| МГТУ | ИУ7 |

ОСНОВЫ РАЗРАБОТКИ

WEB-ПРИЛОЖЕНИЙ

Лектор: Бекасов Д.Е.

Москва, МГТУ им. Н. Э. Баумана.

СПИСОК ВОПРОСОВ

Билет №1.

1. Технологии толстого клиента. Варианты организации персистентного хранилища данных на клиенте (localStorage, IndexedDB).
2. Базы данных. Понятие СУБД. Реляционные и нереляционые базы данных.
3. Варианты организации клиент-серверной архитектуры приложения в веб. Толстый, тонкий и изоморфный клиенты. Определение, принципы.

1. Толстый клиент – клиент, который проводит запрашиваемые пользователем 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.

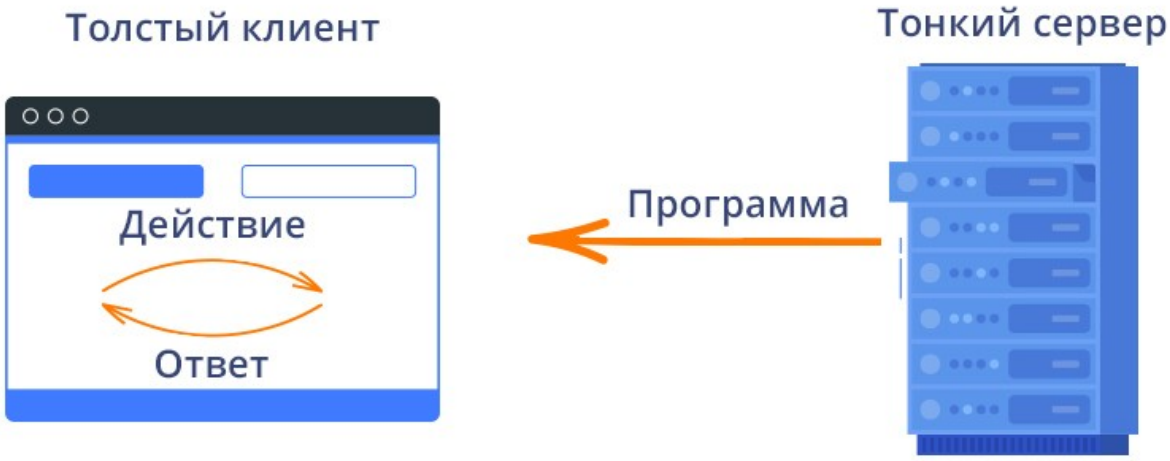

Рисунок 1. Толстый клиент.

Плюсы толстых клиентов:

- ✔ широкая функциональность;
- ✔ многопользовательский режим;
- ✔ работа в режиме оффлайн;
- ✔ высокое быстродействие;
- ✔ минимизация зависимости от дорогих и сложных серверов.

Недостатки:

- ✗ каждая рабочая машина нуждается в постоянном обслуживании;
- ✗ индивидуальное обновление аппаратного обеспечения каждого клиента до уровня приложений, которые будут использоваться;
- ✗ возможность возникновения проблем с удаленным доступом к данным;
- ✗ большие размеры дистрибутивов;
- ✗ зависимость от платформы, для которой клиент был разработан.

Персистентные структуры данных (*англ. persistent data structure*) – это структуры данных, которые при внесении в них каких-то изменений сохраняют все свои предыдущие состояния и доступ к этим состояниям.

Есть несколько **уровней** персистентности:

- **частичная** (*англ. Partial*) – В частично персистентных структурах данных к каждой версии можно делать запросы, но изменять можно только последнюю версию структуры данных.

- **полная** (*англ. Full*) – В полностью персистентных структурах данных можно менять не только последнюю, но и любую версию структур данных, также к любой версии можно делать запросы.
- **конфлюэнтная** (*англ. Confluent*) – Конфлюэнтные структуры данных позволяют объединять две структуры данных в одну (деревья поиска, которые можно сливать).
- **функциональная** (*англ. Functional*) – Функциональные структуры данных полностью персистентны по определению, так как в них запрещаются уничтожающие присваивания, т.е. любой переменной значение может быть присвоено только один раз и изменять значения переменных нельзя. Если структура данных функциональна, то она и конфлюэнтна, если конфлюэнтна, то и полностью персистентна, если полностью персистентна, то и частично персистентна. Однако бывают структуры данных не функциональные, но конфлюэнтные.

LocalStorage представлет собой базу данных на стороне клиента, содержащую пары "ключ-значение". К **преимуществам** локального хранилища относят:

- ✔ Максимально доступный объем хранимых данных определяется браузером, теоретически, может быть ограничен только размерами жесткого диска.
- ✔ Данные хранятся на стороне клиента.
- ✔ Время хранения данных не ограничено.

SessionStorage – Фактически, отличий от **localStorage** не так много и о них мы скажем отдельно. Остается только отметить, что объект **sessionStorage** сохраняет данные в течении пользовательской сессии. Когда браузер пользователя будет за-

крыт данные сохраненные в объекте будут удалены. Данные в хранилище доступны со всех страниц сайта, а не только с той с которой они были сохранены.

Отличия локального хранилища от сеансового – Главное отличие различных типов хранилищ – время хранения данных и их доступность.

