# -== IU7-Web Bilet #1 ==-

**1. Технологии толстого клиента. Варианты организации персистентного хранилища данных на клиенте (localStorage, IndexedDB).**

***Толстый клиент*** (rich client) в архитектуре клиент-сервер — приложение, обеспечивающее (в противовес тонкому клиенту) полную функциональность и независимость от центрального сервера. Часто сервер в этом случае является лишь хранилищем данных, а вся работа по обработке и представлению этих данных переносится на машину клиента.

(+) Достоинства:

* Независимость от сервера
* Режим многопользовательской работы (?)
* Работает при обрыве связи с сервером
* Высокое быстродействие (зависит тока от аппаратных средств клиента)

(-) Недостатки:

* Большой размер дистрибутива
* Часто платформозависимый
* Мб сложный процесс установки, настройки и обновлений
* Мб неактуальность данных

***LocalStorage***: позволяет хранить пары ключ-значение в браузере

Данные, которые в них записаны, сохраняются после обновления страницы (в случае sessionStorage) и даже после перезапуска браузера (при использовании localStorage).

Отличие от куки: объекты веб-хранилища не отправляются на сервер при каждом запросе, – сервер не может манипулировать объектами хранилища через HTTP заголовки. Хранилище привязано к источнику (домен/протокол/порт).

Основные особенности:

* объект один на все вкладки и окна в рамках источника (один и тот же домен/протокол/порт).
* Данные не имеют срока давности, по которому истекают и удаляются.

Хранение пар ключ-значение в браузере:

* key и value должны быть строками.
* Лимит 2 Мб+, зависит от браузера.
* Данные не имеют «времени истечения»
* Данные привязаны к источнику (домен/протокол/порт).

***IndexedDB*** - это встроенная база данных, более мощная, чем localStorage.

* доступны несколько типов ключей, а значения могут быть (почти) любыми.
* Поддерживает транзакции для надёжности.
* Поддерживает запросы в диапазоне ключей и индексы.
* Позволяет хранить больше данных, чем localStorage.

Интерфейс основан на событиях. Может быть множество баз данных с различными именами, но все они существуют в контексте текущего источника (домен/протокол/порт).

***IndexedDB*** имеет встроенный механизм «версионирования схемы», который отсутствует в серверных базах данных.

Работает на стороне клиента, в браузере, и нет прямого доступа к данным.

Чтобы сохранить что-то в IndexedDB, нам нужно хранилище объектов.

Хранилище объектов – это основная концепция IndexedDB. Здесь хранятся данные. В базе данных может быть множество хранилищ: одно для пользователей, другое для товаров и так далее.

Несмотря на то, что название – «хранилище объектов», примитивы тоже могут там храниться. Мы можем хранить почти любое значение, в том числе сложные объекты.

**2. Базы данных. Понятие СУБД. Реляционные и нереляционые базы данных. Сферы применения, достоинства и недостатки.**

***База данных*** - совокупность данных, организованных в соответствии со схемой данных, манипулирование которыми выполняют с правилами средств моделирования данных.

***Система управления базами данных СУБД*** - совокупность программных и лингвистических средств, обеспечивающих управление созданием и использованием баз данных.

Основными функциями СУБД являются:

* управление представлением модели данных и данными в ней;
* управление данными в памяти;
* журнализация изменений, резервное копирование и восстановление базы данных (пункт для галочки, так как на самом деле СУБД можно завести и локальную без всех этих возможностей [см. На структуру access]);
* поддержка языков БД.

***Существует 4 основных типа моделей организации данных:***

* иерархическая (NoSQL);
* сетевая (NoSQL);
* реляционная (SQL);
* объектно-ориентированная (NoSQL).

***Реляционная модель*** использует организацию данных в двумерных таблицах. Любая такая таблица должна обладать следующими свойствами:

* все элементы однородные;
* каждый столбец имеет уникальное имя;
* порядок следования столбцов и строк не имеет значения.

Реляционные соответствует ACID (атомарность, непротиворечивость, изолированность, долговечность).

***NoSQL*** - ряд подходов, направленных на реализацию моделей БД, имеющих существенные отличия от используемых в традиционных реляционных СУБД. В отличие от реляционных, часто имеют более высокую скорость доступа к данным или распределенную архитектуру, которая позволяет хранить неограниченное количество данных или хранить неструктурированные объекты данных. Типы NoSQL (представление на низком уровне):

* Документно-ориентированные (иерархия);
* Графовые (сеть);
* “Ключ-значение” (гибрид, нужно смотреть конкретную СУБД, однако в памяти представляется хэш-таблицей);
* Иные

“Ключ-значение” - ассоциативный массив с уникальными ключами. Не требует никаких схем построения БД, нет никаких связей между значениями. Подобные хранилища используют в тех случаях, когда полностью отсутствуют связи между ячейками.

Документо-ориентированная ДБ представляет собой систему хранения иерархических структур данных, имеющих структуру дерева. Такие БД применяются в задачах, где требуется упорядоченное хранение информации, но нет множества связей между данными.

Графовые БД применимы в случаях, когда связи между данными имеют такую же ценность, как и сами данные. Графовые БД лучше всего подходят для реализации проектов, предполагающих естественную графовую структуру данных - социальных сетей и для создания семантических паутин

Преимущества:

* Возможность хранения больших объемов неструктурированной информации.
* лучше поддаются масштабированию
* Ключевые преимущества NoSQL баз в распределенных системах заключаются в процедурах шаринга и репликации.
* Использование облачных вычислений и хранилищ.
* Быстрая разработка

Недостатки:

* Приложение сильно привязывается к конкретной СУБД
* Ограниченная емкость встроенного языка запросов
* Процесс создания реляционного хранилища включает в себя этап проектирования модели данных.
* Трудности быстрого перехода с одной нереляционной базы данных на другую.
* Приходится разрабатывать собственные инструменты для работы с БД.

Выбор SQL-БД:

* соответствие ACID (атомарность, непротиворечивость, изолированность, долговечность);

Выбор NoSQL-БД:

* данные плохо структурированы или структура часто меняется;
* быстродействие (в основе простые структуры данных [хэши, деревья и т.д.], позволяет не производить склейку как в SQL);
* масштабируемость (спорный пункт, так как все зависит от устойчивости структуры).

3. Задача на: Nginx/REST/Архитектуру/Дизайн/Верстку/JS. В основном - на клиентский JS.

# 

# -== IU7-Web Bilet #2 ==-

**1. HTML. Язык разметки гипертекста. История. Структура HTML-документа. Пример.**

Очень крутая история развития HTML, тут написано дофига, но подумайте, что важнее Бекасову - ваши познания в исторических датах или умение работать с html? Промотайте это глазами и идите в конец этого вопроса, там основная суть.

*История*

*1991 Британец Тимоти Джон Бернерс-Ли в Женевском ЦЕРНе изобрел язык гипертекстовой разметки*

*1993 - HTML 1.2 В этой версии, из её сорока с копейками тегов уже появилось аж 3 тега, которые намекали на какое-то визуальное оформление документа (например,полужирный курсив). Остальные же теги служили исключительно для логической разметки.*

*22 сентября 1995 – Версия 2.0 Процесс разработки и утверждения новой версии был очень неспешным, а единственным заметным улучшением новой версии стали: запросы (например, поиск по ключевым словам), формы для передачи данных с компьютера на сервер (например, ввести дату рождения или выбрать один из нескольких вариантов в опроснике)*

*Март 1995 – начало работы над HTML 3.0 Первый вариант стандарта включал в себя много интересностей: теги для создания таблиц, разметки математических формул, обтекание изображений текстом и др.*

*14 января 1997 – HTML 3.2 вышла спустя месяц после утверждения CSS, и была уже полностью приспособлена к взаимодействию с таблицами стилей.*

*18 декабря 1997 – HTML 4.0 версия включала поддержку фреймов, скриптов, общую процедуру внедрения разных объектов. Также в ней были усовершенствованы таблицы и формы, что кроме прочих плюсов обеспечивало большую доступность для людей с физическими недостатками.*

*24 декабря 1999 – HTML 4.01 В этой версии слегка подправили объекты, формы и изображения, пофиксили баги и в целом создали более стабильную версию, которой пользовались веб-разработчики более 10 лет.*

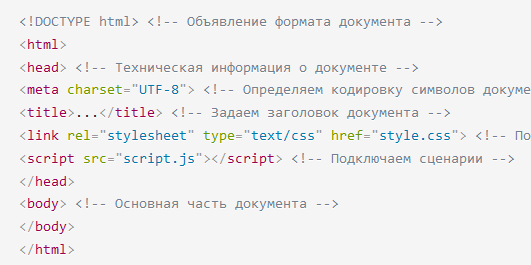
*28 октября 2014 – HTML5 Новая версия сделала синтаксис более строгим по сравнению с предыдущей. Улучшилась поддержка мультимедиа-технологий. Появились 28 новых структурных элементов, благодаря которым код стал более понятным. Исключена еще часть устаревших тегов Стало больше внимания уделяться поддержке скриптов, например javascript*

**Структура HTML-документа**

**HTML (Hyper Text Markup Language)**- язык разметки гипертекста. Основой HTML-документа являются теги. Каждый элемент обозначается в документе открывающимся и закрывающимся тегом.

Браузер просматривает HTML-документ, выстраивая его структуру (DOM) и отображая ее в соответствии с инструкциями, включенными в этот файл (таблицы стилей, скрипты). Если разметка правильная, то в окне браузера будет отображена HTML-страница, содержащая HTML-элементы.

Пример:



Что происходит в примере?

DOCTYPE отвечает за корректное отображение веб-страницы браузером.

Элементы, находящиеся внутри элемента <html>, образуют дерево документа, так называемую объектную модель документа, DOM (document object model). DOM - это представление HTML-документа в виде дерева тегов (DOM - это содержимое страницы в виде объектов, которые можно менять). Теги являются узлами-элементами (или просто элементами). Они образуют структуру дерева: <html> – это корневой узел, <head> и <body> его дочерние узлы и т.д. Всего существует 12 типов узлов… (я углубляться не буду в такие подробности).

****

Все, что находится за пределами элемента <html>, не воспринимается браузером как HTML-код и никак им не обрабатывается.

Элемент <head> содержит метаданные о странице: ключевые слова для поисковых машин, кодировку символов и т. д.

В разделе <body> располагается всё содержимое документа.

**2. MVC-фреймворки. Понятие фреймворка. Примеры. Типовая структура MVC-фреймворка.**

**Веб-фреймворки** - это наборы функций, объектов, правил и других конструкций кода, предназначенных для решения общих проблем, ускорения разработки и упрощения различных типов задач, стоящих в конкретной области.

AngularJS является JavaScript-фреймворком с открытым исходным кодом, разрабатываемым Google. Предназначен для разработки одностраничных приложений…. Backbone.js придает структуру веб-приложениям с помощью моделей с биндингами по ключу и пользовательскими событиями, коллекций с богатым набором…

EmberJS Главной особенностью EmberJS является привязка данных. Вы просто создаете переменную, и, когда значение этой переменной изменяется, обновляется любая…

Knockout.js — это популярная JavaScript библиотека, позволяющая реализовать Model-View-View Model MVVM паттерн на клиенте.

MVC расшифровывается как модель-представление-контроллер (от англ. modelview-controller). Это способ организации кода, который предполагает выделение блоков, отвечающих за решение разных задач. Один блок отвечает за данные приложения, другой отвечает за внешний вид, а третий контролирует работу приложения.

**Компоненты MVC**

**Модель** — этот компонент отвечает за данные, а также определяет структуру приложения. Например, если вы создаете To-Do приложение, код компонента model будет определять список задач и отдельные задачи.

**Представление** — этот компонент отвечает за взаимодействие с пользователем. То есть код компонента view определяет внешний вид приложения и способы его использования.

**Контроллер** — этот компонент отвечает за связь между model и view. Код компонента controller определяет, как сайт реагирует на действия пользователя. По сути, это мозг MVCприложения. Контроллер решает задачи, он не должен включать в себя бизнес-логику приложения. Не стоит заниматься существенным в контроллере, иначе возникает FSUC Fat Stupid Ugly Controllers).

**MVC** — подход к проектированию приложения, который предполагает выделение кода в блоки модель, представление и контроллер. Контроллер обрабатывает входящие запросы. Модель достаёт из базы данных информацию, нужную для выполнения конкретных запросов. Представление определяет результат запроса, который получает пользователь.)

ASP.NET MVC Framework — фреймворк для создания веб-приложений, который реализует шаблон Model-view-controller.



# -== IU7-Web Bilet #3 ==-

**1. Классификация серверов по типу организации ввода-вывода. Примеры веб-серверов на каждый тип. Достоинства и недостатки.**

Процесс — экземпляр программы во время выполнения, независимый объект, которому выделены системные ресурсы.Поток использует то же самое пространства стека, что и процесс, а множество потоков совместно используют данные своих состояний. Как правило, каждый поток может работать (читать и писать) с одной и той же областью памяти, в отличие от процессов, которые не могут просто так получить доступ к памяти другого процесса. Поток — определенный способ выполнения процесса. Когда один поток изменяет ресурс процесса, это изменение сразу же становится видно другим потокам этого процесса. Любая пользовательская программа запускается внутри процесса, а код выполняется в контексте потока.

**Синхронный блокирующий ввод-вывод,** режим реализации сервера - одно соединение на поток, то есть, когда у клиента есть запрос на соединение, серверу необходимо запустить поток для обработки. Если это соединение ничего не делает, это приведет к ненужным накладным расходам потока. Конечно, это может быть через поток Улучшен механизм бассейна.

Примером синхронных, блокирующих операций является то, как некоторые веб-серверы, подобные тем, что в Java или PHP обрабатывают IO или сетевые запросы. Если ваш код читается из файла или базы данных, ваш код “блокирует” все после его выполнения. В этот период ваша машина удерживает память и время обработки для потока, который ничего не делает.

Чтобы удовлетворить другие запросы, пока этот поток застопорился, зависит от вашего программного обеспечения. То, что делает большинство серверных программ, порождает больше потоков для удовлетворения дополнительных запросов. Это требует большего объема памяти и большей обработки.

Преимущество - легкость реализации.

Недостатком такого подхода является линейный рост числа потоков и объема требуемой памяти при увеличении числа обслуживаемых соединений. Если потоков будет создано очень много (десятки тысяч), то операционная система сервера будет занята переключением контекстов потоков, а на полезную работу времени может не остаться.

**Синхронный неблокирующий ввод-вывод**, режим реализации сервера - один запрос на поток, то есть запрос соединения, отправленный клиентом, будет зарегистрирован на мультиплексоре, и мультиплексор запустит его только тогда, когда он опрашивает соединение с запросом ввода-вывода. Нить для обработки.

**преимущество**：

Этот метод ввода / вывода также имеет очевидные преимущества, то есть он не будет блокировать процесс ожидания данных в ядре, и каждый инициированный запрос ввода / вывода может немедленно вернуться без блокировки и ожидания. Он обычно используется, когда объем данных передается неравномерно, а время ожидания крайне случайное.

**недостаток**

Эта особенность опроса недавно выявила недостатки этой модели ввода / вывода. Опрос будет постоянно запрашивать ядро, которое будет занимать много процессорного времени, а использование системных ресурсов будет низким. В то же время модель не проста в использовании и требует написания сложного кода.

**Асинхронный неблокирующий ввод-вывод**, режим реализации сервера - это эффективный запрос для одного потока, и клиентский запрос ввода-вывода завершается ОС перед уведомлением серверного приложения о запуске потока для обработки.

Асинхронные, неблокирующие серверы, например, созданные в Node, используют только один поток для обслуживания всех запросов. Это означает, что экземпляр Node максимально использует один поток. Создатели спроектировали его с предпосылкой, что операции ввода-вывода и сети являются узким местом.

**преимущество**

Нет блокировки в течение всего процесса, действительно асинхронный

*NIO* является хорошим средством масштабирования приложений, активно использующих сетевые соединения, но данный подход имеет серьезный недостаток. Поток *NIO* вынужден явно опрашивать каналы (по-другому это называется "полинг"). При этом, если готовых к осуществлению взаимодействия каналов нет, то данный поток блокируется.

**Что такое блокировка?**

Когда приложение получает сетевые данные, если данные по сети передаются очень медленно, оно будет ждать, пока передача не будет завершена.

**Что такое неблокирование?**

Прикладная программа может напрямую получать готовые данные, не дожидаясь.

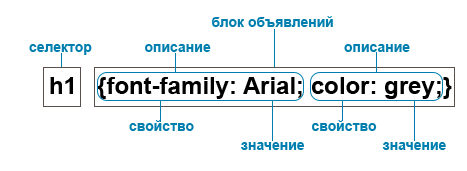
**При синхронизации**, Приложение будет напрямую участвовать в операциях чтения и записи ввода-вывода, и наше приложение будет напрямую блокировать определенный метод до тех пор, пока данные не будут готовы; или использовать циклическую стратегию для проверки состояния готовности данных в режиме реального времени и получения данных, если они готовы.

**Когда асинхронный**, Затем все операции чтения и записи ввода-вывода передаются операционной системе, которая не имеет прямого отношения к нашему приложению. Наша программа не обязательно должна быть связана с чтением и записью ввода-вывода. Когда операционная система завершает операции чтения и записи ввода-вывода, она отправит уведомление нашему приложению. Наше приложение может просто убрать столб данных.



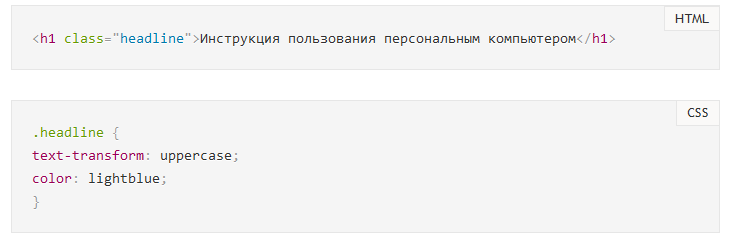
**2. CSS. Селекторы. Виды простых и сложных селекторов.**

Объявление стиля состоит из двух частей: **селектора** и **объявления**.

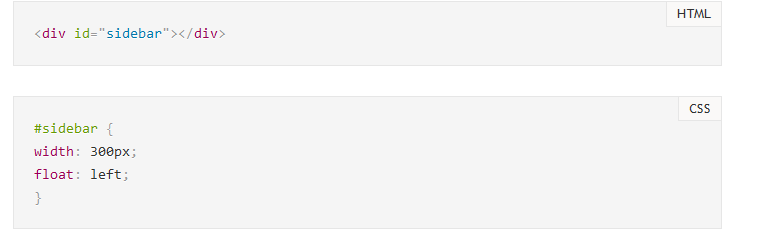


Виды селекторов:

* универсальный селектор, соответствует любому HTML-элементу, обозначается как \*;
* селектор элемента позволяет форматировать все элементы данного типа, например, h1;
* селектор класса позволяет задавать стили для всех элементов с заданным классом;



* селектор идентификатора, позволяет задавать стиль элементу с указанным id;



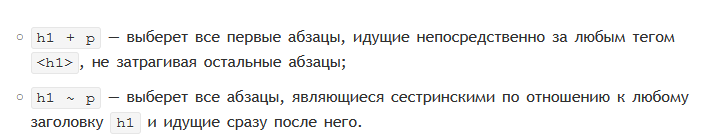
* селектор потомка, позволяет применить стиль к элементам, расположенным внутри контейнера;



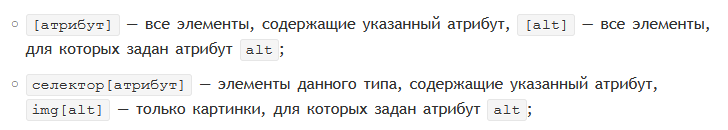
* дочерний селектор, позволяет применить стили к дочерним элементам контейнера;



* сестринский селектор, позволяет применять стили к элементам одного уровня;



* селектор атрибута, позволяет задавать стили элементу на основе атрибута;

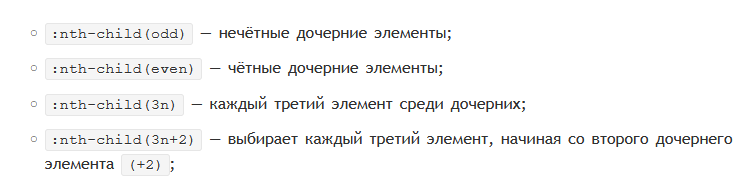


(пример)

* селектор псевдокласса. Псевдоклассы - это классы, фактически не прикрепленные к HTML-тегам. Они позволяют применить CSS-правила к элементам при совершении события или подчиняющимся определенному правилу.

