Протокол OAuth 2.0: назначение, принципы применения.**

**OAuth 2** — это протокол авторизации, предназначенный для организации доступа клиентских приложений к ресурсам, или данным учетных записей, пользователя на другом сервисе. В качестве клиентских приложений выступают веб-сервисы, мобильные и десктопные приложения. В качестве сервисов — mail.ru, GitHub, Bitbucket и др. Протокол используют разработчики сторонних приложений.

**Используемые роли в OAuth 2**

В рамках описываемого протокола выделяются следующие типы ролей:

* владелец (пользователь): авторизует клиентское приложение на доступ к данным своего аккаунта;
* сервер ресурсов/API: здесь располагаются данные пользовательских аккаунтов, а также бизнес-логика авторизации, отвечающая за выдачу новых OAuth-токенов и проверку их подлинности при обращении клиентских приложений к ресурсам. Целесообразно объединять эти роли, так как физически это один сервис;
* клиентское приложение: собственно сервис, которому пользователь делегирует права доступа к своим данным на сервере ресурсов. Пользователь должен авторизовать приложение, а со стороны сервера API оно должно получить подтверждение в виде ключа (токена) доступа.



* Пользователь перенаправляется на страницу авторизации, где у него запрашиваются разрешения для приложения на работу с данными его аккаунта.
* После предоставления необходимых разрешений пользователь попадает на callback URL — адрес, указанный при регистрации приложения, предназначенный для завершения авторизации. При этом происходит подстановка кода авторизации в GET-параметры адреса.
* Сервер клиентского приложения формирует POST-запрос к серверу авторизации API с кодом авторизации в качестве параметра.
* Сервер авторизации проверяет код и возвращает приложению токен доступа (access token).
* Используя токен, приложение авторизуется на сервере API и получает доступ к запрашиваемым пользовательским ресурсам.



**25. Web-сервер: сохранение состояния.**

Веб-серверы по своей природе являются статeless, что означает, что каждый HTTP-запрос обрабатывается независимо от других. Однако, для создания интерактивных и персонализированных веб-приложений, важно сохранять состояние между запросами. Существует несколько механизмов для сохранения состояния в веб-приложениях:

**1. Сессии (Sessions)**

Сессии позволяют сохранять данные на сервере между HTTP-запросами. Каждый клиент получает уникальный идентификатор сессии (Session ID), который передается между клиентом и сервером, обычно через cookie.

**2. Cookie**

Cookie — это небольшие файлы, которые веб-сервер отправляет клиенту, и которые клиент хранит и отправляет обратно с каждым запросом к этому серверу. С их помощью можно сохранять данные на стороне клиента

**3. Local Storage и Session Storage**

Local Storage и Session Storage — это веб-хранилища, которые позволяют сохранять данные в браузере клиента. Local Storage сохраняет данные на постоянной основе, тогда как Session Storage — только на время сессии.

**17. HTTP: сохранение состояния на стороне клиента.**

Зав. каф. ИСиТ В.В. Смелов