- **sessionStorage** хранит данные в рамках одной сессии (посещения, т.е. до закрытия пользователем окна браузера).
- **localStorage** позволяет хранить данные и после прекращения сеанса.
- С точки зрения программирования различие в использовании сеансового и локального типов хранилищ сводится к различию имен объектов, посредством которых осуществляется доступ к ним: sessionStorage и localStorage соответственно.

Ключевые термины и определения

- **cookie** – небольшой фрагмент данных, созданный веб-сервером или веб-страницей и хранимый на компьютере пользователя в виде файла, который веб-клиент (обычно веб-браузер) каждый раз пересылает веб-серверу в HTTP-запросе при попытке открыть страницу соответствующего сайта. Применяется для сохранения данных на стороне пользователя
- **HTML5 Web Storage** – технология, предоставляющая программные интерфейсы для организации хранения данных на стороне клиента.

Перенос вычислительной нагрузки на сторону клиента довольно распространенный способ сбережения вычислительных ресурсов сервера, не говоря уже о том, что в ряде случаев это повышает безопасность и снижает нагрузку на сеть.

В этом контексте умение использовать локальное и сеансовое хранилища является одним из базовых навыков веб - разработчика. Как и с большинством элементов, предоставляемых нам HTML5, сдерживает всеобщее распространение технологии WebStorage не утвержденная спецификация и ограниченная поддержка браузерами.

2. Базы данных – совокупность данных, организованных по определённым правилам, предусматривающим общие принципы описания, хранения и манипулирования данными, независимая от прикладных программ. Эти данные относятся к определённой предметной области и организованы таким образом, что могут быть использованы для решения многих задач многими пользователями.

СУБД – совокупность программ и языковых средств, предназначенных для управления данными в базе данных, ведения базы данных и обеспечения взаимодействия её с прикладными программами.

Реляционная модель – модель типа таблиц и отношений между таблицами. Реляционная модель основана на мощном математическом аппарате теории множеств и математической логики. Эта модель дает возможность ненавигационного манипулирования данными без необходимости знания конкретной физической организации БД во внешней памяти.

Нереляционая модель:

- **Не используется SQL** – Имеется в виду ANSI SQL DML, так как многие базы пытаются использовать query languages похожие на общеизвестный любимый синтаксис, но полностью его реализовать не удалось никому и вряд ли удастся.

- **Неструктурированные (schemaless)** – Смысл таков, что в NoSQL базах в отличие от реляционных структура данных не регламентирована (или слабо типизированна, если проводить аналогии с языками прогаммирования) – в отдельной строке или документе можно добавить произвольное поле без предварительного декларативного изменения структуры всей таблицы. Таким образом, если появляется необходимость поменять модель данных, то единственное достаточное действие – отразить изменение в коде приложения.

3. Толстый клиент (тык сюда)

Тонкий клиент – тип клиента, который переносит задачи по обработке данных на сервер, не используя свои вычислительные возможности для их реализации. Вычислительные ресурсы такого клиента очень ограничены, их должно быть достаточно лишь для запуска необходимого сетевого приложения, используя, например, web-интерфейс.

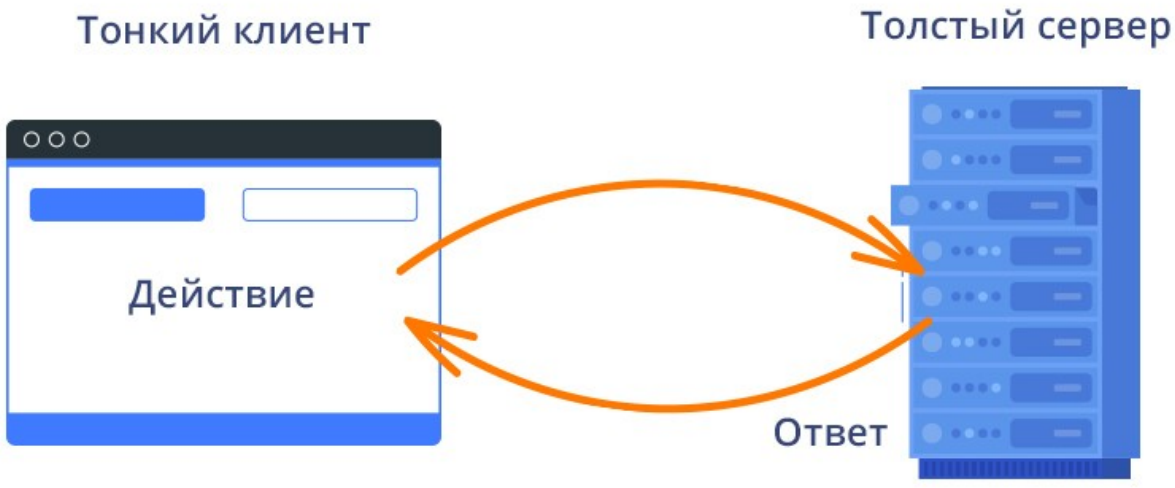

Рисунок 2. Тонкий клиент.