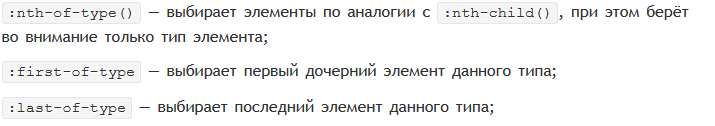
Примеры: :hover, :focus

* селектор структурных псевдоклассов. Структурные псевдоклассы отбирают дочерние элементы в соответствии с параметром, указанным в круглых скобках:



(пример)

* селектор структурных псевдоклассов типа. Указывают на конкретный тип дочернего класса.



(пример)

* селектор псевдоэлемента. Используются для задания стилей содержимому, созданного с помощью свойства content. Самые популярные: :before, :after.

# -== IU7-Web Bilet #4 ==-

**1. Основные идеи и принципы современного CI/CD. Инструменты. Докеризация.**

CI/CD (Continuous Integration/Continuous Delivery) - непрерывная интеграция и непрерывная поставка - набор принципов и практик DevOps, которые позволяют автоматизировать сборку, упаковку, тестирование и развертывание изменения ПО.

Принципы (скорее цели) CI/CD:

* сегрегация ответственности заинтересованных сторон - своевременное участие различных заинтересованных сторон:
  + Разрабы создают опыт и логику продукта - ответственность за функции
  + Девопсы - ответсвенность за доступность продукта (логистика кода)
  + Пользователи - ответственность за использование продукта (продукт должен оправдывать усилия)
* снижение риска;
* короткий цикл обратной связи.

Непрерывная интеграция (CI) - в код вносятся небольшие изменения с частыми коммитами, вносимые изменения автоматизировано тестируются (непрерывное тестирование). Цель - наладить процесс непрерывной интеграции и повысить надежность.

Непрерывная поставка (CD) - “Начинается там, где заканчивается непрерывная интеграция” - автоматизирует развертывание приложений в различные окружения. Как? Позволяют настроить переменные и параметры окружения, конфиги, сикреты, коннекшнстринги и тд, и выполняются все необходимые запросы к внешним сервисам и базам (перезапуск, например, или выполнение каких-то действий, необходимых для развертывание).

Принципы и идеи CI/CD:

* Частые коммиты в системе контроля версий + ветвление
* автоматизация сборки (единый уникальный инструмент для сборки продукта);
* автоматизация установки (централизованное управление);
* сборка после каждого коммита (минус такого подхода - затраты по времени);
* малое время сборки и тестирования;
* production-like среда (простой выход - докер-контейнер, внутри которого можно инкапсулировать);
* доступные и понятные отчёты;
* централизованное хранилище билдов.

Практики и инструменты CI/CD (примеры):

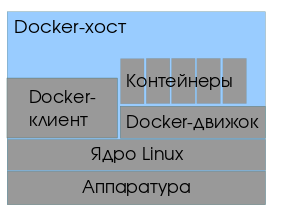
* Самое простое - модель ветвления в гите (Gitflow/Gitlabflow) - например, ветки разработки, тестирования, продакшна, фича-ветки, релиз-ветки, фикс-ветки и тд. Инструменты в представлении не нуждаются..
* Фича-флаги (feature flag) - включение и выключение фич в рантайме: фича на стадии разработки оборачивается фича-флагом и развертывается из мастер-ветки вместе со всем остальным в продакшн, но “выключается” фича-флагом до тех пор, пока не будет готов. Инструменты: LaunchDarkly, Optimizey Rollouts и тд
* Автосборка и автотесты после каждого коммита - gitlab.ci.yml в Гите

Докеризация

Что это? ПО с открытым кодом, предоставляющее легковесную среду для развертывания приложений. Принцип работы схож с виртуализацией, но докер значительно более легковесный - образ контейнера имеет несколько слоев, в том числе и других образов, поэтому для пересборки контейнера неизмененные слои можно переиспользовать (кэшируются). Запущенные докер контейнеры - это процессы, совместно использующие ядро ОС. Докеризацию выполняет докер демон.

Виртуальная машина: содержат всю операционную систему, приложение и необходимые библиотеки

Докер: использует ядро ОС совместно с другим и контейнерами (если гостевая ос отличается от родной, то в образе есть слой, который транслирует вызовы гостевой ос на вызовы родной), приложение и все необходимые зависимости -- поэтому докер контейнеры значительно более легковесные



Недостатки виртуализации:

* Медленная загрузка
* Возможно, платно
* Не все виртуальные машины поддерживают совместное использование
* Большие и тяжелые образы

Преимущества докеризации над виртуализацией:

* Ускоряют процесс разработки
* Удобная инкапсуляций приложений
* Понятный мониторинг
* Удобное масштабирование

Кроме того, докер удобно для деплоя в распределенных системах

**2. Технологии толстого клиента. Организация дуплексной связи клиент-сервер (Polling, Long polling, WebSocket).**

Толстый клиент - обеспечивает полное функционирование независимо от сервера. Часто сервер в этом случае выступает только в роли хранилища данных, а вся обработка выполняется на стороне клиента.

(+) Достоинства:

* Режим многопользовательской работы
* Работает при обрывах связи с сервером
* Высокое быстродействие (но зависит от аппаратных средств клиента)

(-) Недостатки:

* Большой размер дистрибутива
* Мб платформозависимый
* Сложный процесс установки, настройки и обновления
* Мб неактуальность данных из-за сложности обновления

Дуплексная связь клиент-сервер: шаблон обмена сообщениями, в котором обе конечные точки могут отправлять друг другу сообщения.

Подходы к реализации дуплексной связи клиент-сервер:

Polling - клиент раз в несколько секунд опрашивает сервер на наличие событий

(+) Плюсы:

* Простота реализации

(-) Минусы:

* Много запросов серверу -- даже если событий нет, клиент все равно отправляет запросы
* События приходят с опозданием (потому что есть период опроса)
* Серверу приходится хранить события, пока клиент их не заберет

Long Polling - Клиент отправляет запрос на сервер, сервер держит открытым соединение пока не придут какие-нибудь данные или клиент не отключится самостоятельно. Как только данные пришли — отправляется ответ и соединение закрывается и открывается следующее и так далее.

(+) Плюсы:

* Значительно уменьшается количество запросов
* Более высокая временная точность событий
* Сервер хранит события только на время реконнекта

(-) Минусы:

* Более сложная схема
* Долго держим соединения, + их много

WebSockets - бинарный дуплексный протокол, в котором клиент и сервер могут взаимодействовать на равных (инициировать соединение, отправлять запросы и ответы). Применяются для чатов и игр - где нужна высокая временная точность событий.

(+) Плюсы:

* Поднимаем одно соединение
* Очень высокая временная точность событий
* Браузер поддерживает вебсокеты и контроллирует обработку сетевых сбоев

(-) Минусы:

* Протокол не совместим с HTTP - нужен отельный сервер для обработки http запросов

# 

# -== IU7-Web Bilet #5 ==-

**1. Стек TCP/IP. Назначение, структура, применение. Особенности протоколов.**

TCP/IP — сетевая модель передачи данных, представленных в цифровом виде. Модель описывает способ передачи данных от источника информации к получателю. В модели предполагается прохождение информации через четыре уровня, каждый из которых описывается правилом (протоколом передачи). Наборы правил, решающих задачу по передаче данных, составляют стек протоколов передачи данных, на которых базируется Интернет.

Стек протоколов TCP/IP включает в себя четыре уровня:

Прикладной уровень (Application Layer),

Транспортный уровень (Transport Layer),

Межсетевой уровень (Сетевой уровень) Internet Layer),

Канальный уровень (Network Access Layer).

Протоколы этих уровней полностью реализуют функциональные возможности модели OSI (это аббревиатура от английского **Open** **System** **Interconnection**, то есть **модель** взаимодействия открытых систем). На стеке протоколов TCP/IP построено всё взаимодействие пользователей в IPсетях. Стек является независимым от физической среды передачи данных, благодаря чему, в частности, обеспечивается полностью прозрачное взаимодействие между проводными и беспроводными сетями.

**Прикладной уровень**

На прикладном уровне (Application layer) работает большинство сетевых приложений. Эти программы имеют свои собственные протоколы обмена информацией, например, интернет браузер для протокола HTTP, ftp-клиент для протокола FTP (передача файлов), почтовая программа для протокола SMTP (электронная почта), SSH (безопасное соединение с удалённой машиной), DNS (преобразование символьных имён в IPадреса) и многие другие.

**Транспортный уровень**

Протоколы транспортного уровня Transport layer) могут решать проблему негарантированной доставки сообщений, а также гарантировать правильную последовательность прихода данных. В стеке TCP/IP транспортные протоколы определяют, для какого именно приложения предназначены эти данные. TCP (IP идентификатор 6) «гарантированный» транспортный механизм с предварительным установлением соединения, предоставляющий приложению надёжный поток данных, дающий уверенность в безошибочности получаемых данных, перезапрашивающий данные в случае потери и устраняющий дублирование данных. TCP позволяет регулировать нагрузку на сеть, а также уменьшать время ожидания данных при передаче на большие расстояния. Более того, TCP гарантирует, что полученные данные были отправлены точно в такой же последовательности. В этом его главное отличие от UDP.

UDP (IP идентификатор 17) протокол передачи датаграмм без установления соединения. Также его называют протоколом «ненадёжной» передачи, в смысле невозможности удостовериться в доставке сообщения адресату, а также возможного перемешивания пакетов.

В приложениях, требующих гарантированной передачи данных, используется протокол TCP. UDP обычно используется в таких приложениях, как потоковое видео и компьютерные игры, где допускается потеря пакетов, а повторный запрос затруднён или не оправдан, либо в приложениях вида запрос-ответ, где создание соединения занимает больше ресурсов, чем повторная отправка.

**Сетевой (межсетевой) уровень**

Межсетевой уровень Internet layer) изначально разработан для передачи данных из одной сети в другую. На этом уровне работают маршрутизаторы, которые перенаправляют пакеты в нужную сеть путём расчёта адреса сети по маске сети. Примерами такого протокола является X.25 и IPC в сети ARPANET. С развитием в уровень были внесены дополнительные возможности по передаче из любой сети в любую сеть, независимо от протоколов нижнего уровня, а также возможность запрашивать данные от удалённой стороны, например в протоколе ICMP (используется для передачи диагностической информации IP-соединения) и IGMP (используется для управления multicastпотоками). ICMP и IGMP расположены над IP и должны попасть на следующий — транспортный — уровень, но функционально являются протоколами сетевого уровня, и поэтому их невозможно вписать в модель OSI. Пакеты сетевого протокола IP могут содержать код, указывающий, какой именно протокол следующего уровня нужно использовать, чтобы извлечь данные из пакета. Это число — уникальный IPномер протокола. ICMP и IGMP имеют номера, соответственно, 1 и 2. К этому уровню относятся: DVMRP, ICMP, IGMP, MARS, PIM, RIP, RIP2, RSVP

**Канальный уровень**

Канальный уровень (англ. Link layer) описывает способ кодирования данных для передачи пакета данных на физическом уровне (то есть специальные последовательности бит, определяющих начало и конец пакета данных, а также обеспечивающие помехоустойчивость). Ethernet, например, в полях заголовка пакета содержит указание того, какой машине или машинам в сети предназначен этот пакет.

Примеры протоколов канального уровня — Ethernet, IEEE 802.11 WLAN, SLIP, Token Ring, ATM и MPLS.

PPP не совсем вписывается в такое определение, поэтому обычно описывается в виде пары протоколов HDLC/SDLC.

MPLS занимает промежуточное положение между канальным и сетевым уровнем и, строго говоря, его нельзя отнести ни к одному из них.

Канальный уровень иногда разделяют на 2 подуровня — LLC и MAC. Кроме того, канальный уровень описывает среду передачи данных, физические характеристики такой среды и принцип передачи данных (разделение каналов, модуляцию, амплитуду сигналов, частоту сигналов, способ синхронизации передачи, время ожидания ответа и максимальное расстояние). При проектировании стека протоколов на канальном уровне рассматривают помехоустойчивое кодирование — позволяющие обнаруживать и исправлять ошибки в данных вследствие воздействия шумов и помех на канал связи.

**2. Протокол HTTP 1.1. Расшифровка, назначение, структура. Формат пакета.**

HTTP - Hyper Text Transfer Protocol

2 особенности протокола:

текстовый (вся коммуникация между данными - символьный формат) протокол;

без сохранения состояния (отсутствует понятие состояния (только “костыль - cookie”: появились проблемы при появлении веб-приложений (например, чтобы не переавторизовываться в соц.сети))).

**Назначение:**Протокол HTTP предназначен для передачи содержимого в Интернете.

HTTPсессия:

* клиент устанавливает TCPсоединение с сервером (обычно на 80 порт);
* сервер принимает запрос на соединение и ожидает текст HTTPзапроса;
* клиент отправляет запрос со всей необходимой информацией URI, тип запроса, заголовки, тело запроса);
* сервер обрабатывает запрос и отдаёт ответ (код статуса, заголовки ответа и тело ответа).

HTTPструктура:

**Структура запроса:**

* строка запроса;
* заголовки;
* пустая строка;
* тело запроса.

**Структура ответа:**

* строка статуса ответа;
* заголовки ответа;
* пустая строка;
* тело ответа.

**HTTPметоды:**

* OPTIONS - запрос методов сервера;
* GET - запрос документа;
* HEAD - аналог GET, но без тела ответа (метаданные без данных);
* POST - передача данных от клиента;
* PUT - размещение файла по URI/изменение данных;
* DELETE - удаление файла по URI/удаление данных;
* TRACE, LINK, UNLINK, CONNECT - используются редко;
* PATCH - частичное изменение данных на сервере.

**Коды ответа:**

* 1xx - информационные;
* 2xx - успешное выполнение;
* 3xx - перенаправление;
* 4xx - ошибка клиента;
* 5xx - ошибка сервера.

3. Задача на: Nginx/REST/Архитектуру/Дизайн/Верстку/JS. В основном - на клиентский JS.

# -== IU7-Web Bilet #6==-

1. CSS3. Фильтры, трансформации, анимации, переходы. Интерактивные возможности.
2. Язык Javascript. История. Основные идеи. Область применения. Структура программы.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Вопрос 1.**

Спецификация **CSS3 UI** описывает свойства и значения, связанные с пользовательским интерфейсом.

**Свойства CSS3 для стилизации элементов взаимодействия с пользователем**:

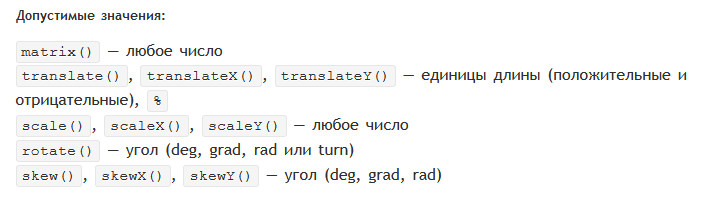
1. Стилизация внешнего контура: outline (краткая запись), outline-width, outline-style, outline-color, outline-offset.
2. Изменение размера блоков: resize, позволяет указать, может ли элемент изменять размеры, и, если да, то вдоль какой оси. Значения свойства: horizontal, vertical и т. д.
3. Стилизация курсора: cursor.
4. Цвет каретки вставки: свойство caret-color.

(Это скорее всего не понадобится, так как в вопросе хотят фильтры, трансформации и т. д.)

**CSS3-фильтры** воспроизводят в браузере визуальные эффекты, похожие на фильтры Photoshop. Их можно применять к изображениям и любым непустым элементам. Примеры возможных значений для свойства filter: blur(), sepia() и т. д. В скобках указывается процент применения свойства. (Прямо как мы в Instagram накладываем фильтры с определенным процентом, тут то же самое.)

**CSS3-трансформации** позволяют сдвигать, поворачивать и масштабировать элементы. Трансформации преобразовывают элемент, не затрагивая остальные элементы веб-страницы, т.е. другие элементы не сдвигаются относительно него. Трансформированным считается элемент с любым установленным значением свойства transform, отличным от none.

Пример для чайников, чтобы было понятнее: указывается css-свойство transform: translateX(5), и элемент смещается на 5 пикселей вправо.



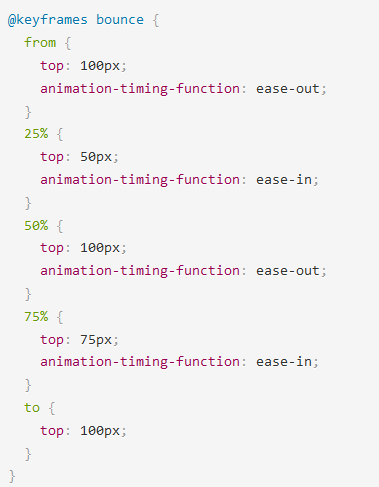
Подробнее про transform: matrix(a, c, b, d, tx, ty). По сути позволяет объединить несколько трансформаций в одну (слева направо): масштаб по горизонтали, сдвиг сторон по оси Y, сдвиг сторон по оси X, масштаб по вертикали, смещение по оси X, смещение по оси Y.

Остальные: translate - сдивг по осиям, scale - масштабирование, rotate - повороот, skew - деформирование сторон элемента.

**CSS3-анимация** используется для анимации элементов (что очевидно). Создание CSS3-анимации состоит из следующих этапов:

1. Описание анимации с помощью *@keyframes имя\_анимации {список правил}*.

В примере ниже мы говорим, что создаем анимацию с именем bounce, и описываем поведение элемента (стили) в начале анимации (from), на промежуточных этапах (25%, 50%, 75%) и в конце (to).



1. Применение анимации к элементу. Либо с помощью свойства animation (включает в себя полную настройку анимации) либо с помощью отдельных, например: animation-name, animation-duration и т. Д.

Например, применить анимацию, описанную выше, к элементу можно так: animation: bounce 1s infinite ease-in-out;, то есть анимация под названием bounce будет длиться 1 секунду, проигрываться бесконечно, колебаний по скорости не будет.

**CSS3-переходы** позволяют анимировать исходное значение CSS-свойства на новое значение с течением времени. Для задания всех свойств перехода обычно используют краткую запись свойства transition.

А теперь по человечески: когда у нас меняется состояние элемента, должны (с точки зрения пользователя) меняться его стили. Например, кнопка синего цвета, но при наведении на неё курсора она становится красной. Момент наведения курсора - момент изменения состояния кнопки. Описание на CSS того, как изменится кнопка при наведении курсора это и есть CSS-переход. В CSS для задания стилей элементу, на который наведен курсор, используется псевдокласс :hover.

button {background-color: blue, transition: background-color 2s}

button: hover {background-color: red}

Мы указали, что background-color - изменяемое свойство, и переход будет с течением 2 секунд.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Вопрос 2.**

Язык появился в 1995 году. Когда JavaScript создавался, у него было другое имя – «LiveScript». Однако, язык Java был очень популярен в то время, и было решено, что позиционирование JavaScript как «младшего брата» Java будет полезно. Со временем JavaScript стал полностью независимым языком со своей собственной спецификацией, называющейся ECMAScript, и сейчас не имеет никакого отношения к Java.

JavaScript изначально создавался только для браузера, но сегодня JavaScript может выполняться не только в браузере, но и на сервере или на любом другом устройстве, которое имеет специальную программу, называющуюся “движком” JavaScript.

Современный JavaScript – это «безопасный» язык программирования. Он не предоставляет низкоуровневый доступ к памяти или процессору, потому что изначально был создан для браузеров, не требующих этого. Этот ЯП занимает уникальную позицию в качестве самого распространённого языка для браузера, обладающего полной интеграцией с HTML/CSS.

Не совсем понятна формулировка вопроса аля “структура кода”. По сути тут могут спросить всё что угодно, поэтому опишу общие черты.

JS состоит из инструкций, которые разделяются точкой с запятой (но в большинстве случаев этот разделитель можно не писать). Комментарии задаются с помощью // или /\*... \*/.

На протяжении долгого времени JavaScript развивался без проблем с обратной совместимостью. Новые функции добавлялись в язык, в то время как старая функциональность не менялась. Так было до 2009 года, когда появился ECMAScript 5 (ES5). Он добавил новые возможности в язык и изменил некоторые из существующих. Чтобы устаревший код работал, как и раньше, по умолчанию подобные изменения не применяются. Поэтому нам нужно явно их активировать с помощью специальной директивы: "use strict".

Управление памятью в JavaScript выполняется автоматически и незаметно с помощью спец. алгоритмов.

# -== IU7-Web Bilet #7 ==-

**1. Основные идеи и возможности HTML5. Область применения. Ограничения.**

***Основные идеи***

**HTML5** (*HyperText Markup Language, version 5*) — язык для структурирования и представления содержимого всемирной паутины, инструмент для упорядочивания Web-контента.

Цель разработки HTML5 — улучшение уровня поддержки мультимедиа-технологий с одновременным сохранением обратной совместимости, удобочитаемости кода для человека и простоты анализа для парсеров.

Веб-страницы на практике оказывались свёрстаны с использованием смеси особенностей, представленных различными спецификациями, включая спецификации программных продуктов, например веб-браузеров, а также сложившихся общеупотребительных приёмов. HTML5 был создан как единый язык разметки, который мог бы сочетать синтаксические нормы HTML и XHTML. Он расширяет, улучшает и рационализирует разметку документов, а также добавляет единый API для сложных веб-приложений.

***Возможности***

Язык HTML5 выполняет разделение и маркирование информации для упрощения ее использования и понимания. Именно в этом состоит огромная семантическая ценность HTML5. Он предоставляет дизайнерам и разработчикам всех уровней возможности для предоставления в публичный доступ буквально любого контента.

HTML5 предоставляет эффективные инструменты для управления данными, для рисования, для воспроизведения видео- и аудиоконтента. HTML5 облегчает разработку кросс-браузерных Web-приложений, а также приложений для мобильных устройств. Кроме того, HTML5 способствует повышению гибкости – благодаря возможности создания интерактивных Web-сайтов.

***Новые особенности***

В HTML5 реализовано множество новых синтаксических особенностей.

*Теги*

Новые теги HTML5 обладают «говорящими» названиями, которые раскрывают назначение и характер использования этих элементов. Например, широко применявшийся до настоящего времени тег <div> был дополнен тегами <section> и <article>. Кроме того, были добавлены теги <video>, <audio>,<canvas> и <figure>, которые обеспечивают более точное описание определенных типов контента.

*Синтаксис и написание*

* Синтаксис больше не базируется на SGML, несмотря на подобие его разметки.
* Применяется новая вводная строка, которая выглядит как объявление типа документа в SGML, <!DOCTYPE html>, запускающая соответствующий стандартам режим визуализации.
* Новые компоненты ввода: date/time, email, url, search, number, range, tel, color.
* Новые атрибуты: charset (в <meta>), async (в script).
* Глобальные атрибуты, которые могут быть применены ко всем элементам: id, tabindex, hidden, data-\* (пользовательские атрибуты данных).

*Другое*

* Постоянная обработка ошибок (улучшенные алгоритмы парсинга)
* Расширенная поддержка функций веб-приложений (расширенный контроль над производительностью своих сайтов)
* Создание мобильных сайтов стало проще (код HTML5 теперь лучше поддерживается маломощными электронными устройствами, такими как смартфоны и планшеты)
* Элемент canvas (внедрение этого уникального тега произвело огромное влияние на Adobe Flash. Используя элемент canvas разработчики могут рисовать с помощью скриптов графические изображения с применением различных цветов.)
* Элемент menu (может быть использован для обеспечения более лучшей интерактивности на странице.
* Настраиваемые атрибуты данных (основной целью его введения было хранение дополнительной информации о разных элементах. Могут быть включены пользовательские данные, что дает разработчикам больше возможностей сделать привлекательные и эффективные страницы, без лишних запросов на сервер или вызовов Ajax.)
* (Возможное) прощание с Cookies (поддержка локального хранилища. Раньше Cookie. Объект localStorage является частью глобального пространства имен и при использовании скриптов может быть доступен из любого места.)

*Также*

* Средства Web Worker для исполнения фоновых процессов.
* Интерфейс WebSocket для установки постоянного соединения между резидентным приложением и сервером.
* Улучшенное извлечение хранящихся данных.
* Повышенная скорость сохранения и загрузки страниц
* Поддержка CSS3 при управлении пользовательским интерфейсом, что обеспечивает контентную ориентированность HTML5.
* Улучшенная обработка форм в браузере.
* Спецификация APIинтерфейса Geolocation, использующая геолокационные возможности смартфонов в интересах задействования облачных сервисов и приложений для мобильных устройств.
* Усовершенствованные формы, ослабляющие потребность в загрузке кода JavaScript, что обеспечивает более эффективную связь между мобильными устройствами и серверами cloud-среды.

***Область применения***

C добавлением новых функций HTML5 стал не просто новой версией языка разметки для создания веб-страниц, но и фактически платформой для создания приложений, а область его использования вышла далеко за пределы веб-среды интернет: HTML5 применяется также для создания мобильных приложений под Android, iOS, Windows Mobile и даже для создания десктопных приложений для обычных компьютеров.

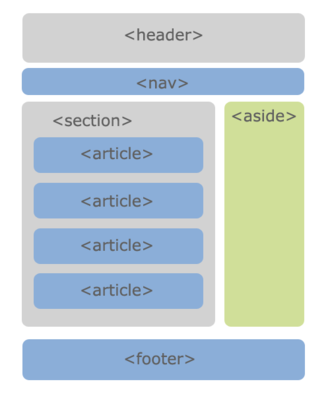
Применяется преимущественно в двух значениях:

1. Как обновленный язык разметки гипертекста (развитие предыдущей версии HTML4)
2. Как мощная платформа для создания веб-приложений, которая включает не только непосредственно язык разметки гипертекста, обновленный HTML, но и язык программирования JavaScript и каскадные таблицы стилей CSS 3.

***Ограничения***

1. Сайты, созданные с помощью обновлённого языка разметки, будут генерировать файлы большого размера, на длительный промежуток времени и с большим количеством секретных данных, что отрицательно скажется на уровне защищённости таких сайтов от злоумышленников.
2. Далеко не все веб-браузеры поддерживают новую технологию.
3. Создавать игры на HTML 5 – невыгодно, ввиду малой востребованности таких игр среди пользователей.

Пример веб-страницы.



**2. MVC-фреймворки. Понятие фреймворка. Примеры. Типовая структура MVC-фреймворка.**

**Веб-фреймворки** - это наборы функций, объектов, правил и других конструкций кода, предназначенных для решения общих проблем, ускорения разработки и упрощения различных типов задач, стоящих в конкретной области.

AngularJS является JavaScript-фреймворком с открытым исходным кодом, разрабатываемым Google. Предназначен для разработки одностраничных приложений…. Backbone.js придает структуру веб-приложениям с помощью моделей с биндингами по ключу и пользовательскими событиями, коллекций с богатым набором…

EmberJS Главной особенностью EmberJS является привязка данных. Вы просто создаете переменную, и, когда значение этой переменной изменяется, обновляется любая…

Knockout.js — это популярная JavaScript библиотека, позволяющая реализовать Model-View-View Model MVVM паттерн на клиенте.

MVC расшифровывается как модель-представление-контроллер (от англ. modelview-controller). Это способ организации кода, который предполагает выделение блоков, отвечающих за решение разных задач. Один блок отвечает за данные приложения, другой отвечает за внешний вид, а третий контролирует работу приложения.

**Компоненты MVC**

**Модель** — этот компонент отвечает за данные, а также определяет структуру приложения. Например, если вы создаете To-Do приложение, код компонента model будет определять список задач и отдельные задачи.

**Представление** — этот компонент отвечает за взаимодействие с пользователем. То есть код компонента view определяет внешний вид приложения и способы его использования.

**Контроллер** — этот компонент отвечает за связь между model и view. Код компонента controller определяет, как сайт реагирует на действия пользователя. По сути, это мозг MVCприложения. Контроллер решает задачи, он не должен включать в себя бизнес-логику приложения. Не стоит заниматься существенным в контроллере, иначе возникает FSUC Fat Stupid Ugly Controllers).

**MVC** — подход к проектированию приложения, который предполагает выделение кода в блоки модель, представление и контроллер. Контроллер обрабатывает входящие запросы. Модель достаёт из базы данных информацию, нужную для выполнения конкретных запросов. Представление определяет результат запроса, который получает пользователь.)

ASP.NET MVC Framework — фреймворк для создания веб-приложений, который реализует шаблон Model-view-controller.



3. Задача на: Nginx/REST/Архитектуру/Дизайн/Верстку/JS. В основном - на клиентский JS.

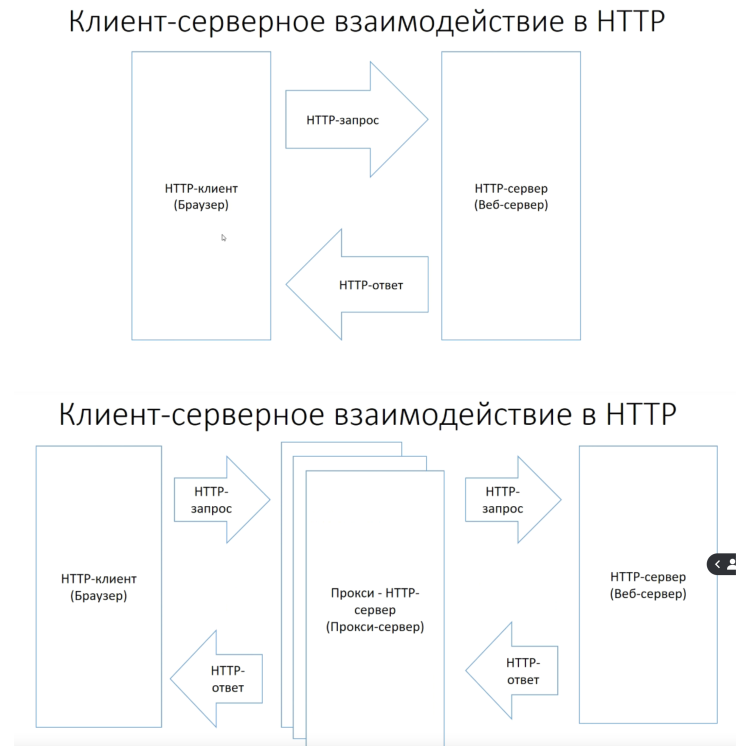
# -== IU7-Web Bilet #8 ==-

**1. Варианты организации клиент-серверной архитектуры приложения в веб. MPA, MPA+AJAX, SPA. Расшифровка, определение, принципы.**

Клиент-сервер - сетевая архитектура, в которой сетевая нагрузка распределена между поставщиками услуг, называемыми серверами и заказчиками услуг, называемыми клиентами.

*Классический "клиент-сервер" в WEB:*

* Взаимодействие по протоколу HTTP/протоколам поверх HTTP
* HTTP работает поверх TCP
* Абстрагирование от физической реализации сети
* Адресация - сокетах (ip-фдрес + порт)
* Доменные имена преобразуются в ip-адрес
* Клиент - всегда инициатор сессии

****

**MPA(Multi Page Application)** — это многостраничные приложения, работающие, как привычные нам веб-сайты. Они отправляют запрос на сервер и полностью обновляют страницу, когда с ней совершается какое-либо действие (переход на другую страницу, внесение и изменение данных). Подобная архитектура приложения значительно влияет на скорость и производительность, поскольку большая часть данных подгружается повторно при каждом переходе. В целом, MPA представляет собой главную страницу и множество ссылок на другие страницы.

Этим принципом пользуются e-commerce сайты вроде Ebay, Ozon и Amazon, где необходимо отображать большое количество контента.

У

### **Преимущества MPA**

* Простая SEO-оптимизация. MPA часто используют сайты, для которых важно попадать в топы поисковых систем. Каждая из страниц имеет уникальный URL и стабильна, что позволяет поисковым ботам адекватно ее просканировать.
* Масштабируемость. В MPA приложение можно вложить столько информации, сколько потребуется, без ограничений по страницам и функциям.
* Проверенная классика. MPA работают по тем же принципам, что и знакомые пользователю веб-сайты с классической навигацией.

### **Недостатки MPA**

* Низкая скорость разработки. MPA приложения требуют использования фреймворков на обеих сторонах: клиента и сервера. Также из-за тесной взаимосвязи frontend и backend усложняется работа разработчиков. Всё это в совокупности увеличивает бюджет и сроки разработки.
* Затраты на создание мобильного приложения. MPA приложения плохо конвертируются в мобильные. Для этого в значительной части случаев потребуется разработка backend с нуля.

**MPA + AJAX (Multi Page Applications + Asynchronous Javascript and XML) -** ajax

позволяет обновлять только часть страницы, а не всю страницу целиком.

AJAX базируется на двух основных принципах:

* использование технологии динамического обращения к серверу «на лету», без перезагрузки всей страницы полностью;
* использование DHTML(Dynamic HTML) для динамического изменения содержания страницы.

При использовании AJAX:

1. Пользователь заходит на веб-страницу и нажимает на какой-нибудь её элемент;
2. Скрипт определяет, какая информация необходима для обновления страницы;
3. Браузер отправляет соответствующий запрос на сервер;
4. Сервер возвращает только ту часть документа, на которую пришёл запрос;
5. Скрипт вносит изменения с учётом полученной информации (без полной перезагрузки страницы).

Пример такого сочетания - лайки ВК. Сам ВК - MPA, но внутри страниц JS. Обработчик перехватывает лайк, происходит асинхронный запрос на сервер, обработка и возврат.

В таком варианте полностью ломается логика архитектуры, “как она размазана, никто не знает”, полностью теряется контроль над ситуацией в таком приложении.

**SPA(Single page application)** — это одностраничное приложение, содержащее HTML-страницу, которая динамически (без полной перезагрузки) обновляется в ответ на действия пользователя. Архитектура приложения устроена так, что при первоначальном запуске посетитель видит основной контент сайта в браузере, а новые данные загружаются на ходу по мере необходимости, например, при прокрутке или клике на иконку.

По сути одностраничные приложения во многом похожи на десктопные: при переходе на новую страницу обновляется только часть контента, позволяя не загружать одни и те же элементы сайта множество раз. Подобный эффект для SPA обеспечивают современные JavaScript фреймворки. Для небольших проектов подойдёт jQuery, однако для создания серьёзного SPA потребуются такие frontend-библиотеки, как Angular.

Пример SPA - gmail.

SPA = MVC + AJAX/JSON + REST.

**Преимущества SPA**

Исследования показывают, что пользователи чувствительны к скорости загрузки страницы. В вопросе скорости подгрузки у SPA сайтов нет равных, что и является их главным преимуществом. SPA быстро обновляет необходимую часть данных, значительно экономя время.

Однако высокая скорость — не единственный плюс SPA приложения. Помимо этого одностраничные приложения имеют следующие преимущества:

* лёгкость создания. Для разработки SPA приложения уже готовы библиотеки и фреймворки, работа над frontend и backend может вестись параллельно. Кроме того, на основе готового кода в дальнейшем можно построить мобильное приложение;
* гибкость пользовательского интерфейса. Для одной страницы проще разработать увлекательный и интерактивный дизайн интерфейса;
* простое кэширование данных. Важные данные загружаются в один запрос, а далее пользователь одностраничного приложения может работать частично оффлайн, подключаясь к интернету, чтобы сохранить прогресс.

**Недостатки SPA**

Некоторые пользователи отключают JavaScript в браузерах, чтобы ускорить загрузку сайта и избавиться от нежелательной рекламы. У таких пользователей приложение просто не заработает. Кроме того, существует проблема утечки памяти JavaScript, которая уменьшает скорость загрузки и открывает ваши данные для действий злоумышленников.

Кроме того, одностраничные приложения:

* плохо поддаются SEO оптимизации. URL страниц практически не меняется, а данные подгружаются динамически, когда для оптимизации важна устойчивость и уникальные URL для каждой страницы. Поисковым ботам сложно просканировать SPA приложение, и на данный момент грамотно индексировать такие сайты умеет только Google;
* сильно нагружают браузер. Такая проблема возникает из-за относительно “тяжёлых” клиентских фреймворков;
* могут недёшево стоить. Разумеется, итоговая стоимость разработки разнится, но цена сложных сайтов довольно высока.

**2. Структура типового MVC web-приложения на примере одного из фреймворков (Express, ASP, Django, Spring, Rails, etc.).**

**Django:**

Вместе эти три части — логика доступа к данным, логика исполнения и логика представления данных — составляют концепт, который обычно называют концепцией Model-View-Controller MVC (модель-представление-контроллер). В этой схеме «Модель» относится к структуре данным, «Представление» — относится к части, которая отвечает за выборку того, что и как отображать, а «Контроллер» — относится к части, которая определяет какое представление использовать, например — в зависимости от ввода данных пользователя.

Django следует концепции MVC достаточно строго, чтобы его можно было называть “MVC фреймворк”. Вот краткое описание того, как M, V и C разделены в среде Django:

M — часть доступа к данным, обслуживается системой баз данных Django.

V — часть, которая занимается выборкой того, что именно и как именно отображать, в Django этим занимаются представления (views) и шаблоны (templates).

C — часть, которая определяет представление в зависимости от указаний пользователя, этим занимается сам фреймворк, следуя настройкам URLconf и вызывая соответствующую функцию Python для заданного URLа. Так как «C» занимается сам фреймворк и больше всего действий происходит в моделях, шаблонах и представлениях, Django может называться «MVC фреймворком«.

В интерпретации Django, представление (view) описывает данные, которые должны быть представлены пользователю; т.е. не столько как они будут переданы, сколько какие именно данные передать. В Ruby on Rails, и других подобных фреймворках, предполагается что роль контроллера включает в себя выбор какие данные передать пользователю, тогда как представление определяет — как именно эти данные будут выглядеть, а не какие данные передавать.

MVC в django = MTV + встроенное C

В концепции **MTV**:

* *M* означает “*Model*” – модель, уровень доступа к данным. Сюда относится всё, что связано с данными – как получить к ним доступ, как проверить их, какое у них поведение и отношение друг с другом.
* *T* означает “*Template*” – шаблоны. Этот уровень содержит в себе всё, что связано с отображением: как именно что-то должно быть отображено на веб-странице или любом другом документе.
* *V* означает “*View*” – представление. Этот уровень связан со всей логикой, связывающей модель и необходимые шаблоны. Вы можете себе представлять это уровень как мост между *моделями* и *шаблонами*.

*В основном все, дальше дополнения*

ASP и Visaul Studio:

Когда вы впервые создаете новый проект MVC Framework, у вас есть на выбор две базовых отправных точки: шаблон Empty (Пустой) и шаблон MVC. Реальное отличие связано с дополнительным содержимым, которое шаблон MVC добавляет в новые проекты. Оно предоставляет готовую отправную точку, включающую стандартные контроллеры и представления, конфигурацию безопасности, ряд популярных пакетов JavaScript и CSS (таких как jQuery и Bootstrap), а также компоновку - которая применяет библиотеку Bootstrap для построения темы, оформляющей пользовательский интерфейс приложения.