Сравним толстые и тонкие клиенты по основным характеристикам:

- Независимость – толстые клиенты работают независимо от центрального сервера и используют свои ресурсы. Тонкие клиенты почти полностью зависимы от центрального сервера и доступных на нем ресурсов.
- Ресурсозатратность – толстые клиенты используют больше локальных ресурсов, поскольку сами выполняют все функции. Локальные ресурсы тонких клиентов предназначены лишь для создания сессии связи с сервером.
- Сохранение данных – данные пользователей толстых клиентов хранятся локально на рабочей машине.
- Подключение к сети – толстые клиенты могут работать в оффлайн режиме, тонкие клиенты же нуждаются в постоянном доступе к интернету.
- Развертывание – толстые клиенты нуждаются в больших затратах, так как нужно индивидуально обновлять каждую рабочую машину под конкретные задачи пользователя. Тонкие клиенты не такие затратные, потому могут быть использованы очень простые по аппаратному обеспечению ПК. Главной затратой при работе с тонкими клиентами будет высокопроизводительный сервер и его настройка.
- Безопасность – повышенные проблемы с безопасностью могут возникнуть при работе с толстыми клиентами из-за большого риска потери данных на индивидуальных рабочих машинах. Тонкие клиенты считаются более безопасными, так как данные хранятся на сервере.

Примеры использования и приложений – Все пользователи в той или иной мере встречаются с тонкими и толстыми клиентами в процессе работы или просто используя ПК для решения своих задач.

С аппаратной точки зрения тонкий клиент используется в качестве офисных рабочих машин.

С программной стороны, примерами толстых клиентов являются программы и приложения для совместной работы, особенно если они установлены и обрабатываются в конкретном вычислительном устройстве. Некоторые примеры этих приложений включают Microsoft Office 365 и Microsoft Outlook.

Web-браузеры и web-приложения, такие как WordPress, Google Docs и онлайн-игры, являются примерами тонкого клиента. Устройства, используемые для потоковой передачи мультимедиа, такие как Chromecast и Apple TV, установленные с потоковыми приложениями, такими как Netflix или Spotify, технически являются примерами тонких клиентов. Также к тонким клиентам можно отнести и сайты-поисковики Google и Yahoo.

Изоморфные – С помощью React[1], мы можем создавать так называемые “изоморфные” приложения. Если говорить простым языком, “изоморфность” состоит в том, что мы можем использовать один и тот же код как на стороне клиента, так и на стороне сервера. Очевидно, что у такого подхода есть масса преимуществ.

1. Один из главных плюсов – разработан facebook, т. е. поддерживается «глобально».

Билет №2.

1. HTML. Язык разметки гипертекста. История. Структура документа.
2. MVC-фреймворки. Понятие фреймворка. Примеры. Типовая структура.
3. Каскадные таблицы стилей. CSS. История. Структура описания стиля.

1. HTML (*от англ. HyperText Markup Language* – «язык гипертекстовой разметки») – стандартизированный язык разметки веб-страниц во Всемирной паутине. Код HTML интерпретируется браузерами; полученная в результате интерпретации страница отображается на экране монитора компьютера или мобильного устройства.

Язык гипертекстовой разметки HTML был разработан британским учёным Тимом Бернерсом-Ли приблизительно в 1986–1991 годах в стенах ЦЕРНа в Женеве в Швейцарии. HTML создавался как язык для обмена научной и технической документацией, пригодный для использования людьми, не являющимися специалистами в области вёрстки. HTML успешно справлялся с проблемой сложности SGML путём определения небольшого набора структурных и семантических элементов – дескрипторов. Дескрипторы также часто называют «тегами». С помощью HTML можно легко создать относительно простой, но красиво оформленный документ. Помимо упрощения структуры документа, в HTML внесена поддержка гипертекста. Мультимедийные возможности были добавлены позже.

Первым общедоступным описанием HTML был документ «Теги HTML», впервые упомянутый в Интернете Тимом Бернерсом-Ли в конце 1991 года. В нём описываются 18 элементов, составляющих первоначальный, относительно простой дизайн HTML. За исключением тега гиперссылки, на них сильно повлиял SGMLguid, внутренний формат документации, основанный на стандартном обоб-

щенном языке разметки (SGML), в CERN. Одиннадцать из этих элементов всё ещё существуют в HTML 4.

Изначально язык HTML был задуман и создан как средство структурирования и форматирования документов без их привязки к средствам воспроизведения (отображения). В идеале, текст с разметкой HTML должен был без стилистических и структурных искажений воспроизводиться на оборудовании с различной технической оснащённостью (цветной экран современного компьютера, монохромный экран органайзера, ограниченный по размерам экран мобильного телефона или устройства и программы голосового воспроизведения текстов). Однако современное применение HTML очень далеко от его изначальной задачи, с течением времени основная **идея платформонезависимости** языка HTML **была принесена в жертву** современным потребностям в мультим. и граф. Оформлении.