Вариант Empty проекта содержит только базовые ссылки, требуемые для MVC Framework, и базовую структуру папок. В проекте MVC применяются два вида соглашений.

* Соглашения первого вида - это просто предположения о том, как может выглядеть структура проекта. Например, общепринято размещать файлы JavaScript в папке Scripts. Здесь их рассчитывают обнаружить другие разработчики, использующие MVC, и сюда будут устанавливаться пакеты NuGet.
* Соглашения второго вида произрастают из принципа соглашения по конфигурации (convention over configuration) или соглашения над конфигурацией. Соглашение по конфигурации означает, что вы не должны явно конфигурировать, к примеру, ассоциации между контроллерами и их представлениями. Нужно просто следовать определенному соглашению об именовании для файлов - и все будет работать. При соглашении такого рода снижается гибкость в изменении структуры проекта.

Классы контроллеров должны иметь имена, заканчивающиеся на Controller, например, ProductController, ShopController. При ссылке на контроллер где-либо в проекте указывается первая часть имени (такая как Product), а инфраструктура MVC Framework автоматически добавляет к этому имени слово Controller и начинает поиск класса контроллера.

Представления и частичные представления располагаются в папке: /Views/ИмяКонтроллера

3. Задача на: Nginx/REST/Архитектуру/Дизайн/Верстку/JS. В основном - на клиентский JS.

# -== IU7-Web Bilet #9 ==-

1. JS. Область определения. Объявление переменных (в стиле ES5 и ES2015). Замыкание. Примеры.

2. Унифицированный идентификатор ресурса (URI). URL. URN. Структура, применение.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Вопрос 1.**

**Область видимости**

(Могут спросить что угодно, опишу ниже лишь краткую сводку про области видимости и основные моменты).

Из внутренней области видимости имеется доступ к данным из внешней области видимости, но из внешней области видимости нет доступа к данным внутренней области видимости.

Короче, если речь зайдет про var - это полный отстой. Var - это устаревший способ объявления переменных, он вообще границ области видимости не знает. Var это полное говно. Область видимости переменных var ограничивается либо функцией, либо, если переменная глобальная, то скриптом. Такие переменные доступны за пределами блока.

Пример, демонстрирующий, какое var говно:

if (true) {

var test = true; // используем var вместо let

}

*alert(test); // true, переменная существует вне блока if*

*Var настолько мерзкий, что я не хочу про него больше писать.*

**Объявление переменных**

Мы можем объявить переменные для хранения данных с помощью ключевых слов var, let или const:

* let – это современный способ объявления;
* var – это устаревший способ объявления, пережиток прошлого;
* const – похоже на let, но значение переменной не может изменяться.

Раньше не нужно было указывать let, var или const для объявления переменных. В 2009 году появился ES5, который внес изменения в язык. Для активации этих изменений нужно добавить директиву “use strict”. В этом режиме мы обязаны ставить одно из ключевых слов let (let супер), var (var говно) или const для объявления переменных.

В JavaScript есть два ограничения, касающиеся имён переменных:

1. Имя переменной должно содержать только буквы, цифры или символы $ и \_.
2. Первый символ не должен быть цифрой.

P.S. Ну и конечно же именем переменной не может быть зарезервированное ключевое слово JS (удивительно и неожиданно).

P.P.S. Язык регистрозависимый, я не думаю, что нужно объяснять, что это значит при объявлении переменных.

**Замыкание**

Замыкание - функция, которая имеет доступ к переменным своего внешнего лексического окружения. Каждая функция в JS является замыканием. При запуске программы у каждой функции есть внутреннее свойство [[Enviroment]] - ссылка на внешнее лексическое окружение функции. (Функция в JS является объектом, поэтому у неё могут быть свойства).

При запуске функции создается объект лексического окружения функции с двумя свойствами:

1. Первое свойство - ссылка на внешнее лексическое окружение функции, которое берется из свойства [[Enviroment]] функции.
2. Второе свойство - список переменных/других функций, участвующих в функции.

Лексическое окружение есть у функции, блока кода, скрипта… у всего. Объект лексического окружения существует до того момента, пока есть функция, которая может его использовать.

А теперь главная идея замыкания: **когда код хочет получить доступ к переменной – сначала происходит поиск во внутреннем лексическом окружении, затем во внешнем, затем в следующем и так далее, до глобального.**

То есть, если мы пытаемся обратиться к переменным внутри функции, которые в ней не определены, поиск будет вестись во внешнем лексическом окружении и т. д.

Самый простой пример из всех возможных:

let counter = 0;

function makeCounter() {

counter++;

return counter;

}

console.log(makeCounter());

Что произойдет во время запуска скрипта? Если вкратце, то:

1. При запуске скрипта создается объект лексического окружения скрипта. У этого объекта будут два свойства: первое null, так как у него нет внешнего лексического окружения, а второй объект будет со свойствами counter и makeCounter.
2. Внутреннее свойства функции makeCounter [[Enviroment]] будет ссылаться на объект внешнего лексического окружения скрипта.
3. Когда выполнение скрипта дойдет до запуска функции makeCounter(), создастся объект её внешнего лексического окружения с двумя свойствами, первое из которых будет равен [[Enviroment]] функции, а второй будет пустой, так как внутри функции нет объявления переменных.
4. Во время выполнения функции переменная counter будет искаться сначала в лексическом окружении функции, а потом во внешнем лексическом окружении, то есть в лексическом окружении скрипта. И там эта переменная изменит своё значение.
5. Будет выведено один :).

Примечание: почему-то распространено заблуждение, что замыкание - это функция, которая возвращает функцию, и если функция не возвращает функцию - это не замыкание. Естественно, это полный бред, так как суть замыкания в том, что ведется поиск переменных во внешнем лексическом окружении, что собственно и демонстрирует первый пример. Во втором примере функция возвращает функции, однако его суть точно такая же - ведется поиск count во внешнем лексическом окружении. **Делайте акцент в объяснении второго примера именно на поиске переменной count во внешнем лексическом окружении.**

Второй пример:

function makeCounter() {

let count = 0;

return function() {

return count++; // есть доступ к внешней переменной "count"

};

}

let counter = makeCounter();

alert(counter()); // Будет выведено 0

alert(counter()); // Будет выведено 1

alert(counter()); // Будет выведено 2

2. Унифицированный идентификатор ресурса (URI). URL. URN. Структура, применение.

URI — символьная строка, позволяющая идентифицировать какой-либо ресурс: документ, изображение, файл, службу, ящик электронной почты и т.д. Прежде всего, речь идёт о ресурсах сети Интернет и Всемирной паутины. URI предоставляет простой и расширяемый способ идентификации ресурсов. Расширяемость URI означает, что уже существуют несколько схем идентификации внутри URI, и ещё больше будет создано в будущем.

URI является либо URL, либо URN, либо одновременно обоими.

URL — это URI, который, помимо идентификации ресурса, предоставляет ещё и информацию о местонахождении этого ресурса. А URN — это URI, который только идентифицирует ресурс в определённом пространстве имён (и, соответственно, в определённом контексте), но не указывает его местонахождение. Например, URN urn:ISBN-0-395-36341-1 — это URI, который указывает на ресурс (книгу) 0-395-36341-1 в пространстве имён ISBN, но, в отличие от URL, URN не указывает на местонахождение этого ресурса: в нём не сказано, в каком магазине её можно купить или на каком сайте скачать.

Впрочем, в последнее время появилась тенденция говорить просто URI о любой строке-идентификаторе, без дальнейших уточнений. Так что, возможно, термины URL и URN скоро уйдут в прошлое.

**URI = [ схема ":" ] иерархическая-часть [ "?" запрос ] [ "#" фрагмент ]**

The following are two example URIs and their component parts:

foo://example.com:8042/over/there?name=ferret#nose

\\_/ \\_\_\_\_\_\_\_\_\_\_\_\_\_\_/\\_\_\_\_\_\_\_\_\_/ \\_\_\_\_\_\_\_\_\_/ \\_\_/

| | | | |

scheme authority path query fragment

| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_

/ \ / \

Urn:example:animal:ferret:nose

В этой записи:

схема

схема обращения к ресурсу (часто указывает на сетевой протокол), например

http, ftp, file, ldap, mailto, urn

Иерархическая-часть

содержит данные, обычно организованные в иерархической форме, которые,

совместно с данными в неиерархическом компоненте запрос, служат для

идентификации ресурса в пределах видимости URIсхемы. Обычно иер-часть

содержит путь к ресурсу (и, возможно, перед ним, адрес сервера, на котором

тот располагается) или идентификатор ресурса (в случае URN.

запрос

этот необязательный компонент URI описан выше.

фрагмент

(тоже необязательный компонент)

RFC 3986

позволяет косвенно идентифицировать вторичный ресурс посредством ссылки

на первичный и указанием дополнительной информации. Вторичный

идентифицируемый ресурс может быть некоторой частью или подмножеством

первичного, некоторым его представлением или другим ресурсом,

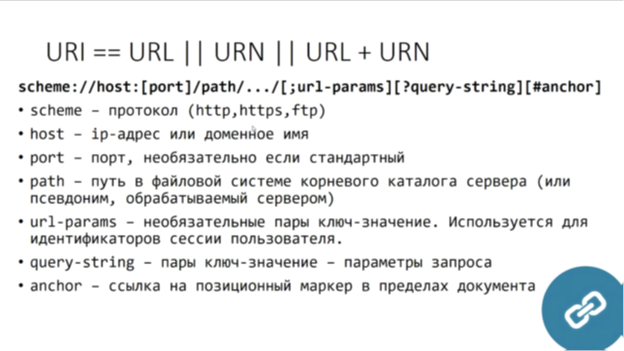
определённым или описанным таким ресурсом. Часть идентификатора URI без

схемы обращения к ресурсу часто называется «ссылкой URI» (англ. URI

reference). Прецеденты применения ссылок URI имеются в HTML, XHTML, XML и

XSLT. Процесс превращения ссылки URI в абсолютную форму URI называют

«разрешением URI» (англ. URI resolution).



# 

# -== IU7-Web Bilet #10 ==-

## 1. Эволюция HTTP. Протоколы HTTP2 и HTTP3. Основные идеи, решаемые проблемы. Тенденции дальнейшего развития.

HTTP/2 2016

http2 – это бинарный протокол.

Server-Push (решает проблему скорости первоначальной загрузки страницы);

Мультиплексирование.

Каждый поток имеет приоритет, используемый для того, чтобы показать другому участнику обмена, какие потоки считать более важными.

HTTP/3 — трансляция транспортного протокола QUIC для прикладного уровня.

Раньше НТТР/3 был известен как HTTP по QUIC, а до этого — как HTTP/2 по gQUIC, а ещё раньше — SPDY по gQUIC. Но суть в том, что HTTP/3 — просто новый синтаксис HTTP, который работает на протоколе IETF QUIC, мультиплексированном и безопасном транспорте на основе UDP.

Основная проблема - передача мультимедийного контента. HTTP/2 вводится в первую очередь для мультимедийного бинарного контента.

В чем суть: Одним из главных преимуществ HTTP/3 заявлялось повышение производительности, особенно при одновременном запросе нескольких объектов. В HTTP/2 любая помеха (потеря пакетов) в TCP-соединении блокирует сразу все потоки (блокировка начала строки). Поскольку HTTP/3 основан на UDP, то здесь при потере пакета прерывается только один поток, а не все.

## 2. Клиент в Web. Браузер. Алгоритм исполнения типового GET-запроса на web-ресурс.

Web-клиент (англ. Web client) как программа — браузер.

Браузер, или веб-обозреватель (web browser) — прикладное программное обеспечение для просмотра веб страниц, содержания веб-документов, компьютерных файлов и их каталогов; управления веб-приложениями; а также для решения других задач. В глобальной сети браузеры используют для запроса, обработки, манипулирования и отображения содержания веб-сайтов. Многие современные браузеры также могут использоваться для обмена файлами с серверами FTP, а также для непосредственного просмотра содержания файлов многих графических форматов , аудио-видео форматов, текстовых форматов и других файлов.

Одним из способов, как можно отправить запрос по протоколу HTTP к серверу, является запрос методом GET. Этот метод является самым распространенным и запросы к серверу чаще всего происходят с его использованием.

Самый простой способ, как можно создать запрос методом GET - это набрать URL адрес в адресную строку браузера.

Браузер сформирует и передаст веб-серверу примерно следующую информацию: GET / HTTP/1.1 Host: site.ru

User-Agent: Mozilla/5.0 Windows NT 6.1; rv:18.0 Gecko/20100101 Firefox/18.0 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,/;q=0.8

Accept-Language: ru-RU,ru;q=0.8,en-US;q=0.5,en;q=0.3 Accept-Encoding: gzip, deflate

Cookie: wp-settings Connection: keep-alive

Запрос состоит из двух частей:

* строка запроса (Request Line)
* заголовки (Message Headers)

Обратите внимание, что GET запрос не имеет тела сообщения. Но, это не означает, что с его помощью мы не можем передать серверу никакую информацию. Это можно делать с помощью специальных GET параметров.

Чтобы добавить GET параметры к запросу, нужно в конце URL адреса поставить знак «?» и после него начинать задавать их по следующему правилу:

имя\_параметра1=значение\_параметра1&имя\_параметра2=значение\_параметра2&…

Разделителем между параметрами служит знак «&».

К примеру, если мы хотим передать серверу два значения, имя пользователя и его возраст, то это можно сделать следующей строкой: http://site.ru/page.php?name=dima&age=27

Когда выполнен данный запрос, данные попадают в так называемую переменную окружения QUERY\_STRING, из которой их можно получить на сервере с помощью серверного языка веб-программирования.

Сервер обработает запрос, после чего вернет код ответа и содержимое.

# -== IU7-Web Bilet #11 ==-

## 1. JS (ES5). Встроенные типы данных.

## 2. HTML. Атрибуты.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Вопрос 1.**

В JavaScript есть 8 основных типов:

* number для любых чисел: целочисленных или чисел с плавающей точкой; целочисленные значения ограничены диапазоном ±(253-1). Кроме обычных чисел, существуют так называемые «специальные числовые значения», которые относятся к этому типу данных: Infinity, -Infinity и NaN;
* bigint для целых чисел произвольной длины;
* string для строк. Строка может содержать ноль или больше символов, нет отдельного символьного типа;
* boolean для true/false;
* null для неизвестных значений – отдельный тип, имеющий одно значение null;
* undefined для неприсвоенных значений – отдельный тип, имеющий одно значение undefined;
* object для более сложных структур данных. Каждая функция в JS так же является объектом;
* symbol для уникальных идентификаторов.

Оператор typeof позволяет нам увидеть, какой тип данных сохранён в переменной.

* Имеет две формы: typeof x или typeof(x).
* Возвращает строку с именем типа. Например, "string".
* Для null возвращается "object" – это ошибка в языке, на самом деле это не объект.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Вопрос 2.**

HTML-атрибуты сообщают браузеру, каким образом должен отображаться тот или иной элемент страницы. Атрибуты позволяют сделать более разнообразными внешний вид информации, добавляемой с помощью одинаковых элементов.

Значение атрибута заключается в кавычки "". Названия и значения атрибутов не чувствительны к регистру, но, тем не менее, рекомендуется набирать их в нижнем регистре.

Имеется ряд глобальных атрибутов, которые могут использоваться для любого элемента, например style для задания стилей, class и id (как селекторы), title (подсказка для страницы) и т. д.

# -== IU7-Web Bilet #12 ==-

1. Паттерны проектирования MVC. Примеры паттернов MVC-семейства. Расшифровка аббревиатуры. Описание архитектуры.

2. CSS. Правила каскадирования и наследования.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Паттерны проектирования MVC. Примеры паттернов MVC-семейства. Расшифровка аббревиатуры. Описание архитектуры.

Наиболее распространенные виды MVC-паттерна, это:

• Model-View-Controller

• Model-View-Presenter

• Model-View-View Model

• (Model View Intent)(какой-то таинственный про который мало пишут)

• Model-Template-View - Django-взгляд на MVC (на самом деле, классический MVC, только V == T, а C == V)

Семейство MVCпаттернов (по лекциям):

• MVC;

• MTV;

• MPV;

• MVVM;

• MVPVM.

MVC расшифровывается как модель-представление-контроллер (от англ. modelview-

controller). Это способ организации кода, который предполагает выделение блоков, отвечающих за решение разных задач. Один блок отвечает за данные приложения, другой отвечает за внешний вид, а третий контролирует работу приложения.

Компоненты MVC:

1. Модель — этот компонент отвечает за данные, а также определяет структуру приложения. Например, если вы создаете To-Do приложение, код компонента model будет определять список задач и отдельные задачи.

2. Представление — этот компонент отвечает за взаимодействие с пользователем. То есть код компонента view определяет внешний вид приложения и способы его использования.

3. Контроллер — этот компонент отвечает за связь между model и view. Код компонента controller определяет, как сайт реагирует на действия пользователя. По сути, это мозг MVC-приложения. Контроллер решает задачи, он не должен включать в себя бизнес-логику приложения. Не стоит заниматься существенным в контроллере, иначе возникает FSUC -Fat Stupid Ugly Controllers).

MVC — подход к проектированию приложения, который предполагает выделение кода в блоки модель, представление и контроллер. Контроллер обрабатывает входящие запросы. Модель достаёт из базы данных информацию, нужную для выполнения конкретных запросов. Представление определяет результат запроса, который получает пользователь.



2. CSS. Правила каскадирования и наследования.

**Каскадирование** — это механизм, который управляет конечным результатом в ситуации, когда к одному элементу применяются разные CSS-правила. Существует три критерия, которые определяют порядок применения свойств:

* правило !important;
* Специфичность;
* порядок, в котором подключены таблицы стилей.

##### **Правило !important**

Вес правила можно задать с помощью ключевого слова !important, которое добавляется сразу после значения свойства, например, span {font-weight: bold!important;}. Правило необходимо размещать в конец объявления перед закрывающей скобкой, без пробела. Такое объявление будет иметь приоритет над всеми остальными правилами. Это правило позволяет отменить значение свойства и установить новое для элемента из группы элементов в случае, когда нет прямого доступа к файлу со стилями.

##### **Специфичность**

Для каждого правила браузер вычисляет **специфичность селектора**, и если у элемента имеются конфликтующие объявления свойств, во внимание принимается правило, имеющее наибольшую специфичность. Значение специфичности состоит из четырех частей: 0, 0, 0, 0. Специфичность селектора определяется следующим образом:

* для id добавляется 0, 1, 0, 0;
* для class добавляется 0, 0, 1, 0;
* для каждого элемента и псевдоэлемента добавляется 0, 0, 0, 1;
* для встроенного стиля, добавленного непосредственно к элементу — 1, 0, 0, 0;
* универсальный селектор не имеет специфичности.

В результате к элементу применятся те правила, специфичность которых больше. Например, если на элемент действуют две специфичности со значениями 0, 0, 0, 2 и 0, 1, 0, 1, то выиграет второе правило.

##### **Порядок подключённых таблиц**

Вы можете создать несколько внешних таблиц стилей и подключить их к одной веб-странице. Если в разных таблицах будут встречаться разные значения свойств одного элемента, то в результате к элементу применится правило, находящееся в таблице стилей, идущей в списке ниже.

**Наследование** является механизмом, с помощью которого определенные свойства передаются от предка к его потомкам.

Спецификацией CSS предусмотрено наследование свойств, относящихся к текстовому содержимому страницы, таких как color, font, letter-spacing, line-height, list-style, text-align, text-indent, text-transform, visibility, white-space и word-spacing. Во многих случаях это удобно, так как не нужно задавать размер шрифта и семейство шрифтов для каждого элемента веб-страницы.

Свойства, относящиеся к форматированию блоков, не наследуются. Это background, border, display, float и clear, height и width, margin, min-max-height и -width, outline, overflow, padding, position, text-decoration, vertical-align и z-index.

С помощью ключевого слова inherit можно принудить элемент наследовать любое значение свойства родительского элемента. Это работает даже для тех свойств, которые не наследуются по умолчанию.