Структура документа – HTML – теговый язык разметки документов. Любой документ на языке HTML представляет собой набор элементов, причём начало и конец каждого элемента обозначается специальными пометками – **тегами**. Элементы могут быть пустыми, то есть не содержащими никакого текста и других данных. В этом случае обычно не указывается закрывающий тег (*например, тег переноса строки
 – одиночный и закрывать его не нужно*) . Кроме того, элементы могут иметь атрибуты, определяющие какие-либо их свойства (например, атрибут href=" у ссылки). Атрибуты указываются в открывающем теге. Пример фрагментов HTML-документа:

Листинг 1. Пример HTML.

```
1. <!DOCTYPE html>
2. <html>
3.    <head>
4.       <meta charset="utf-8" />
```

```
5.       <title>HTML Document</title>
6.    </head>
7.    <body>
8.       <p>
9.          <b>
10.                Этот текст будет полужирным, <i>а этот – ещё и
   курсивным</i>.
11.           </b>
12.        </p>
13.    </body>
14. </html>
```

2. MVC – это шаблон программирования, который позволяет разделить логику приложения на три части:

- **Model** (*модель*). Получает данные от контроллера, выполняет необходимые операции и передаёт их в вид.
- **View** (*вид или представление*). Получает данные от модели и выводит их для пользователя.
- **Controller** (*контроллер*). Обрабатывает действия пользователя, проверяет полученные данные и передаёт их модели.

Если же говорить о приложениях, то компоненты будут следующие:

- **Вид** – интерфейс.
- **Контроллер** – обработчик событий, инициируемых пользователем (нажатие на кнопку, переход по ссылке, отправка формы).
- **Модель** – метод, который запускается обработчиком и выполняет все основные операции (получение записей из базы данных, проведение вычислений).

ASP.NET MVC Framework 10 декабря 2007 года Microsoft представила свой вариант реализации MVC для ASP.NET. Он по-прежнему базируется на .aspx, .ascx

и .master файлах, полностью поддерживает аутентификацию на базе форм, роли, кэширование данных, управление состоянием сессий, health monitoring, конфигурирование, архитектуру провайдеров и другое.

С другой стороны, MVC Framework не предполагает использование классических web-форм и web-элементов управления, в нем отсутствуют такие механизмы как обратные вызовы (postbacks) и состояние представления (viewstate). MVC Framework так же предлагает использование URL-mapping[2] и архитектуру REST в качестве модели запросов, что положительно повлияет на поисковую оптимизацию web-проектов.

Структура Шаблон MVC описывает простой способ построения структуры приложения, целью которого является отделение бизнес-логики от пользовательского интерфейса. В результате, приложение легче масштабируется, тестируется, сопровождается и конечно же реализуется.

Пример последовательности работы MVC-приложения:

1. При заходе пользователя на веб-ресурс, скрипт инициализации создает экземпляр приложения и запускает его на выполнение.
2. При этом отображается вид, скажем главной страницы сайта.
3. Приложение получает запрос от пользователя и определяет запрошенные контроллер и действие. В случае главной страницы, выполняется действие по умолчанию (index).
4. Приложение создает экземпляр контроллера и запускает метод действия,

2. При отправке данных с полей формы, в ссылку, по которой происходит переадресация, добавляется их содержимое. Пример – http://example.com/?name=ИМЯ&email=АДРЕС… и т. д.

5. в котором, к примеру, содержаться вызовы модели, считывающие информацию из базы данных.
6. После этого, действие формирует представление с данными, полученными из модели и выводит результат пользователю.

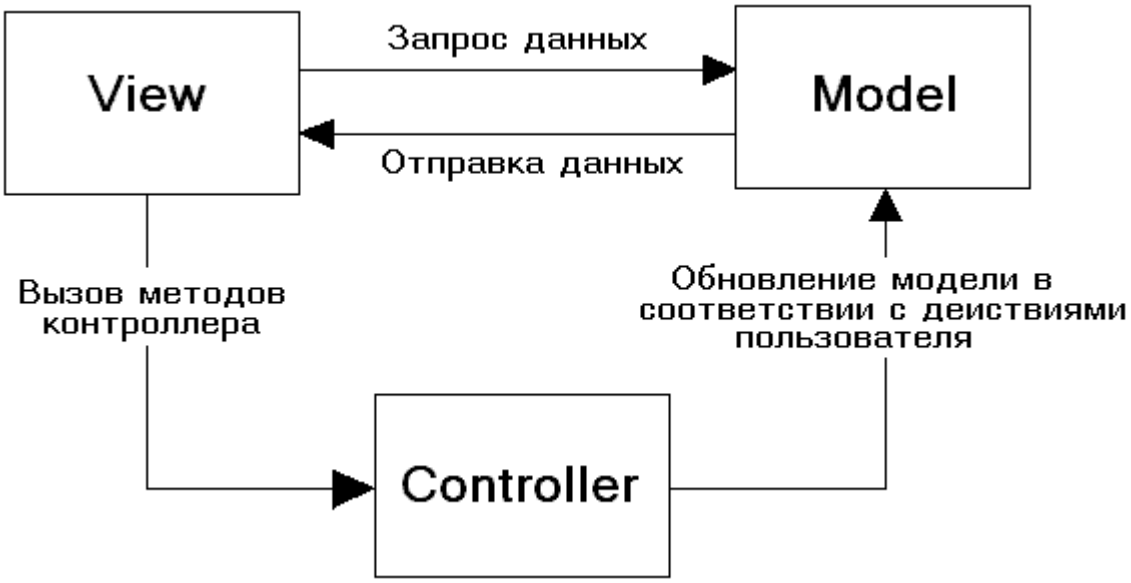

Рисунок 3. Структура MVC

3. CSS Каскадные таблицы стили CSS (Cascading Style Sheets) – стандарт стилей, объявленный консорциумом W3C. Термин каскадные указывает на возможность слияния различных видов стилей и на наследование стилей внутренними тегами от внешних.