# -== IU7-Web Bilet #13 ==-

## 1. История развития сети Интернет и WWW.

Пума навалила дат на страницу, тут урезано. Файл теории 2019.

1958 - В США создали ARPA(advanced research Project Agency), исследует обеспечение связи для военных.

1961 - Леонард Клейнрок описывает разбитие файлов на куски. Начало пакетной передачи.

1969 - сеть ARPANET. Связывает лаборатории и исследовательские центры США.

1971 - разработана система эл. Почты.

1974 - Telenet, коммерческая версия ARPANET.

1976 - Роберт Метклаф создал Ethernet - первая локальная сеть

1982 - ARPA создала TCP/IP

1983 - ARPANET стала называться Internet.

1980е - бурный рост. 1987 - 10, 1989 - 100 тысяч хостов

1989 - Сэр Тим Бернерс-Ли в процессе разработки WWW, HTTP

1991 - CERN представила WWW, HTTP/0.9, HTML. Первые компьютерные вирусы

1993 - первый браузер Mosaic. 2млн хостов.

1995 - ECMAScript(JS)

1996 - CSS

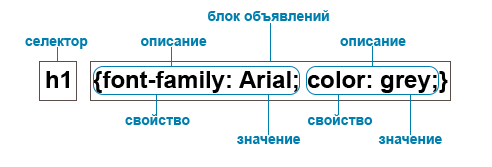
1999 - Первые попыти цензуры.

2000-н.в - модернизация, привод к текущему виду, HTTP/2, HTTP/3, ECMAScript2021, HTML5.

## 2. Каскадные таблицы стилей. CSS. История. Синтаксис языка.

**CSS (Cascading Style Sheets)** — язык таблиц стилей, который позволяет прикреплять стиль (например, шрифты и цвет) к структурированным документам (например, документам HTML и приложениям XML).

Стиль задается с помощью селектора и объявления:



Каскадные таблицы стилей описывают правила форматирования элементов с помощью свойств и допустимых значений этих свойств. Для каждого элемента можно использовать ограниченный набор свойств, остальные свойства не будут оказывать на него никакого влияния.

Существует 4 способа задания стилей: внешние таблицы стилей, внутренние таблицы стилей, встроенные стили и правило @import

**Внешняя таблица стилей** представляет собой текстовый файл с расширением .css, в котором находится набор CSS-стилей элементов. Файл создаётся в редакторе кода, так же как и HTML-страница. Внутри файла могут содержатся только стили, без HTML-разметки. Внешняя таблица стилей подключается к веб-странице с помощью тега <link>, расположенного внутри раздела <head></head>. Такие стили работают для всех страниц сайта.

К каждой веб-странице можно присоединить несколько таблиц стилей, добавляя последовательно несколько тегов <link>, указав в атрибуте тега media назначение данной таблицы стилей. rel="stylesheet" указывает тип ссылки (ссылка на таблицу стилей).

<head>

<link rel="stylesheet" href="css/style.css">

<link rel="stylesheet" href="css/assets.css" media="all">

</head>

**Внутренние стили** встраиваются в раздел <head></head> HTML-документа и определяются внутри тега <style></style>. Внутренние стили имеют приоритет над внешними, но уступают встроенным стилям (заданным через атрибут style).

<head>

<style>

h1,

h2 {

color: red;

font-family: "Times New Roman", Georgia, Serif;

line-height: 1.3em;

}

</style>

</head>

<body>

...

</body>

Когда мы пишем **встроенные стили**, мы пишем CSS-код в HTML-файл, непосредственно внутри тега элемента с помощью атрибута style:

<p style="font-weight: bold; color: red;">Обратите внимание на этот текст.</p>

Такие стили действуют только на тот элемент, для которого они заданы.

**Правило @import** позволяет загружать внешние таблицы стилей. Чтобы директива @import работала, она должна располагаться в таблице стилей (внешней или внутренней) перед всеми остальными правилами:

@import url(mobile.css);

Мне кажется, этого и немного из истории ниже будет достаточно, но может кто-то хочет расписать всё, что есть в CSS… тогда читайте дальше (но селекторы и каскадирование с наследованием есть в других билетах.

**CSS - cascade style sheet -** язык описания внешнего вида документа, созданного с помощью языка разметки (html).

Термин «каскадные таблицы стилей» был предложен Хоконом Ли в 1994 году. Совместно с Бертом Босом он стал развивать CSS.

1996 - CSS

2009 - CSS2

CSS3 - Общее название новых, пока не стандарт. Главной особенностью CSS3 является возможность создавать анимированные элементы без использования [JS](https://ru.wikipedia.org/wiki/JavaScript)[[5]](https://ru.wikipedia.org/wiki/CSS#cite_note-5), поддержка линейных и радиальных градиентов, теней, сглаживания и прочее

**История**

До появления CSS оформление веб-страниц осуществлялось исключительно средствами HTML, непосредственно внутри содержимого документа.

Однако с появлением CSS стало возможным принципиальное разделение содержания и представления документа. За счёт этого нововведения стало возможным лёгкое применение единого стиля оформления для массы схожих документов, а также быстрое изменение этого оформления.

CSS используется для задания цветов, шрифтов, стилей, расположения отдельных блоков и других аспектов представления внешнего вида веб-страниц.

Основной целью разработки CSS являлось отделение описания логической структуры веб-страницы (HTML или других языков разметки) от описания внешнего вида этой веб-страницы (CSS).

Такое разделение может увеличить доступность документа, предоставить большую гибкость и возможность управления его представлением, а также уменьшить сложность и повторяемость в структурном содержимом.

Кроме того, CSS позволяет представлять один и тот же документ в различных стилях или методах вывода, таких как экранное представление, печатное представление, чтение голосом (специальным голосовым браузером или программой чтения с экрана) или при выводе устройствами, использующими шрифт Брайля

Преимущества:

* Несколько дизайнов страницы для разных устройств просмотра. Например, на экране дизайн будет рассчитан на большую ширину
* Уменьшение времени загрузки страниц сайта за счёт переноса правил представления данных в отдельный CSS-файл. В этом случае браузер загружает только структуру документа и данные, хранимые на странице, а представление этих данных загружается браузером только один раз и может быть [закэшировано](https://ru.wikipedia.org/wiki/%D0%9A%D1%8D%D1%88).
* Простота последующего изменения дизайна.
* Дополнительные возможности оформления.

Недостатки:

* Различное отображение вёрстки в различных браузерах (особенно устаревших)
* Часто встречающаяся необходимость на практике исправлять не только один CSS-файл, но и теги HTML, которые сложным и ненаглядным способом связаны с селекторами CSS, что иногда сводит на нет простоту применения единых файлов стилей и значительно увеличивает время редактирования и тестирования.

**Применение**

Применение CSS к документам HTML основано на принципах ***наследования*** и ***каскадирования***.

Принцип ***наследования*** заключается в том, что свойства CSS, объявленные для элементов-предков, почти всегда наследуются элементами-потомками.

Принцип ***каскадирования*** применяется в случае, когда какому-то элементу HTML одновременно поставлено в соответствие более одного правила CSS, то есть, когда происходит конфликт значений этих правил. Чтобы разрешить такие конфликты, вводятся правила приоритета.

Приоритеты общих стилей:

Стиль браузера < пользовательский для браузера < стили автора страницы

Приоритеты стилей для элемента:

Стиль предка < внешние подключенные таблицы стилей < непосредственно из блока style в соответствии с приоритетами с електоров

Способы подключения в документ:

* Через <link> в <head>  
  <**link** rel="stylesheet" type="text/css" href="style.css">
* @import в элементе<style> в head  
  <**style** media="all"> @**import** **url**(**style**.**css**); </**style**>
* <style> в <head> напрямую  
  <**style**>

**body** {

**color**: **red**;

}

</**style**>

* в атрибуте style отдельного элемента

**Синтаксис. Селекторы**

Весь синтаксис:

**селектор, селектор {**

**свойство: значение;**

**свойство: значение;**

**свойство: значение;**

**}**

Селекторы от менее уникальных к более:

**Универсальный селектор**

**\* {**

**margin: 0;**

**}**

**Селектор тегов**

**p {**

**font-family: arial, helvetica, sans-serif;**

**}**

**Селектор классов**

**.note {**

**color: red;**

**}**

**Селектор идентификаторов**

**#paragraph1 {**

**margin: 0;**

**}**

**Селектор атрибутов**

**a[href="http://www.somesite.com"] {**

**font-weight: bold;**

**}**

**Селектор потомков (контекстный селектор**

**div#paragraph1 p.note {**

**color: green;**

**}**

**Селектор дочерних элементов**

**p.note > b {**

**color: blue;**

**}**

**Селектор элементов одного уровня**

**h1 + p {**

**font-size:24px;**

**}**

**Селектор псевдоклассов**

**a:active {**

**color: blue;**

**}**

**Селектор псевдоэлементов**

**p::first-letter {**

**font-size: 32px;**

**}**

# -== IU7-Web Bilet #14 ==-

## 1. Методы и заголовки протокола HTTP 1.1

HTTP - Hyper Text Transfer Protocol

Особенности протокола:

Текстовый (вся коммуникация между данными - символьный формат) протокол;

Без сохранения состояния (отсутствует понятие состояния (только “костыль - cookie”)).

Назначение: Протокол HTTP предназначен для передачи содержимого в Интернете.

Стандарт HTPP/1.1: <https://datatracker.ietf.org/doc/html/rfc7231#section-4>

Patch метод: <https://datatracker.ietf.org/doc/html/rfc5789#section-2>

**HTTP/1.1 методы:**

* **GET** - запрос документа;
* **HEAD** - аналог GET, но без тела ответа (метаданные без данных);
* **POST** - передача данных;
* **PUT** - размещение файла по URI/изменение данных;
* **DELETE** - удаление файла по URI/удаление данных;
* **CONNECT** - используются редко;
* **OPTIONS** - спользуется для описания параметров соединения с ресурсом, запрос методов сервера;
* **TRACE** - выполняет вызов возвращаемого тестового сообщения с ресурса;
* **PATCH** - частичное изменение данных на сервере.

**HTTP/1.1 заголовки:**

**Все заголовки разделяются на четыре основных группы:**

* **General Headers (рус. Общие заголовки) —** используются в запросах и ответах.
* **Request Headers (рус. Заголовки запроса) —** используются только в запросах.
* **Response Headers (рус. Заголовки ответа) —** используются только в ответах.
* **Entity Headers (рус. Заголовки сущности) —** сопровождают каждую сущность сообщения. Используются в запросах и ответах.

Их много…

* **User-Agent -** информация о средстве (браузер, утилита), совершившем запрос
* **Accept -** принимаемый тип данных ответа (text/plain; text/html, etc)
* **Accept-Charset -** принимаемые кодировки ответа (UTF-8, etc).
* **Accept-Encoding -** принимаемое “content-codings“ ответа. Тут gzip, compress и подобное.
* **Accept-Language -** предпочитаемые User-Agent’ом языки ответа
* **Cache-Control -** указать директивы кеша для цепочки запросов / ответов
* **Expect -** запрос показывает, что определенное поведение необходимо для корректной обработки этого запроса
* **Host -** предоставляет хост и порт из целевого URI
* **Pragma -** позволяет поддерживать обратную совместимость с HTTP/1.0 кешами
* **Range**
* **TE**
* **Max-Forwards -** ограничивает кол-во проксирований
* **If-Match**
* **If-None-Match**
* **If-Modified-Since**
* **If-Unmodified-Since**
* **If-Range**
* **Authorization -** позволяет идентифицировать себя серверу
* **Proxy-Authorization -** позволяет идентифицировать себя для прокси, требующей аутентификацию
* **From -** email пользователя
* **Referer** - URI, откуда User-Agent узнал об этом URI

## 2. Клиент-серверная архитектура. Принципы построения. Понятие клиента и сервера в Web.

Клиент-сервер - сетевая архитектура, в которой сетевая нагрузка распределена между поставщиками услуг, называемыми серверами и заказчиками услуг, называемыми клиентами.

*Классический "клиент-сервер" в WEB:*

* Взаимодействие по протоколу HTTP/протоколам поверх HTTP
* HTTP работает поверх TCP
* Абстрагирование от физической реализации сети
* Адресация - сокетах (ip-фдрес + порт)
* Доменные имена преобразуются в ip-адрес
* Клиент - всегда инициатор сессии

Для разработки клиент / серверных систем имеется два подхода:

* - построение систем на основе двухзвенной архитектуры;
* - построение систем на основе трехзвенной архитектуры.

**Двухзвенная архитектура** состоит из клиентской и серверной части. Как правило,

серверная часть представляет собой сервер БД, на котором расположены общие

данные. А клиентская часть представляет приложение, которое связывается с сервером БД, осуществляет к нему запросы и получает ответы.

При разработке информационных систем, рассчитанных на широкую аудиторию,

возникают проблемы с использованием двухзвенной архитектуры. Во-первых,

пользователю необходимо иметь в наличии клиентскую часть, а, во-вторых, у

неопытного пользователя, могут возникнуть проблемы с конфигурированием такой

системы. Поэтому в последнее время, более часто разрабатывают приложения на базе трехзвенной архитектуры.

**Трезвенная архитектура** также состоит из двух частей: клиента и сервера. Но серверная часть в этой архитектуре представляет собой сервер приложений и сервер БД. А в качестве клиента выступает web браузер. Пользователю необходимо знать только адрес сервера приложения и иметь web браузер. Передача запросов от клиента к серверу приложений происходит по CGI интерфейсу. Сервер приложений общается с сервером БД, используя другой интерфейс, зависящий от того, на основе каких средств строится конкретная информационная система.

Сервером мы считаем абстрактную машину в сети, способную получить HTTP-запрос,

обработать его и вернуть корректный ответ. Не важны его физическая суть и внутренняя архитектура.

Клиентом может быть все, что угодно, что способно сформировать и отправить HTTPзапрос. Клиентом может быть JavaScript-сценарий, работающий в браузере, мобильное приложение, демон, запущенный на сервере и т.д.

3. Задача на: Nginx/REST/Архитектуру/Дизайн/Верстку/JS. В основном - на клиентский JS.

# -== IU7-Web Bilet #15 ==-

**1. Понятие шаблонизатора и маршрутизатора в MVC-фреймворке на сервере и клиенте.**

Система маршрутизации определяет, какое приложение должно запуститься и какую операцию это приложение должно выполнить.

Для формирования HTML кода Шаблонизатор на основе соответствующих шаблонов формирует код веб-страницы с использованием полученных данных.

***Шаблонизатор*** (в web) — это программное обеспечение, позволяющее DNSиспользовать html-шаблоны для генерации конечных html-страниц. Основная цель использования шаблонизаторов — это отделение представления данных от исполняемого кода. Часто это необходимо для обеспечения возможности параллельной работы программиста и дизайнера-верстальщика.

Класс MVC ***Router*** (который является частью более широкого *фронтального контроллера* ) разбивает URL запроса HTTPв частности, компонент пути (и потенциально строку запроса).

Router пытается сопоставить первую или две части компонента path соответствующей комбинации маршрутов ( Controller / Action [ method ], или просто Controller , который выполняет действие по умолчанию (method).

Действие или команда-это просто метод от конкретного Controller .

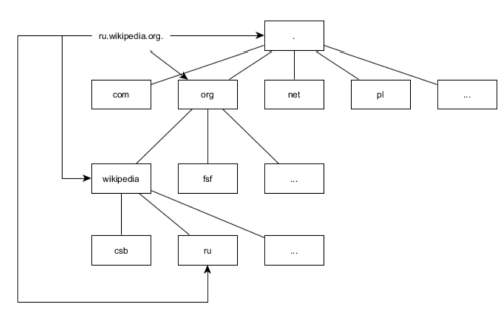
Обычно существует abstract Controller и много потомков Controller , по одному для каждой веб-страницы (вообще говоря).

***2. Система доменных имен (DNS). Назначение, расшифровка, алгоритм работы. Пример работы.***

DNS Domain Name System - система доменных имён) - система для получения информации о доменах.

Изначально адресация узлов сети шла исключительно по IPадресам. Запоминать их неудобно, что повлекло появление DNS (большой самообновляемый реестр записи, распределённый по всему миру; делегирует записи и ответственность).

Пример делегирования полномочий:



URI == URL || URN || URL + URN

Scheme - протокол (http, https, ftp)

Host - IPадрес или доменное имя

Port - порт (необязательно, если стандартный)

Path - путь в файловой системе корневого каталога сервера (или псевдоним, обрабатываемый сервером); может быть виртуальный путь

Url-params - необязательные пары ключ-значение, используются для идентификаторов сессии пользователя;

Query-string - пары ключ-значение (параметры запроса)

Anchor - ссылка на позиционный маркер в пределах документа

scheme://host:[port]/path/.../[;url-params][?query-string][#anchor]

***Как работает DNS***

Система доменных имен состоит из следующих компонентов:

Иерархическая структура доменных имен:

* Доменные зоны верхнего уровня (первого уровня) – например: "ru", "com", или "org". Они включают в себя все доменные имена, входящие в эту зону. В любую доменную зону может входить неограниченное количество доменов.
* Доменные имена (доменные зоны второго уровня) – например: "google.com" или "yandex.ru". Т.к. система доменных имен является иерархичной, то "yandex.ru" можно также назвать поддоменом вышестоящей зоны "ru". Поэтому, правильнее указывать именно уровень домена. Однако, на практике, доменную зону любого уровня называют просто «доменом».
* Поддомены (доменные зоны третьего уровня) – например: "api.google.com" или "mail.yandex.ru". Могут быть доменные зоны 4, 5 уровней и так далее. "www.gооgle.com" и "google.com" - это, фактически, разные домены. Надо не забывать указывать А-записи для каждого из них.

Для того, чтобы узнать IP адрес, домена компьютер / сервер обращается к DNSсерверу, который указан у него в сетевых настройках. Обычно, это DNS сервер Интернет провайдера. DNS сервер проверяет делегирован домен ему или нет. Если да, то сразу отвечает на запрос. Если нет, то запрашивает информацию о DNS сервере, обслуживающем этот домен, у корневого сервера, и затем у сервера доменных зон верхнего уровня. После этого, непосредственно делает запрос на NS (name server) сервер, обслуживающий этот домен, и транслирует ответ вашему компьютеру / серверу

3. Задача на: Nginx/REST/Архитектуру/Дизайн/Верстку/JS. В основном - на клиентский JS.

# -== IU7-Web Bilet #16 ==-

**1. Макетирование. Основные подходы и идеи. Инструменты (Figma).**

Макет - это способ организации материалов на веб-странице. Самих макетов существует огромное количество, но их, тем не менее, можно систематизировать по ряду признаков. Наиболее популярным является деление макетов по ширине и количеству колонок.

Различают пять типов макетов, связанных с шириной:

1. фиксированные;
2. резиновые;
3. эластичные;
4. адаптивные;
5. комбинированные.

**1. Фиксированный макет**

Макет обычно располагается по центру окна браузера, а его ширина ограничивается заданными размерами в пикселах

**Преимущества**

* Из-за того, что ширина всех колонок известна, проще указывать размеры изображений, видео и других элементов страницы.
* Браузеры, как правило, лояльнее относятся к таким макетам, поэтому на вёрстку и отладку уходит меньше времени.

**Недостатки**

* Сайт смотрится плохо на мониторах с высоким разрешением, неэффективно используя свободное место.

**2. Резиновый макет**

Ширина колонок задается в процентах или сочетаются проценты и пикселы таким образом, что макет занимает всю свободную ширину окна браузера. При изменении размеров окна или другом разрешении монитора макет подстраивается под них.

**Преимущества**

* Используется вся эффективная область страницы.
* Веб-страницы удобно печатаются на бумаге любого формата.
* Веб-страницы хорошо смотрятся на разных устройствах от iPhone до ноутбука.

**Недостатки**

* На мониторах с высоким разрешением сайт плохо читается из-за чрезмерного удлинения строк текста. Здесь помогает ограничение ширины контента с помощью свойства max-width. Опять же некоторые владельцы больших мониторов уменьшают окно браузера до комфортных для них размеров.
* Резиновые макеты сложнее верстать и отлаживать в разных браузерах.

**3. Эластичный макет**

Этот макет по своему виду может не отличаться от фиксированного или резинового макета. Размер элементов задаётся не в пикселах и процентах, а в em, привязанному к размеру шрифта. Значение em можно использовать не для всех элементов, оставляя ширину некоторых фиксированной.

**Преимущества**

* Макет целиком или отдельные его части легко масштабировать, подгоняя под комфортный для восприятия размер.
* Макет будет одинаково смотреться на разных операционных системах, имеющих различия в выборе размера и типа шрифта.

**Недостатки**

* В современных браузерах функция масштаба страницы уже встроена, и пользоваться ей довольно удобно.
* Верстать эластичный макет крайне сложно, поскольку единица em имеет относительные размеры и зависит от используемого шрифта.
* В действительности сфера применения этого макета очень ограниченна.

**4. Адаптивный макет**

Этот макет подстраивается под разрешение монитора и окна браузера, меняя при необходимости ширину макета, число колонок, размеры изображений и текста. Для этого заготавливается несколько стилевых правил или файлов под разный диапазон разрешений, выбор правил происходит через скрипты или CSS3, которые и определяют нужную для этого информацию о пользователе.

**Преимущества**

* Этот тип макета наиболее удобен для пользователя, поскольку не зависит от разрешения и ширины окна браузера, приспосабливаясь под них.
* Макет комфортно можно смотреть на любом устройстве.

**Недостатки**

* Это самый сложный тип из всех макетов, ведь, по сути, вместо одного требуется сделать несколько макетов со своей графикой и CSS, а также прописать механизм определения разрешения монитора или ширины окна браузера.
* За счёт универсальности макет сложно проверять на разные условия, которые возможны у пользователей.

**5. Комбинированный макет**

Этот макет предполагает использование разной ширины для отдельных частей страницы, например, шапку и подвал делают резиновыми, а контент фиксированным.

Этот макет в действительности не является самостоятельным типом, поэтому наследует все плюсы и минусы фиксированного и резинового макета.

Надо отметить, что некоторые макеты, хотя и выглядят комбинированными, в действительности ими не являются. Резиновая шапка страницы может оказаться всего-навсего широким фоновым рисунком.

**Макеты по колонкам**

Наиболее распространенным вариантом является наличие на веб-странице двух колонок — одна из них, как правило, содержит навигацию, а во второй, более широкой колонке, размещается контент. Для резиновых макетов имеет смысл установить три колонки, чтобы эффективно использовать полезную площадь веб-страницы. В любом случае выбор числа колонок зависит исключительно от объема информации на сайте и способе её организации.

Figma — это графический онлайн-редактор для дизайнеров интерфейсов и вебразработчиков. Сейчас это удобная, бесплатная альтернатива Photoshop. Большое преимущество платформы — возможность работать прямо в браузере. При этом есть и десктопная версия.

**2. Паттерны проектирования MVC. Примеры паттернов MVC-семейства. Расшифровка аббревиатуры. Описание архитектуры.**

Наиболее распространенные виды MVC-паттерна, это:

· Model-View-Controller

· Model-View-Presenter

· Model-View-ViewModel

· Model-View-Intent

· Model-Template-View - Django-взгляд на MVC (на самом деле, классический MVC, только V == T, а C == V)

Семейство MVC-паттернов:

· MVC;

· MTV;

· MPV;

· MVVM;

· MVPVM.

MVC расшифровывается как модель-представление-контроллер (от англ. Model-View-Controller). Это способ организации кода, который предполагает выделение блоков, отвечающих за решение разных задач. Один блок отвечает за данные приложения, другой отвечает за внешний вид, а третий контролирует работу приложения.

**Компоненты MVC**

Модель — этот компонент отвечает за данные, а также определяет структуру приложения. Например, если вы создаете To-Do приложение, код компонента model будет определять список задач и отдельные задачи.

Представление — этот компонент отвечает за взаимодействие с пользователем. То есть код компонента view определяет внешний вид приложения и способы его использования.

Контроллер — этот компонент отвечает за связь между model и view. Код компонента controller определяет, как сайт реагирует на действия пользователя. По сути, это мозг MVC-приложения. Контроллер решает задачи, он не должен включать в себя бизнес-логику приложения. Не стоит заниматься существенным в контроллере, иначе возникает FSUC (Fat Stupid Ugly Controllers).

MVC — подход к проектированию приложения, который предполагает выделение кода в блоки модель, представление и контроллер.

Контроллер обрабатывает входящие запросы. Модель достаёт из базы данных информацию, нужную для выполнения конкретных запросов. Представление определяет результат запроса, который получает пользователь.



# -== IU7-Web Bilet #17 ==-

**1.** **Варианты организации клиент-серверной архитектуры приложения в веб. Толстый, тонкий и изоморфный клиенты. Определение, принципы.**

Клиент-сервер - сетевая архитектура, в которой сетевая нагрузка распределена между поставщиками услуг, называемыми серверами и заказчиками услуг, называемыми клиентами.

**Классический "клиент-сервер" в WEB:**

· Взаимодействие по протоколу HTTP/протоколам поверх HTTP

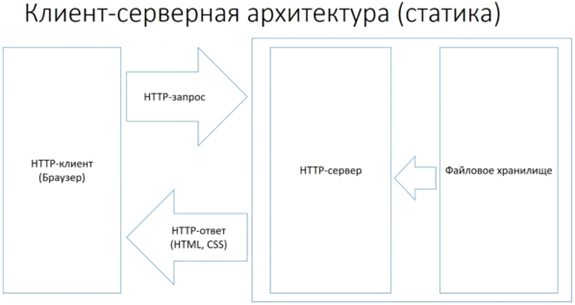
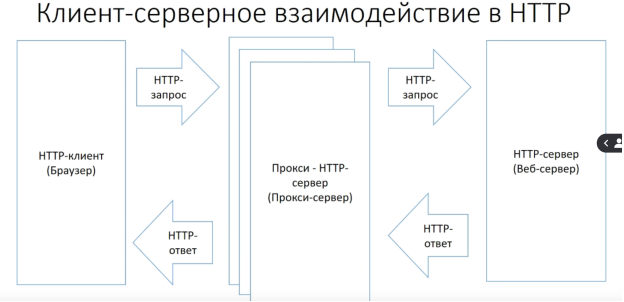
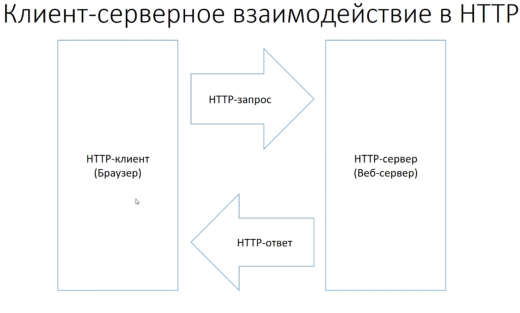
· HTTP работает поверх TCP

· Абстрагирование от физической реализации сети

· Адресация - сокетах (ip-адрес + порт)

· Доменные имена преобразуются в ip-адрес

· Клиент - всегда инициатор сессии



Толстый клиент, rich client архитектуре клиент-сервер — это приложение, обеспечивающее (в противовес тонкому клиенту) полную функциональность и независимость от центрального сервера. Часто сервер в этом случае является лишь хранилищем данных, а вся работа по обработке и представлению этих данных переносится на машину клиента.

Тонкий клиент, thin client в компьютерных технологиях — компьютер или программа-клиент в сетях с клиент-серверной или терминальной архитектурой, где большая часть задач по обработке информации перенесена на сервер и права доступа клиента строго ограничены.

Изоморфный совмещает в себе элементы толстого и тонкого клиента.

**2. JS (ES5). Функции. This. Функция-конструктор.**

**Функции**

В JS существует 3 способа создания функций:

1. Function Declaration;
2. Function Expression;
3. Стрелочная функция (самое вкусное).

Рассмотрим сначала первые два из них.

Первое отличие двух способов. В случае Function Declaration функция объявляется отдельной конструкцией «function…» в основном потоке кода. Синтаксис Function Declaration выглядит следующим образом:

*function sayHi() {*

*alert( "Привет" );*

*}*

В случае Function Expression функция создается внутри другого выражения или синтаксической конструкции. В данном случае функция создаётся в правой части «выражения присваивания».

*let sayHi = function() {*

*alert( "Привет" );*

*};*

Более тонкое отличие состоит, в том, *когда* создаётся функция движком JavaScript. Function Expression создаётся, когда выполнение потока кода доходит до него, и затем уже может использоваться.

Function Declaration можно использовать во всем скрипте (или блоке кода, если функция объявлена в блоке). Другими словами, когда движок JavaScript *готовится* выполнять скрипт или блок кода, прежде всего он ищет в нём Function Declaration и создаёт все такие функции. Можно считать этот процесс «стадией инициализации».

В результате, функции, созданные, как Function Declaration могут быть вызваны раньше своих определений.

Существует ещё более простой и краткий синтаксис для создания функций, который часто лучше, чем синтаксис Function Expression. Он называется «функции-стрелки» или «стрелочные функции» (arrow functions), т.к. выглядит следующим образом: let func = (arg1, arg2, ...argN) => expression

Другими словами, это более короткий вариант такой записи:

let func = function(arg1, arg2, ...argN) {

return expression;

};

Правила синтаксиса: если нет аргументов, то пустые скобки, если 1 - скобки можно опустить, если 2 и более - скобки нужны. Если в теле функции одно выражение, можно не писать фигурные скобки и return, иначе их указывать обязательно.

Стрелочные функции:

* Не имеют this. Если происходит обращение к this, его значение берётся снаружи.
* Не имеют arguments (короче не можем получит список аргументов через arguments)
* Не могут быть вызваны с new (не могут быть функциями конструкторами)
* У них также нет super (наследование классов)

**this**

Для доступа к информации внутри объекта метод может использовать ключевое слово this.

Значение this – это объект «перед точкой», который использовался для вызова метода.

**Функция-конструктор**

Такая функция позволяет создать множество однотипных объектов.

Функции-конструкторы являются обычными функциями. Но есть два соглашения:

1. Имя функции-конструктора должно начинаться с большой буквы.
2. Функция-конструктор должна вызываться при помощи оператора "new".

Пример:

function User(name) {

// this = {} неявно

this.name = name;

this.isAdmin = false;

// return this неявно

}

let user = new User("Вася");

Когда функция вызывается как new User(...), происходит следующее:

1. Создаётся новый пустой объект, и он присваивается this.
2. Выполняется код функции. Обычно он модифицирует this, добавляет туда новые свойства.
3. Возвращается значение this.

# -== IU7-Web Bilet #18 ==-

1. Что такое браузер? Какие функции он выполняет? Место в Web-стеке. Современное состояние рынка браузеров.
2. CSS. Основные свойства отображения.
3. Задача на: Nginx/REST/Архитектуру/Дизайн/Верстку/JS. В основном - на клиентский JS.

**1. Что такое браузер? Какие функции он выполняет? Место в Web-стеке. Современное состояние рынка браузеров.**

Браузер – прикладное программное обеспечение для просмотра страниц, содержания веб-документов, компьютерных файлов и их каталогов; управления веб-приложениями; а также для решения других задач.

Какие основные функции:

1. Оперативный выход в сеть Всемирной паутины.

2. Предоставление всех необходимых инструментов для мониторинга интернет-среды.

3. Возможность скачивания улучшений, как в ручном, так и в автоматическом режиме.

4. Периодические обновления для различных операционных систем.

5. Возможность задавать определенные настройки, отвечающие пользовательским требованиям.

6. Умение не только просматривать web-страницы, но и открывать электронные документы, мультимедийные файлы и прочее.

7. Предоставление средств для воспроизведения онлайн-видео со звуком, в том числе потоковых трансляций. В этом случае предполагается установка специализированных кодеков и подключение плагинов.

8. Тщательная обработка информации, вводимой владельцем. Способность корректировки текста в процессе поиска.

В настоящее время существует широкий выбор веб-обозревателей. Каждый браузер имеет свои достоинства, недостатки и индивидуальные возможности. Практически все популярные браузеры распространяются бесплатно или «в комплекте» с другими приложениями, так что каждому пользователю не составит труда сравнить отдельные браузеры и выбрать для себя наиболее приемлемый вариант. При выборе браузера следует уделять внимание на поддержку им утвержденных веб-стандартов, простоту и удобство интерфейса, функциональность и безопасность.

Браузеры уже достигли своего максимального развития. Они все выполняют функции просмотра веб страниц, поддерживают мультимедиа. По моему мнению, в настоящее время развитие браузеров будет направлено на обеспечение конфиденциальности передаваемых данных между сайтом и пользователем.

В связи с разработкой новых вычислительных устройств, использующих явление квантовой суперпозиции (квантовый компьютер) и появлению угрозы расшифровки информации, зашифрованной на основе системы открытого и закрытого ключа, требуется разработка новых видов шифрования. Соответственно новые программы браузеров должны поддерживать новые виды шифрования информации.

**2. CSS. Основные свойства отображения.**

Свойства шрифта:

* font-family- определяет используемый элементом шрифт. Если указать URL(file), то шрифт автоматически установится на компьютер пользователя;
* font-style- стиль шрифта (normal, italic);
* font-variant- варианты отображения шрифта (normal, small-caps);
* font-weight- жирность шрифта (normal, bold, bolder, lighter, значение от 100 до 900);
* font-size- размер шрифта (размер, xx-small, x-small, small, medium, large, x-large, xx[1]large, smaller, larger);
* font - обобщает вышеперечисленные свойства (любая комбинация из вышеперечисленных значений);

Свойства фона и цвета:

* color- цвет элемента (значение);
* background-color- цвет фона элемента (значение);
* background-image- изображение фон (none, URL);
* background-repeat- варианты повторения фонового изображения (repeat, repeat-x, repeat-y, no-repeat);
* background-attachment- возможность прокрутки фонового изображения (scroll, fixed);
* background-position- положение фонового изображения (%ширины%высоты, top, middle, bottom, left, center, right);
* background-обобщает вышеперечисленные свойства (любая комбинация из вышеперечисленных значений);

Свойства блока:

* margin-top- определяет отступ сверху (значение, %, auto);
* margin-right- определяет отступ справа (значение, %, auto);
* margin-bottom- определяет отступ снизу (значение, %, auto);
* margin-left- определяет отступ слева (значение, %, auto);
* margin - обобщает все вышеперечисленные свойства;
* padding-top- отступ от верхнего border'а (значение, %);
* padding-right- отступ от правого border'а (значение, %);
* padding-bottom- отступ от нижнего border'а (значение, %);
* padding-left- отступ от левого border'а (значение, %);
* padding - обобщает все вышеперечисленные свойства;

# -== IU7-Web Bilet #19 ==-

1. Стек OSI/ISO. Структура. Применение. Протоколы.
2. JS и ООП. Реализация ООП в ES5 и ES2015+.

**1. Стек OSI/ISO. Структура. Применение. Протоколы.**

Методологической основой стандартизации в компьютерных сетях является многоуровневый подход к разработке средств сетевого взаимодействия. Модель OSI определяет различные уровни взаимодействия систем и указывает, какие функции должен выполнять каждый уровень. В модели OSI средства взаимодействия делятся на семь уровней: прикладной, представительный, сеансовый, транспортный, сетевой, канальный и физический. На самом верхнем уровне, прикладном, пользователь взаимодействует с приложениями. Самый нижний уровень, физический, обеспечивает обмен сигналами между устройствами. Обмен данными через каналы связи происходит путем перемещения данных с верхнего уровня на нижний, затем транспортировки по линиям связи и, наконец, обратным воспроизведением данных в компьютере клиента в результате их перемещения с нижнего уровня на верхний. Для обеспечения необходимой совместимости на каждом из уровней архитектуры компьютерной сети действуют специальные стандартные протоколы. Коммуникационные протоколы могут быть реализованы как программно, так и аппаратно. Протоколы нижних уровней часто реализуются комбинацией программных и аппаратных средств, а протоколы верхних уровней — как правило, чисто программными средствами.

Уровни:

1. **Физический**Физический уровень получает пакеты данных от вышележащего канального уровня и преобразует их в оптические или электрические сигналы, соответствующие 0 и 1 бинарного потока. Эти сигналы посылаются через среду передачи на приемный узел.  
   GSM (сотовая связь), 802.11 (Wi-Fi), USB, IrDA (инфракрасный порт), Bluetooth
2. **Канальный**Канальный уровень обеспечивает создание, передачу и прием кадров данных. Этот уровень обслуживает запросы сетевого уровня и использует сервис физического уровня для приема и передачи пакетов.  
   PPP, IEEE 802.22, DSL, ARP
3. **Сетевой**Сетевой уровень отвечает за деление пользователей на группы. На этом уровне происходит маршрутизация пакетов на основе преобразования MAC-адресов в сетевые адреса.  
   IPv4, IPv6, IPsec, AppleTalk
4. **Транспортный**Транспортный уровень делит потоки информации на достаточно малые фрагменты (пакеты) для передачи их на сетевой уровень.  
   TCP, UDP, SCTP, PORTS
5. **Сеансовый**Сеансовый уровень отвечает за организацию сеансов обмена данными между оконечными машинами. Протоколы сеансового уровня обычно являются составной частью функций трех верхних уровней модели.  
   RPC, PAP, L2TP
6. **Уровень представления**Уровень представления отвечает за возможность диалога между приложениями на разных машинах. Этот уровень обеспечивает преобразование данных (кодирование, компрессия и т.п.) прикладного уровня в поток информации для транспортного уровня. Протоколы уровня представления обычно являются составной частью функций трех верхних уровней модели.  
   ASCII, EBCDIC
7. **Прикладной**Прикладной уровень отвечает за доступ приложений в сеть. Задачами этого уровня является перенос файлов, обмен почтовыми сообщениями и управление сетью.  
   HTTP, FTP, POP3, WebSocket

**2. JS и ООП. Реализация ООП в ES5 и ES2015+.**

Базовый синтаксис выглядит так:

class MyClass {

// методы класса

constructor() { ... }

method1() { ... }

method2() { ... }

method3() { ... }

...

}

Для создания объекта используется вызов new MyClass(), при этом автоматически вызывается метод класса constructor, внутри которого инициализируется объект.

В JavaScript класс – это разновидность функции (Да-да). Вот что на самом деле делает конструкция class MyClass {...}:

1. Создаёт функцию с именем MyClass, которая становится результатом объявления класса. Код функции берётся из метода constructor (она будет пустой, если такого метода нет).
2. Сохраняет все методы, такие как sayHi, в User.prototype.

При вызове метода объекта new User он будет взят из прототипа. Таким образом, объекты new User имеют доступ к методам класса.

Наследование классов. Чтобы унаследовать от класса: class Child extends Parent.

При переопределении конструктора обязателен вызов конструктора родителя super() в конструкторе Child до обращения к this.

При переопределении другого метода мы можем вызвать super.method() в методе Child для обращения к методу родителя Parent.

Статические методы используются для функциональности, принадлежат классу «в целом», а не относятся к конкретному объекту класса. Наследуются.

Для сокрытия внутреннего интерфейса мы используем защищённые или приватные свойства:

* Защищённые поля имеют префикс \_. Это хорошо известное соглашение, не поддерживаемое на уровне языка. Программисты должны обращаться к полю, начинающемуся с \_, только из его класса и классов, унаследованных от него.
* Приватные поля имеют префикс #. JavaScript гарантирует, что мы можем получить доступ к таким полям только внутри класса.

В настоящее время приватные поля не очень хорошо поддерживаются в браузерах, но можно использовать полифил.

# -== IU7-Web Bilet #20 ==- занято

1. История создания WWW. Основные технологии и принципы.