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 при-

менение единого стиля оформления для массы схожих документов, а также быстрое изменение этого оформления.

Преимущества:

- ✔ Несколько дизайнов страницы для разных устройств просмотра. Например, на экране дизайн будет рассчитан на большую ширину, во время печати меню не будет выводиться, а на КПК и сотовом телефоне меню будет следовать за содержимым.
- ✔ 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.
- ✔ Простота последующего изменения дизайна. Не нужно править каждую страницу, а достаточно лишь изменить CSS-файл.
- ✔ Дополнительные возможности оформления. Например, с помощью CSS-вёрстки можно сделать блок текста, который остальной текст будет обтекать (например для меню) или сделать так, чтобы меню было всегда видно при прокрутке страницы.

Недостатки:

- ✗ Различное отображение вёрстки в различных браузерах (особенно устаревших), которые по-разному интерпретируют одни и те же данные CSS.
- ✗ Часто встречающаяся необходимость на практике исправлять не только один CSS-файл, но и теги HTML, которые сложным и ненаглядным способом свя-

заны с селекторами CSS, что иногда сводит на нет простоту применения единых файлов стилей и значительно увеличивает время редактирования и тестирования.

История: CSS – одна из широкого спектра технологий, одобренных консорциумом W3C и получивших общее название «стандарты Web». В 1990-х годах стала ясна необходимость стандартизировать Web, создать какие-то единые правила, по которым программисты и веб-дизайнеры проектировали бы сайты. Так появились языки HTML 4.01 и XHTML, и стандарт CSS.

Но, так как в начале 90х различные браузеры имели свои стили для отображения веб-страниц, и HTML развивался очень быстро и был способен удовлетворить все существовавшие на тот момент потребности по оформлению информации, CSS не получил тогда широкого признания.

Термин «каскадные таблицы стилей» был предложен Хоконом Ли в 1994 году. Совместно с Бертом Босом он стал развивать CSS. В отличие от многих существовавших на тот момент языков стиля, CSS использует наследование от родителя к потомку, поэтому разработчик может определить разные стили, основываясь на уже определённых ранее стилях. В середине 90х Консорциум Всемирной паутины (W3C) стал проявлять интерес к CSS, и в декабре 1996 года была издана рекомендация CSS1.

Уровень 1. (CSS1) Рекомендация W3C, принята 17 декабря 1996 года, откорректирована 11 января 1999 года. Среди возможностей, предоставляемых этой рекомендацией:

- Параметры шрифтов. Возможности по заданию гарнитуры и размера шрифта, а также его стиля – обычного, курсивного или полужирного.

- Цвета. Спецификация позволяет определять цвета текста, фона, рамок и других элементов страницы.
- Атрибуты текста. Возможность задавать межсимвольный интервал, расстояние между словами и высоту строки (то есть межстрочные отступы)
- Выравнивание для текста, изображений, таблиц и других элементов.
- Свойства блоков, такие как высота, ширина, внутренние (`padding`) и внешние (`margin`) отступы и рамки. Также в спецификацию входили ограниченные средства по позиционированию элементов, такие как `float` и `clear`.

Уровень 2. (CSS2) Рекомендация W3C, принята 12 мая 1998 года. Основана на CSS1 с сохранением обратной совместимости за несколькими исключениями. Добавление к функциональности:

- Блочная вёрстка. Появились относительное, абсолютное и фиксированное позиционирование. Позволяет управлять размещением элементов по странице без табличной вёрстки.
- Типы носителей. Позволяет устанавливать разные стили для разных носителей (например монитор, принтер, КПК).
- Звуковые таблицы стилей. Определяет голос, громкость и т. д. для звуковых носителей (например для слепых посетителей сайта).
- Страничные носители. Позволяет, например, установить разные стили для элементов на чётных и нечётных страницах при печати.
- Расширенный механизм селекторов.
- Указатели.

- Генерируемое содержимое. Позволяет добавлять содержимое, которого нет в исходном документе, до или после нужного элемента.

В настоящее время W3C больше не поддерживает CSS2 и рекомендует использовать CSS2.1.

Уровень 2, ревизия 1 (CSS2.1) Рекомендация W3C, принята 7 июня 2011 года. CSS2.1 основана на CSS2. Кроме исправления ошибок, в новой ревизии изменены некоторые части спецификации, а некоторые и вовсе удалены. \

Уровень 3. (CSS3) Активно разрабатываемая спецификация CSS. Представляет собой формальный язык, реализованный с помощью языка разметки. Самая масштабная редакция по сравнению с CSS1, CSS2 и CSS2.1. Главной особенностью CSS3 является возможность создавать анимированные элементы без использования JS, поддержка линейных и радиальных градиентов, теней, сглаживания и прочее.

Преимущественно используется как средство описания и оформления внешнего вида веб-страниц, написанных с помощью языков разметки HTML и XHTML, но может также применяться к любым XML-документам, например, к SVG или XUL. В отличие от предыдущих версий спецификация разбита на модули, разработка и развитие которых идёт независимо. CSS3 основан на CSS2.1, дополняет существующие свойства и значения и добавляет новые.

Нововведения, начиная с малых, вроде закруглённых углов блоков, заканчивая трансформацией (анимацией) и, возможно, введением переменных.

Уровень 4. (CSS4) Разрабатывается W3C с 29 сентября 2011 года. Модули CSS4 построены на основе CSS3 и дополняют их новыми свойствами и значениями. **Все они существуют пока в виде черновиков (working draft)!**

Любое CSS правило состоит из двух частей: CSS селектор, при помощи CSS селекторов мы задаем элементы, к которым хотим применить CSS правила и блок объявлений CSS. Блок объявлений может состоят из одного или нескольких CSS объявлений. Каждое объявление состоит из двух частей: CSS свойство и значение CSS. Так, каждая каскадная таблица стилей CSS состоит из набора CSS правил.

Рисунок 4. Структура CSS.

Приоритет – высший приоритет имеют CSS свойства, которые написаны **ниже** по коду. На самом деле применяются оба CSS свойства, просто вначале цвет ссылок станет красным, а затем синим, в итоге мы увидим синие ссылки.

Билет №3.

1. Классификация серверов по типу организации ввода-вывода. Примеры веб-серверов на каждый тип.
2. CSS. Селекторы. Виды селекторов.
3. JS (ES5). Функции. This. Функция-конструктор.

1. Классификация серверов – по назначению, выполняемым функциям или ролям:

Сервер рабочей группы. Представляет собой систему начального уровня, как правило, однопроцессорный. Небольшие компании и удаленные офисы не имеют выделенного специального помещения и располагают сервер непосредственно в своем офисе. По функциям, такая машина служит для разграничения прав доступа сотрудников к файловым ресурсам, либо служит как емкость для хранения общих данных.

Сервер контроллер домена, Domain Controller server. Необходим в организации с количеством сотрудников более 20 рабочих мест, позволяет централизованно управлять сетевыми и файловыми ресурсами компании, также обычно выполняет роль сервера печати. DC server должен быть уже на порядок качественнее и надежнее в отличии от сервера рабочей группы, иметь возможность масштабирования при росте количества пользователей локальной сети. Производительность его зависит от масштаба компании, обычно это одно-двухпроцессорный узел, под управлением MS Windows Server с настроенной службой каталогов Active Directory.

Прокси Сервер - шлюз в Интернет. В этой роли серверная машина обеспечивает общий доступ в интернет всем (или определенным компьютерам офиса) без-

опасную работу сотрудников в Интернете. В случае, если бизнес компании жестко связан с работой сотрудников во внешней сети, такой шлюз должен быть не только отказоустойчивым, но и достаточно производительным: работа специального программного обеспечения (антивирусных программ, анализ и учет трафика, анализаторы атак и т.п.) может требовать большого количества системных ресурсов и высокоскоростных интерфейсов связи.

Сервер эл. почты. Mail Server. Выделенный узел для обработки почтовых приложений, позволяет централизованно управлять внешней корреспонденцией, внутренней перепиской и документооборотом. Серверные версии антивирусных программ и грамотно настроенные фильтры снизят риск потери или утечки конфиденциальной информации и уменьшат объемы нежелательной почты.

Сервер web приложений. Многие современные компании и организации имеют свой виртуальный офис или магазин в сети Интернет WEB-сайт. В нашем случае от веб сервера, его доступность и отказоустойчивость, возможность противостоять внешним негативным воздействиям, атакам и попыткам взлома, достаточной производительностью для сотни или тысячи одновременно принимаемых запросов из сети. Выделенный узел для веб приложений позволит обеспечить доступ большому количеству посетителей, гарантировать работу сложных, критически важных веб приложений компании.

Популярные веб-сервера:

- **Apache** – примерно 55-60% всех сайтов в Интернете (самый популярный web-сервер в мире);

  Данный веб сервер отличается своей гибкостью в конфигурировании и расширяемостью, т.е. к нему можно подключать внешние модули. На данном

web сервере можно разрабатывать сайты на таких языках программирования как:

- PHP;
- Python;
- Ruby;
- Perl;
- ASP;
- Tcl.

А самое главное, что подключить эти языки довольно просто, всего лишь нужно прописать в конфигурационном файле httpd.conf подключение нужных модулей.

Одной из главных особенностей в Apache является то, что разработчик сайта, *например Вы*, разместив свой сайт на хостинге, можете управлять и изменять настройки данного сервера, без его перезагрузки и без ущерба для других сайтов, которые располагаются на этом сервере, это делается с помощью файла .htaccess.

- **Microsoft-IIS** – примерно 12-14 % всех сайтов в Интернете;

Основными компоненты web сервера IIS являются:

- сама web служба;
- служба FTP, может, кстати, функционировать как самостоятельный сервер, если Вы вдруг хотите настроить ftp, но при этом не использовать web

сервер, ничего страшного в этом нет, устанавливайте и пользуйтесь на здоровье;

- SMTP сервер, также можете использовать его как отдельный почтовый сервер у себя в организации.

- **Nginx** – примерно 10-12% всех сайтов в Интернете.

  Это веб-сервер и почтовый прокси-сервер, разработанный российским программистом, который его активно продвигает, сейчас даже появилась компания Nginx Inc. Заточен под UNIX подобные ОС, но есть реализация на винде.

  - **Главные особенности** Nginx это: простота, быстрота, надежность.