Тим Бернерс-Ли в 80-х годах разработал для личного пользования программу называвшуюся Enquire, дословно переводимую как «Дознаватель». Она использовала случайные ассоциации для хранения данных. Именно эта программа послужила фундаментом для Всемирной паутины . Это было почти за 9 лет до появления проекта WWW, который был представлен в 1989 году Тимом Бернерсом.

В то время Тим работал в CERN Европейском совете по ядерным исследованиям) и разрабатывал для них внутреннюю сеть. В 1989 году он представил проект, предназначенный для публикации гипертекстовых документов, связанных между собой гиперссылками, что облегчило бы поиск и консолидацию информации для учёных CERN.

Для этой технологии были разработаны такие стандарты: протокол HTTP Hyper Text Transfer Protocol), идентификаторы URL и язык текстовой разметки HTML.

Официальный год рождения сети – **1989** год, хотя данные стандарты Бернерс совершенствовал вплоть до 1993 года.

В **1994** году был образован «Консорциум Всемирной паутины World Wide Web Consortium, W3C», который действует до сих пор.

Технология WWW позволяет создавать ссылки, которые реализуют переходы не только внутри исходного документа, но и на любой другой документ, находящийся на данном компьютере и на любой документ любого компьютера, подключенного к Интернету.

В **1989** году Тим Бернерс-Ли предложил свой проект гипертекстовой системы, согласно которой нажатие на ссылку вызывает переход на требуемый документ или фрагмент документа. В качестве указателей ссылок, то есть объектов, активизация которых вызывает переход на другой документ, могут использоваться на только фрагменты текста, но и графические изображения.

Серверы Интернета, реализующие WWWтехнологию, называются Web-серверами, а документы, реализованные по технологии WWW, называются Web-страницами.

Создание Web-страниц осуществляется с помощью языка разметки гипертекста Hyper Text Markup Language - HTML. Основа используемой в HTML технологии состоит в том, что в обычный текстовый документ вставляются управляющие символы (тэги). В результате текстовый документ в браузере выглядит как Webстраница. Базовым кирпичиком для WWW является компьютер с установленным на нём ВЕБсервером подключённый к сети, то есть к другим компьютерам. ВЕБ-сервер программа, запускаемая на подключённом к сети компьютере, и использующей протокол HTTP для передачи данных. В простейшем виде такая программа получает по сети HTTPзапрос на определённый ресурс, находит соответствующий файл на локальном жёстком диске и отправляет его по сети запросившему компьютеру. Более сложные веб-серверы способны динамически формировать ресурсы в ответ на HTTPзапрос.

Для идентификации ресурсов во Всемирной паутине используются единообразные идентификаторы ресурсов URI Uniform Resource Identifier). Для определения местонахождения ресурсов в сети используются единообразные локаторы ресурсов URL Uniform Resource Locator). Такие URLлокаторы сочетают в себе технологию идентификации URL и систему доменных имён DNS Domain Name System) — доменное имя (или непосредственно IPадрес в числовой записи) входит в состав URL для обозначения компьютера (точнее — одного из его сетевых интерфейсов), который исполняет код нужного веб-сервера.

Для просмотра информации, полученной от веб-сервера, на клиентском компьютере применяется специальная программа — веб-браузер. Основная функция веббраузера — отображение гипертекста.

Гипертекст – это текст, размеченный языком гипертекстовой разметки HTML, после HTMLразметки получившийся гипертекст помещается в файл, такой HTMLфайл является самым распространённым ресурсом Всемирной паутины.

После того, как HTMLфайл становится доступен веб-серверу, его начинают называть «веб-страницей». Набор веб-страниц образует веб-сайт. В гипертекст вебстраниц добавляются гиперссылки. Гиперссылки, основанные на технологии URL, помогают пользователям Всемирной паутины легко перемещаться между ресурсами (файлами) вне зависимости от того, находятся ресурсы на локальном компьютере или на удалённом сервере.

1. JS. Выражения. Операторы.

2 Вопрос.

Ну, погнали. Мы ребята умные(не все) и не будем расписывать на 228 строк что делает оператор плюс (хотя я расскажу). В ОБЩЕМ Я выделю зеленым самое интересное про операторы, что есть в JS.

Математические операторы:

* Сложение +,
* Вычитание -,
* Умножение \*,
* Деление /,
* Взятие остатка от деления %,
* Возведение в степень \*\*.

Возвращаясь к истории про оператор плюс. Оператор плюс позволяет сложить a + b и получить ответ. НО… помимо этого унарный оператор плюс в JS является операцией преобразования типа - он пытается привести значение к числу. Если удачно - будет чиселко, иначе будет бяка.

// Не влияет на числа

let x = 1;

alert( +x ); // 1

let y = -2;

alert( +y ); // -2

*// Преобразует не числа в числа*

*alert( +true ); // 1*

*alert( +"5" ); // 5*

Есть оператор присваивания, есть операторы инкремента и декремента, всякие побитовые аля &, |, ^, ~, <<, >>, >>>. Операторы сравнения >, >=, <, <=. Отдельный интерес представляет оператор сравнения. Двойное равно == старается привести к одинаковому типу, это не строгое сравнение. Тройное равно === - это строгое сравнение. Сл-но 0 == false => true, а 0 === false => false, очевидно.

Условный оператор, есть также тернарный. Есть логические &&, ||, !.

Оператор ?? - возвращает первую часть, если она не null/undefined, иначе правую.

# -== IU7-Web Bilet #21 ==-

1. Технологии толстого клиента. Пример с отображением списка пользователей на Angular/React/Vue/JQuery/VanilaJS (код и рассказ как он работает).

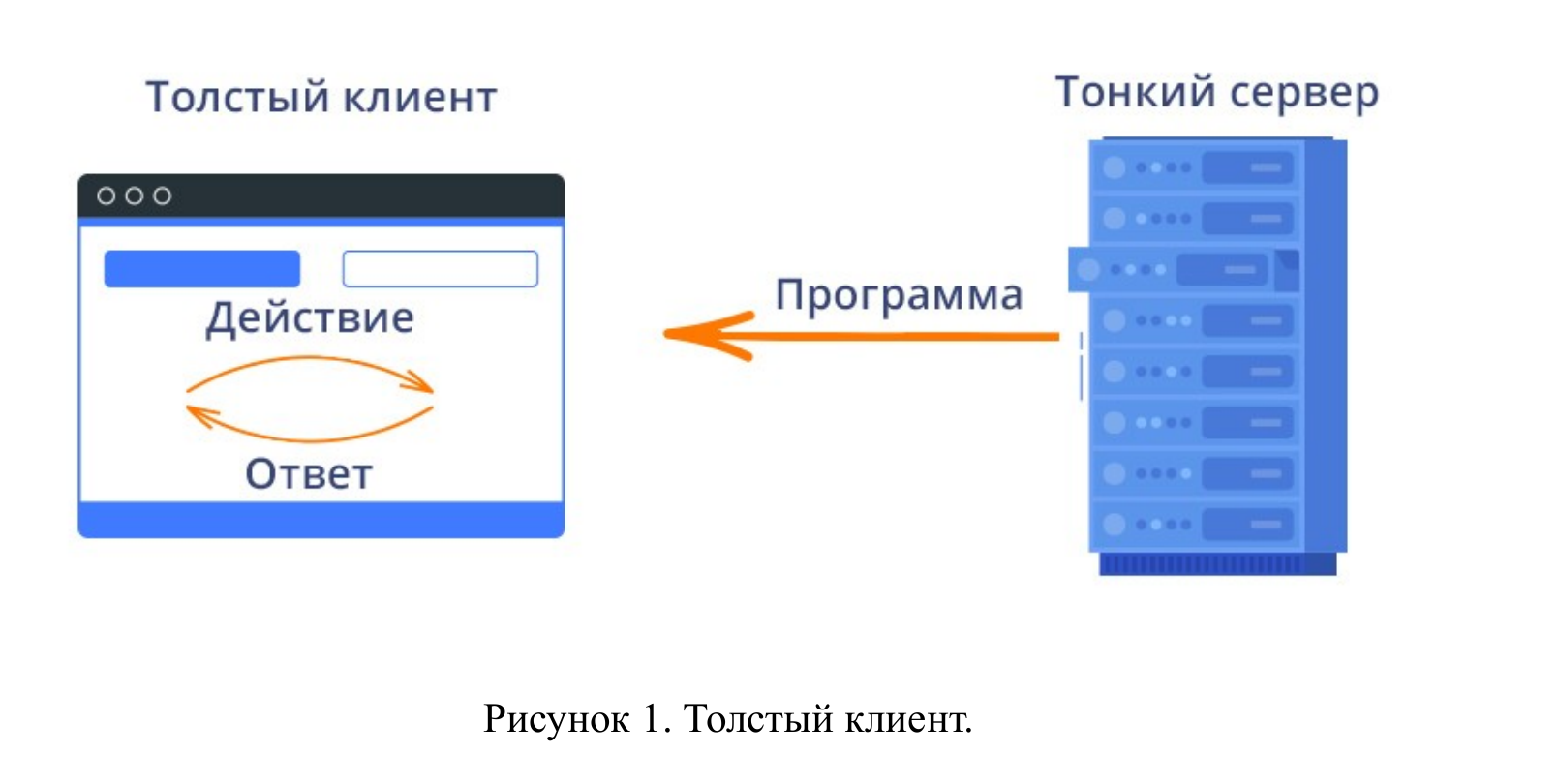
ммсмч

1. Понятие "статического" и "динамического" веб-сервера.

**1.Технологии толстого клиента. Пример с отображением списка пользователей на Angular/React/Vue/JQuery/VanilaJS (код и рассказ как он работает).**

**Толстый клиент** – клиент, который проводит запрашиваемые пользова -телем операции независимо от центрального сервера. Центральный сервер в таком варианте архитектуры может использоваться как хранилище данных, обработка и предоставление которых переносится на рабочую машину клиента.

**Толстым клиентом** является рабочая станция или ПК, которые работают под управлением собственной операционной системы и имеют полный необходимый набор программного обеспечения для реализации задач пользователя.



(+) Достоинства:

* Независимость от сервера
* Режим многопользовательской работы (?)
* Работает при обрыве связи с сервером
* Высокое быстродействие (зависит тока от аппаратных средств клиента)

(-) Недостатки:

* Большой размер дистрибутива
* Часто платформозависимый
* Мб сложный процесс установки, настройки и обновлений
* Мб неактуальность данных

**2. Понятие "статического" и "динамического" веб-сервера.**

# -== IU7-Web Bilet #22 ==-

1. Верстка. Типы Верстки. CSS-фреймворки. Пример организации лейаута (верхнеуровневой структуры) веб-страницы.

**Вёрстка страницы** представляет собой процесс разработки структуры html-документа, результатом которого является веб-страница. Структура веб-страницы определяется соответствующими html-тегами.

**Типы верстки: фиксированная, резиновая и адаптивная.**

**Фиксированная**.

**Фиксированная вёрстка** (также она называется статической) представляет собой такой вид дизайна страницы, в котором размеры указываются точно в пикселях (px) и являются статичными вне зависимости от размеров экрана.

При фиксированной верстке мы делаем контейнер с фиксированными размерами и работаем внутри него. При разных разрешениях размеры контейнера всегда будут одинаковые, и содержимое будет одинаково располагаться на всех устройствах.

В фиксированной верстке мы задаем ширину контейнера в пикселях и, как правило, выравниваем его посередине.

.container {

width: 980px;

margin: 0 auto;

}

Использование такого макета на текущий момент времени оправдано лишь в том случае, если точно известно, что продукт будет применяться только на определенных экранах с четко известным разрешением. В случае открытия на более мелком экране размер веб-страницы будет просто не влезать в размеры окна браузера. А внизу добавится горизонтальная полоса прокрутки.

**Резиновая (тянущаяся).**

При резиновой верстке при увеличении размеров окна, мы растягиваем контент. Здесь размеры принято задавать в процентах от ширины экрана, на котором выполнено открытие страницы.

.container {

width: 100%

}

На широких мониторах сайт расплывётся, хотя, к примеру, на квадратном может быть всё красиво.

**Адаптивная**.

**Адаптивная вёрстка** подстраивает вид страницы под максимальное удобство для пользователя в зависимости от того, на каком устройстве отрыт ресурс. Отображение на различных устройствах может выполняться с перестройкой блоков, с изменением внешнего вида элементов страницы.

Например, при использовании большого экрана, панель навигации будет выглядеть строкой с пунктами меню, в то время как при открытии той же страницы на мобильном телефоне с меньшим разрешением меню свернётся в кнопку в углу.

Под эту верстку обычно создаётся несколько макетов под основные разрешения девайсов. Для реализации подобной вёрстки под каждый размер экрана прописывают свои стили CSS и правила @media.

**CSS-фреймворк**

CSS-фреймворки созданы для упрощения работы верстальщика, исключения ошибок при создании проекта и избавления от скучного монотонного написания одного и того же кода. Профессиональное сообщество разработчиков до сих пор спорит, хорошо или плохо использовать фреймворки. Сложно дать однозначный ответ на этот вопрос, потому что по мере увеличения опыта и с ростом личного профессионализма можно будет просто написать свою личную CSS-библиотеку под себя и свои нужды.

Плюсы CSS-фреймворков:

* Кроссбраузерность
* Возможность создать корректный HTML макет даже не очень опытному специалисту
* Единообразие кода
* Увеличение скорости разработки

Минусы:

* Привязанность к стилю CSS библиотеки
* Избыточный код

Самые популярные: Bootstrap, Kube, Foundation и т. Д.

Я не знаю что они хотят услышать на вопрос про пример организации Layout, поэтому почитайте про потоки блоков, flex и grid. Я пошла спать.

1. **Веб-сервер. Назначение, область применения, типы. Примеры.**

Веб-сервер — сервер, принимающий HTTP запросы от клиентов, обычно веб браузеров, и выдающий им HTTP ответы, как правило, вместе с HTML страницей, изображением, файлом, медиа-потоком или другими данными.

Веб-сервером называют как программное обеспечение, выполняющее функции веб сервера, так и непосредственно компьютер , на котором это программное обеспечение работает.

Клиент, которым обычно является веб-браузер, передаёт веб-серверу запросы на получение ресурсов, обозначенных URLадресами. Ресурсы — это HTML страницы, изображения, файлы, медиа-потоки или другие данные, которые необходимы клиенту. В ответ веб-сервер передаёт клиенту запрошенные данные. Этот обмен происходит по протоколу HTTP.

Различают статические и активные серверы Web. Если страницы сервера содержат только статическую текстовую и мультимедийную информацию, а также гипертекстовые ссылки на другие страницы, то сервер называется статическим. Если страницы web сервера изменяют своё содержимое в зависимости от действий пользователя, то такие серверы называют активными. Статический сервер Web не может служить основой для создания интерактивных приложений с доступом через Интернет, так как он не предусматривает никаких средств ввода и обработки запросов.

Типы:

* статичные веб-сервера - отдают только готовые данные
* динамические - подготавливают контент и/или выполняют дополнительные задачи/скрипты

Самые популярные HTTP серверы:

* Apache – наиболее популярный и распространенный HTTP сервер, используется для Unix систем, но есть версии и для ОС семейства Windows. Данный HTTP сервер является свободным;
* ISS – веб-сервер от компании Microsoft, распространяется беслпатно с операционными система семейства Windows;
* nginx – свободный HTTP сервер, разрабатываемый российским программистом Игорем Сысоевым; стоит отметить, что многие крупные проекты используют NGNIX
* Google Web Server – этот веб-сервер распространяется и поддерживается компанией Гугл, за основу они взяли HTTP сервер Apache и доработали его.
* Cherokee – свободный веб-сервер, особенность которого заключается в том, что управлять им можно только через веб-интерфейс.

# -== IU7-Web Bilet #23 ==-

1. **Передача данных по HTTP. Сериализация. Форматы сериализации (XML, JSON, Protobuf).**

HTTP - текстовый протокол, из-за этого бинарные данные нужно привести к текстовому представлению - сериализовать. Наиболее понятный способ - использование кодировки Base64.

Сериализация — процесс перевода какой-либо структуры данных в последовательность битов. Обратной к операции сериализации является операция десериализации (структуризации) — восстановление начального состояния структуры данных из битовой последовательности. Сериализация используется для передачи объектов по сети и для сохранения их в файлы.

При **XML-сериализации** в поток XML-сериализуются только открытые поля и значения свойств объекта. XML-сериализация не учитывает информацию о типе. Например, если имеется объект Book, который существует в пространстве имен Library, нет никакой гарантии, что он десериализуется в объект аналогичного типа. XML-сериализация не выполняет преобразование методов, индексаторов, закрытых полей или свойств только для чтения (кроме коллекций только для чтения).

**JSON** (англ. JavaScript Object Notation) — текстовый формат обмена данными, основанный на JavaScript. Как и многие другие текстовые форматы, JSON легко читается людьми. За счёт своей лаконичности по сравнению с XML формат JSON может быть более подходящим для сериализации сложных структур. Применяется в веб-приложениях как для обмена данными между браузером и сервером AJAX, так и между серверами (программные HTTPсопряжения).

Использование **Protocol Buffers** — протокол сериализации (передачи) структурированных данных, предложенный Google как эффективная бинарная альтернатива текстовому формату XML. Разработчики сообщают, что Protocol Buffers проще, компактнее и быстрее, чем XML, поскольку осуществляется передача бинарных данных, оптимизированных под минимальный размер сообщения.

1. **HTML. Элементы. Виды элементов. Семантическая вёрстка.**

HTML (от английского HyperText Markup Language) — это язык гипертекстовой разметки страницы. Он используется для того, чтобы дать браузеру понять, как нужно отображать загруженный сайт. Язык состоит из тегов — это своеобразные команды, которые преобразовываются в визуальные объекты в браузере пользователя.

**Элемент HTML** — это основная структурная единица [веб-страницы](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D1%82%D1%80%D0%B0%D0%BD%D0%B8%D1%86%D0%B0), написанная на языке [HTML](https://ru.wikipedia.org/wiki/HTML). Данный элемент вы можете увидеть в исходном коде для всех веб-страниц после задания типа документа на первой строке на странице. DOCTYPE определяет, какую версию (X) HTML эта страница использует. Элементы страницы находятся между открывающим тегом <HTML> и закрывающим </ HTML>. Элемент<html> называется корневым элементом.

HTML элемент определяется начальным тегом, некоторым содержимым и конечным тегом: <tagname>Контент находится здесь...</tagname> HTML **элемент** - все от начального тега до конечного тега.

**Виды элементов**

**https://developer.mozilla.org/ru/docs/Web/HTML/Element**

Все элементы HTML можно сгруппировать по разным типам, в зависимости от их назначения, поведения, соответствия стандарту, версии HTML и др. Это деление довольно условно, не все элементы однозначно попадают в ту или иную категорию.

[**Основные элементы**](https://developer.mozilla.org/ru/docs/Web/HTML/Element#%D0%BE%D1%81%D0%BD%D0%BE%D0%B2%D0%BD%D1%8B%D0%B5_%D1%8D%D0%BB%D0%B5%D0%BC%D0%B5%D0%BD%D1%82%D1%8B)

Основные элементы являются основой любого HTML документа.

[<html>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/html) - представляет собой корень (элемент верхнего уровня) HTML-документа. Все другие элементы должны быть потомками этого элемента (должны находиться внутри него).

[Метаданные документа](https://developer.mozilla.org/ru/docs/Web/HTML/Element#%D0%BC%D0%B5%D1%82%D0%B0%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D0%B5_%D0%B4%D0%BE%D0%BA%D1%83%D0%BC%D0%B5%D0%BD%D1%82%D0%B0)

Метаданные содержат информацию о странице. Они включают в себя информацию о стилях, скрипты и данные, чтобы помочь программному обеспечению (поисковые системы, [браузеры](https://developer.mozilla.org/ru/docs/Glossary/Browser) и т.д.) использовать и отображать страницу. Метаданные для стилей и скрипты могут быть определены на странице или ссылке на другой файл, который имеет информацию.Примеры: [<head>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/head), [<base>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/base) (определяет основной адрес (URL), используемый для всех относительных адресов (URLs) в документе. Может быть только один **<base>** элемент в одном документе), [<link>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/link), [<style>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/style), [<title>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/title), <meta>(представляет типы которые не могут быть описаны тегами ранее).