Терминальный сервер. Работу удаленных офисов с обеспечением привычного доступа к рабочим ресурсам посредством сети Интернет или выделенных каналов связи способен обеспечить терминальный сервер. Шифрование передаваемых данных обеспечивает безопасность такого вида связи. Пользователь соединяется через канал связи с сервером, вводит свои учетные данные и попадает на свой виртуальный рабочий стол, или документам. Эта служба удобна тем что важные данные хранятся непосредственно на сервере, и доступ к ним можно получить из любой точки мира.

Сервер баз данных. Database server. Обработка данных, организованных и структурированных согласно определенным правилам и хранимых совместно. Наиболее часто используемые средства управления данными это MS SQL Server, Oracle, Apache, MySql.

Файловый сервер. Предназначен для организации и структурированного хранения данных пользователей с учетом политик безопасности и доступа. Количество пользователей и объем хранимых данных являются определяющими моментами при определении состава такой системы.

Серверы приложений. Для сервера приложений характерны расширенные возможности обработки информации, а взаимодействие с клиентом становится подобным работе приложения. Для некоторых организаций такой комплексный подход к построению сервера приложений облегчает разработку благодаря унификации разрабатываемых моделей и централизации поддержки.

«Беспроводной» сервер. В своей простейшей интерпретации такой компьютер может представлять собой типичный Web-сервер или сервер приложений, который просто знает, как передавать документы, составленные на стандартном для беспроводных устройств языке. Часто в качестве такого языка выступает Wireless Markup Language (WML). Адаптация Web-сервера для работы в качестве беспроводного сервера, способного обрабатывать документы WML-типа, обычно сводится просто к тому, чтобы обучить сервер распознаванию этих документов. Web-серверу требуется только сообщить клиенту, что документ составлен в формате для беспроводных устройств, и на этом его работа заканчивается.

Брандмауэры, файрволлы. Защитный экран от вредоносных воздействий из интернета, стена в одну строну пропускает исходящие данные, а в обратную (на прием) уже анализирует что именно поступает в сеть, определяя вредоносные данные, отсеивает их из общего потока входящей информации.

Прокси-серверы можно сконфигурировать так, что они будут принимать или отвергать определенные типы сетевых запросов, поступающие как из локальной

сети, так и из Интернета. В такой конфигурации прокси-сервер становится межсетевым экраном – брандмауэром. Брандмауэр представляет собой средство обеспечения безопасности – осматривать каждый фрагмент данных, который пытается пересечь сеть.

Серверы DHCP. В настоящее время во многих локальных сетях также используется протокол TCP/IP. IP-адрес компьютерам можно присваивать вручную, или же на одной из машин запускается так называемый сервер DHCP (Dynamic Host Configuration Protocol), который автоматически присваивает IP-адрес каждой локальной машине. Основное преимущество сервера DHCP – свобода изменения конфигурации локальной сети при ее расширении, добавлении или удалении машин (например, портативных ПК).

Серверы FTP. Подобные серверы, работающие на основе протокола File Transfer Protocol, уже много десятилетий назад стали стандартом де-факто при перемещении файлов в Интернете. FTP-серверы поддерживают работу простых файловых менеджеров – клиентов. Сложные FTP-серверы обеспечивают администратору большие возможности управления в том, что касается прав на подключение и совместного использования файлов, типов разделяемых файлов и их размещения. Конфигурируемые ресурсы, выделяемые ряду соединений с сервером, ограничения на количество передаваемых данных и минимальную скорость передачи и т.п., становятся все более популярными средствами, помогающими повысить безопасность FTP-серверов.

Принт-серверы. Такие серверы позволяют всем подключенным к сети компьютерам распечатывать документы на одном или нескольких общих принтерах. В этом случае отпадает необходимость комплектовать каждый компьютер собственным печатающим устройством. Кроме того, принимая на себя все заботы

о выводе документов на печать, принт-сервер освобождает компьютеры для другой работы. Например, принт-сервер хранит посланные на печать документы на своем жестком диске, выстраивает их в очередь и выводит на принтер в порядке очередности.

Домашний сервер. В связи с тем, что компьютерная техника имеет очень доступную цену, и проникает повсюду, а также современные операционные системы имеют серверные возможности. С их помощью можно предоставлять пользователям других (соседних) компьютеров доступ к данным на жестком диске или к принтеру, а также «делиться» каналом интернета. Кроме того, домашний сервер можно использовать для резервного хранения данных или, сделав его доступным через Интернет, работать с документами на нем с любого ПК, подключенного к глобальной Сети.

2. Селектор – это часть CSS-правила, которая сообщает браузеру, к какому элементу (или элементам) веб-страницы будет применён стиль. Термин селектор может относиться к простому селектору (simple selector), составному селектору (compound selector), сложному селектору (complex selector) или списку селекторов.