[Секционирование содержания](https://developer.mozilla.org/ru/docs/Web/HTML/Element#%D1%81%D0%B5%D0%BA%D1%86%D0%B8%D0%BE%D0%BD%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D1%81%D0%BE%D0%B4%D0%B5%D1%80%D0%B6%D0%B0%D0%BD%D0%B8%D1%8F)

Секционирование содержания элементов позволяет разложить содержимое документа на логические части. Используйте секционирование элементов для создания общих черт содержания страницы, включая шапку и заголовочные элементы для обозначения разделов. Примеры: [<header>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/header), [<footer>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/footer), [<h1> (en-US)](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/Heading_Elements), [<h2> (en-US)](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/Heading_Elements), [<h3> (en-US)](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/Heading_Elements), [<h4> (en-US)](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/Heading_Elements), [<h5> (en-US)](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/Heading_Elements), [<h6> (en-US)](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/Heading_Elements), [<section>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/section) …

[Текстовое содержание](https://developer.mozilla.org/ru/docs/Web/HTML/Element#%D1%82%D0%B5%D0%BA%D1%81%D1%82%D0%BE%D0%B2%D0%BE%D0%B5_%D1%81%D0%BE%D0%B4%D0%B5%D1%80%D0%B6%D0%B0%D0%BD%D0%B8%D0%B5)

Используйте текстовые элементы HTML для организации блоков или разделов содержания, размещённые между тегами открытия <body> и закрытия </body>. Важное значение для [accessibility](https://developer.mozilla.org/ru/docs/Glossary/Accessibility) и [SEO](https://developer.mozilla.org/ru/docs/Glossary/SEO), чтобы эти элементы определили цель или структуру этого контента. Примеры: [<blockquote>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/blockquote)(**HTML-элемент <blockquote>** (от англ. *Block Quotation*) указывает на то, что заключённый в нем текст является развёрнутой цитатой.) [<li>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/li)(используется для создания элементов списка.) [<ol>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/ol)(используется для упорядоченного списка — в частности для пронумерованного списка) <p>( представляет собой абзац.)

[Встроенные текстовые семантики](https://developer.mozilla.org/ru/docs/Web/HTML/Element#%D0%B2%D1%81%D1%82%D1%80%D0%BE%D0%B5%D0%BD%D0%BD%D1%8B%D0%B5_%D1%82%D0%B5%D0%BA%D1%81%D1%82%D0%BE%D0%B2%D1%8B%D0%B5_%D1%81%D0%B5%D0%BC%D0%B0%D0%BD%D1%82%D0%B8%D0%BA%D0%B8)

Используйте встроенную в HTML текстовую семантику, чтобы определить смысл, структуру или стиль текста, линий или части текста.Примеры: [<a>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/A) (определяет гиперссылку для перехода на определённое место на странице или на другую страницу в Интернете.) <span>(**HTML-элемент <span>** является основным строковым контейнером для фразового контента, который, по существу, ничего не представляет. Он может использоваться для группировки элементов в целях стилизации) [<ruby>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/ruby)(HTML-элемент**<ruby>** предназначен для добавления небольшой аннотации сверху или снизу от заданного текста. ) ….

[Изображения и мультимедиа](https://developer.mozilla.org/ru/docs/Web/HTML/Element#%D0%B8%D0%B7%D0%BE%D0%B1%D1%80%D0%B0%D0%B6%D0%B5%D0%BD%D0%B8%D1%8F_%D0%B8_%D0%BC%D1%83%D0%BB%D1%8C%D1%82%D0%B8%D0%BC%D0%B5%D0%B4%D0%B8%D0%B0)

HTML позволяет использовать различные мультимедийные ресурсы, такие как изображения, аудио и видео.Примеры:[<img>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/img), [<video>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/video)

[Встроенное содержание](https://developer.mozilla.org/ru/docs/Web/HTML/Element#%D0%B2%D1%81%D1%82%D1%80%D0%BE%D0%B5%D0%BD%D0%BD%D0%BE%D0%B5_%D1%81%D0%BE%D0%B4%D0%B5%D1%80%D0%B6%D0%B0%D0%BD%D0%B8%D0%B5)

Содержание мультимедийных HTML включает в себя другие материалы, даже если они не всегда легко взаимодействуют.Примеры: [<embed>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/embed) (вставляет расширенный контент в выбранное место документа.) [<picture>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/picture) (**HTML-элемент <picture>** служит контейнером для одного или более элементов source и одного элемента img для обеспечения оптимальной версии изображения для различных размеров экрана.)

[Скриптинг](https://developer.mozilla.org/ru/docs/Web/HTML/Element#%D1%81%D0%BA%D1%80%D0%B8%D0%BF%D1%82%D0%B8%D0%BD%D0%B3)

Чтобы создавать динамический контент и веб-приложения, HTML поддерживает использование скриптовых языков, наиболее известным является JavaScript. Некоторые элементы поддерживают эту возможность. Примеры: [<canvas>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/canvas)(отрисовка графики через скрипты), [<noscript>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/noscript)(определяет секцию html кода, которая будет вставлена, если в браузере пользователя нет либо отключена поддержка JavaScript.), [<script>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/script)

[Разграничительные правки](https://developer.mozilla.org/ru/docs/Web/HTML/Element#%D1%80%D0%B0%D0%B7%D0%B3%D1%80%D0%B0%D0%BD%D0%B8%D1%87%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D1%8B%D0%B5_%D0%BF%D1%80%D0%B0%D0%B2%D0%BA%D0%B8)

Эти элементы позволяют вам отметить определённые части текста. Примеры: [<del>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/del)(диапазон текста который был удален из док), [<ins>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/ins)(диапазон текста который был добавлен)

[Содержание таблиц](https://developer.mozilla.org/ru/docs/Web/HTML/Element#%D1%81%D0%BE%D0%B4%D0%B5%D1%80%D0%B6%D0%B0%D0%BD%D0%B8%D0%B5_%D1%82%D0%B0%D0%B1%D0%BB%D0%B8%D1%86)

Эти элементы используются для создания и обработки табличных данных. Примеры:[<caption>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/caption)(заголовок таблицы), [<col>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/col)(столбец), [<td>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/td)(ячейка).

[Формы](https://developer.mozilla.org/ru/docs/Web/HTML/Element#%D1%84%D0%BE%D1%80%D0%BC%D1%8B)

HTML содержит некоторое количество элементов, которые используются вместе для создания форм, которые пользователь может заполнить и отправить на сервер. Доступно множество информации по этой теме в [руководстве по HTML формам](https://developer.mozilla.org/ru/docs/Learn/Forms). Примеры:[<button>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/button), [<form>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/form), [<input>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/Input), [<label>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/label).

[Интерактивные элементы](https://developer.mozilla.org/ru/docs/Web/HTML/Element#%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D0%B0%D0%BA%D1%82%D0%B8%D0%B2%D0%BD%D1%8B%D0%B5_%D1%8D%D0%BB%D0%B5%D0%BC%D0%B5%D0%BD%D1%82%D1%8B)

HTML содержит в себе элементы, которые помогают создавать интерактивные объекты пользовательского интерфейса. Примеры: [<details>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/details), [<menu>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/menu), [<dialog>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/dialog)

[Веб-компоненты](https://developer.mozilla.org/ru/docs/Web/HTML/Element#%D0%B2%D0%B5%D0%B1-%D0%BA%D0%BE%D0%BC%D0%BF%D0%BE%D0%BD%D0%B5%D0%BD%D1%82%D1%8B)

Веб-компоненты - технология, относящаяся к HTML и делающая возможным создание и использование пользовательских элементов, как будто они в обычном HTML. Кроме того, вы можете даже создавать свои версии стандарта HTML элементов. Примеры:[<template>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/template), [<slot>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/slot)(является частью набора технологии [Web Components](https://developer.mozilla.org/en-US/docs/Web/Web_Components), является заполнителем внутри веб компонента, который можно заполнить собственной разметкой, которая позволяет создавать отдельные деревья DOM и представлять их вместе.)

**Семантическая вёрстка** — подход к разметке, который опирается не на содержание сайта, а на смысловое предназначение каждого блока и логическую структуру документа.

**Почему семантика важна**

* **Чтобы сделать сайт доступным.** Зрячие пользователи могут без проблем с первого взгляда понять, где какая часть страницы находится — где заголовок, списки или изображения. Для незрячих или частично незрячих всё сложнее. Основной инструмент для просмотра сайтов не браузер, который отрисовывает страницу, а скринридер, который читает текст со страницы вслух.
* **Чтобы сайт был выше в поисковиках.** Компании, которые создают поисковики, не разглашают правила ранжирования, но известно, что наличие семантической разметки страниц помогает поисковым ботам лучше понимать, что находится на странице, и в зависимости от этого ранжировать сайты в поисковой выдаче.
* **Семантика прописана в стандартах.**

Как разметить страницу с точки зрения семантики

Процесс разметки можно разделить на несколько шагов с разной степенью детализации.

1. Крупные смысловые блоки на каждой странице сайта. Теги: <header>, <main>, <footer>.
2. Крупные смысловые разделы в блоках. Теги: <nav>, <section>, <article>, <aside>.
3. Заголовок всего документа и заголовки смысловых разделов. Теги: <h1>-<h6>.
4. Мелкие элементы в смысловых разделах. Списки, таблицы, демо-материалы, параграфы и переносы, формы, цитаты, контактная информация и прогресс.
5. Фразовые элементы. Изображения, ссылки, кнопки, видео, время и мелкие текстовые элементы.

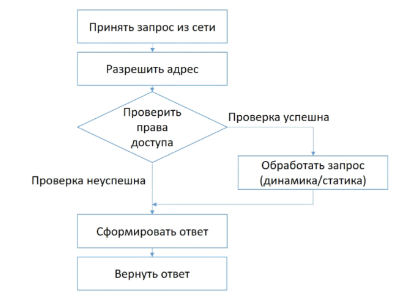
# -== IU7-Web Bilet #24 ==-

1. Алгоритм запуска и работы веб-сервера (на примере Nginx или Apache).

Алгоритм запуска веб-сервера (общий):

1. чтение конфигурации
2. получение порта
3. открытие логов
4. понижение привилегий
5. запуск дочерних процессов/потоков
6. ожидание запросов

Алгоритм работы веб-сервера - общий:

****

Алгоритм работы сервера на примере NNginx.

При каждом поступлении клиентского запроса Nginx определяет, какие блоки конфигурации следует использовать для его обработки.

Блок server — это часть конфигурации Nginx, которая определяет виртуальный сервер, используемый для обработки запросов заданного типа.

Блок location располагается внутри блока server и определяет, как Nginx будет обрабатывать запросы различных ресурсов и URI для родительского сервера.

1. Синтаксический анализ директивы listen: Nginx преобразует все неполные директивы listen, заменяя отсутствующие значения значениями по умолчанию так, что каждый блок оценивается по IP-адресу и порту. При получении запроса, Nginx смотрит IP-адрес и порт запроса. Он сверяет их с директивой listen каждого сервера, создавая список серверных блоков, которые могут обработать данный запрос.
2. Проверка директивы server\_name для поиска наилучшего совпадения со значением Host в заголовке запроса
3. Проверки директив location: сравнивает URI запроса с каждым расположением.
4. иногда определенные директивы в выбранном расположении могут активировать новый поиск расположения (try\_files, rewrite)
5. когда для обслуживания запроса выбирается блок расположения, запрос полностью обрабатывается в этом контексте, начиная с этого момента

Рассмотрим пример:

root /var/www/main;

location / {

try\_files $uri $uri.html $uri/ /fallback/index.html;

}

location /fallback {

root /var/www/another;

}

если мы делаем запрос /blahblah, запрос получит первое расположение. Оно попытается найти файл с именем blahblah в каталоге /var/www/main. Если это не получится сделать, будет выполнен поиск файла с именем blahblah.html. Затем будет выполнен поиск каталога blahblah/ в каталоге /var/www/main. Если все эти попытки закончатся неудачно, будет выполнена переадресация на /fallback/index.html. В этом случае будет активирован другой поиск расположения, который будет перехвачен вторым блоком расположения. Он выдаст файл /var/www/another/fallback/index.html

1. Тенденции развития Web-технологий (HTTP3, WebAssembly, ES2022, PWA)

HTTP/3

HTTP/3 — трансляция транспортного протокола QUIC для прикладного уровня.

Название HTTP/3 официально утвердили лишь недавно, в 17-й версии черновика (draftietf-quic-http-17. Его предложили в конце октября 2018 года, а консенсус был достигнут на встрече IETF 103 в Бангкоке в ноябре.

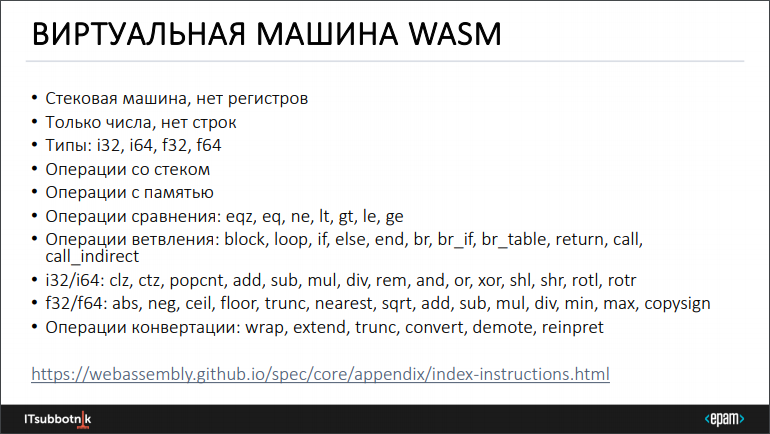
Раньше НТТР/3 был известен как HTTP по QUIC, а до этого — как HTTP/2 по gQUIC, а ещё раньше — SPDY по gQUIC. Но суть в том, что HTTP/3 — просто новый синтаксис HTTP, который работает на протоколе IETF QUIC, мультиплексированном и безопасном транспорте на основе UDP.

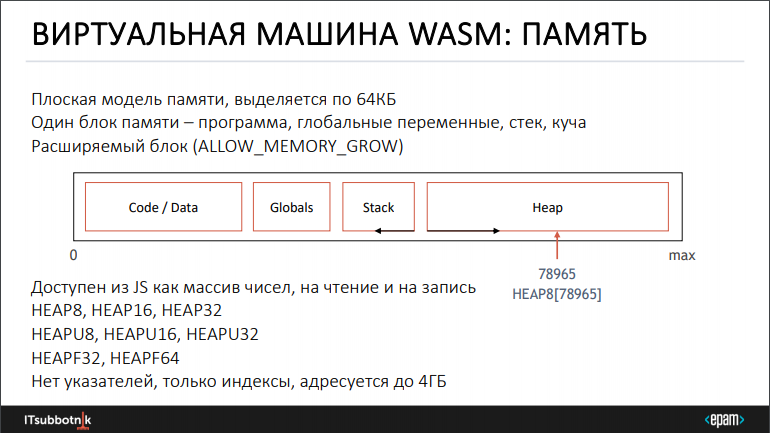
Web Assembly.

Начнём с проблемы, которая решается этой (относительно новой) технологией. Проблема эта — быстро исполнять код в браузере. Быстро — это значит, «быстрее чем JavaScript».

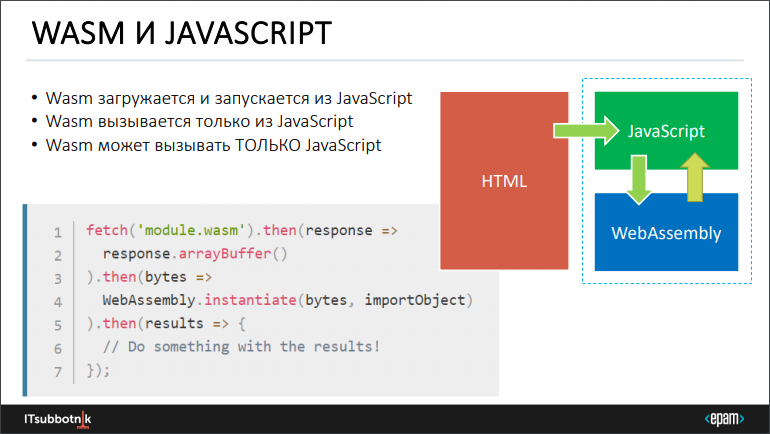
WebAssembly (или Wasm) — это бинарный формат, запускаемый в браузере, виртуальная машина, и результат компиляции с языка высокого уровня.

Wasm это не язык программирования, подобно тому как байт-код Java это не язык программирования, а результат компиляции и запускаемый блок кода.





Вся память WebAssembly полностью доступна из JavaScript, причём как на чтение, так и на запись.



ES2022 (ECMAScript 2022)

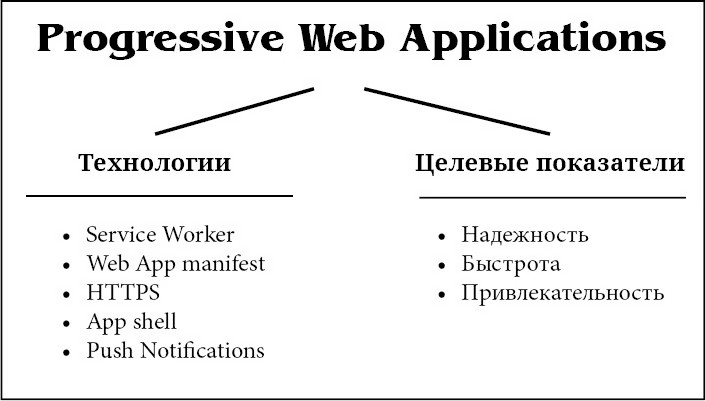
Презентуют новые возможности JavaScript, например

* await верхнего уровня: возможность использовать ключевое слово await на верхнем уровне (top level). Под верхним уровнем в данном случае подразумевается область видимости (scope) модуля.
* метод at() для индексируемых сущностей: предназначен для получения элементов индексируемых сущностей по индексам, в том числе отрицательным (!), по аналогии с тем, как это реализовано, например, в Python. К индексируемым сущностям относятся массивы, типизированные массивы и строки
* 5 предложений для классов (специальные проверки для частных полей, блоки статической инициализации и др.) - ключевое изменение для развития классов в джаваскрипте - вводят публичные/закрытые подя экземлпяров, публичные/закрытые методы экземпляров, публичные статические методы классов

и многое другое

Progressive Web Applications (PWA)

Это философия, набор принципов, практик и инструментов для реализация этих принципов.



* Надежность (Reliable) — приложение загружается и показывается сразу же, вне зависимости от статуса и качества сетевого соединения.
* Быстрота (Fast) — взаимообмен данными по сети происходит быстро, UI плавный и отзывчивый.
* Привлекательность (Engaging) — делает для пользователя опыт работы с приложением комфортным и приятным, побуждая его захотеть пережить его снова

Service Worker

Сердце PWA — Service Worker. Это проксирующий слой между фронтэндом и бэкэндом, находящийся в браузере. Все запросы браузера идут через него. PWA не зависит ни от каких-то фреймворков, это чистый javascript.

HTTPS

PWA требует, чтобы все ресурсы сайта передавались по HTTPS протоколу.

Application Shell

App shell — это просто скелет графического интерфейса, шаблон. Для примера, возьмем средний сайт c хидером, двумя колонками и прочим. Грубо говоря, вырежем из него контент текущей страницы и всю динамическую информацию, оставшаяся статика — app shell. Суть в том, что app shell хранится на клиенте и загружается при запуске приложения, а затем уже в него грузится из сети динамическая информация.

Web App manifest

JSON файл, декларативно определяющий для браузера название приложения, иконку, как будет выглядеть PWA (fullscreen, standalone и др.) и некоторые другие параметры.