К простым селекторам относятся:

- селектор типа
- универсальный селектор
- селекторы атрибутов
- селектор идентификатора
- селектор класса
- псевдо-классы

CSS селектор	Пример	Описание	CSS
`.class`	`.myClass`	Выбирает все элементы с классом myClass (`class="myClass"`).	1
`#id`	`#main`	Выбирает элемент с идентификатором main (`id="main"`).	1
`*`	`*`	Выбор всех элементов	2
`элемент`	`span`	Выбор всех элементов `<span>`.	1
`элемент, элемент`	`div,span`	Выбор всех элементов `<div>` и всех элементов `<span>`.	1
`[атрибут]`	`[title]`	Выбирает все элементы с атрибутом `title`.	2
`[атрибут="значение"]`	`[title="cost"]`	Выбирает все элементы с атрибутом `title`, значение которого в точности совпадает со значением указанным в селекторе (`title="cost"`).	2
`[атрибут~="значение"]`	`[title~="один"]`	Выбирает все элементы с атрибутом `title`, в значении которого (в любом месте) встречается подстрока (в виде отдельного слова) "один" (`title="один и два"`).	2
`[атрибут|="значение"]`	`[lang|="ru"]`	Выбор всех элементов с атрибутом `lang`, значение которого начинается с "ru".	2
`[атрибут^="значение"]`	`a[href^="https"]`	Выбор каждого элемента `<a>` с атрибутом `href`, значение которого начинается с "https".	3
`[атрибут$="значение"]`	`[src$=".png"]`	Выбирает все элементы с атрибутом `src`, значение которого оканчивается на ".png" (`src="some_img.png"`).	3
`[атрибут*="значение"]`	`[title*="один"]`	Выбирает все элементы с атрибутом `title`, в значении которого (в любом месте) встречается подстрока (в виде отдельного слова или его части) "один" (`title="один и два"`).	3

Составной селектор – это последовательность простых селекторов, которые не разделены комбинаторами, т. е. за одним селектором сразу идёт следующий. Он выбирает элемент, который соответствует всем простым селекторам, которые он содержит. Селектор типа или универсальный селектор, входящий в составной селектор, должен быть расположен первым в этой последовательности. В составном селекторе допустим только один селектор типа или универсальный селектор.

Листинг 2. CSS

```
1. span[title].className
2. p.className1.className2#someId:hover
```

Сложный селектор – это последовательность селекторов, которые разделены комбинаторами.

Список селекторов – это селекторы, перечисленные через запятую.

Комбинаторы. Для объединения простых CSS селекторов, используются комбинаторы, которые указывают взаимосвязь между простыми селекторами. Существует несколько различных комбинаторов в CSS2, и один дополнительный в CSS3, когда вы их используете, они меняют характер самого селектора.

Комбинатор	Пример	Описание	CSS
`элемент элемент`	`div span`	Выбор всех элементов `<span>` внутри `<div>`.	1
`элемент>элемент`	`div>span`	Выбирает все дочерние элементы `<span>`, у которых родитель – элемент `<div>`.	2
`элемент+элемент`	`div+p`	Выбирает все элементы `<p>`, которые расположены сразу после элементов `<div>`.	2
`элемент1~элемент2`	`p~ol`	Выбор всех элементов `<ol>`, которым предшествует элемент `<p>`.	3

3. ECMAScript – это встраиваемый расширяемый не имеющий средств ввода-вывода язык программирования, используемый в качестве основы для построения других скриптовых языков. Стандартизирован международной организацией ECMA в спецификации ECMA-262. Расширения языка: JavaScript, JScript и ActionScript.

Функции в ECMAScript являются объектами. Конструктор, с помощью которого они создаются – **`function()`**. Функции, как и любые другие объекты, могут храниться в переменных, объектах и массивах, могут передаваться как аргументы в другие функции и могут возвращаться функциями. Функции, как и любые другие объекты, могут иметь свойства. Существенной специфической чертой функций является то, что они могут быть вызваны. При инициализации функции, используйте функциональную декларацию вместо функциональных выражений.

Листинг 3. JS Функции (1).

```
1. // плохо
2. var sum = function (x, y) {
3.     return x + y;
4. };
5.
6. // хорошо
7. function sum (x, y) {
8.     return x + y;
9. }
```

Имейте в виду, что функциональная декларация функций будет доступна и в области видимости выше, так что не имеет значения, в каком порядке была объявлена эта функция. Как правило, вы всегда должны объявлять функции в глобальной области видимости, при этом вы должны стараться избегать размещения таких функций внутри условных операторов.

Листинг 4. JS Функции (2).

```
1. // плохо
2. if (Math.random() > 0.5) {
3.     sum(1, 3);
4.
5.     function sum (x, y) {
6.         return x + y;
7.     }
8. }
9.
10. // хорошо
11. if (Math.random() > 0.5) {
12.     sum(1, 3);
13. }
14.
15. function sum (x, y) {
16.     return x + y;
17. }
```

`valueOf()` – возвращает значение **`this`**. Если же объект является результатом вызова конструктора объекта расширения, значение `valueOf()` зависит от реализации[Спецификация 26]. Зачастую в качестве возвращаемого значения выступает значение примитивного типа, соответствующее объекту. Как правило, результат данного метода совпадает с результатом `toString()`. Объекты, созданные с помощью конструктора `Date()` – яркий пример, где результаты `toString()` и `valueOf()` не совпадают. Значение **`this`** – это объект «перед точкой», который использовался для вызова метода.

В языке нет классов, однако их можно эмулировать за счёт использования конструкторов. Стрелочные функции особенные: у них нет своего «собственного» **`this`**. Если мы используем **`this`** внутри стрелочной функции, то его значение берётся из внешней «нормальной» функции. Например, здесь `arrow()` использует значение **`this`** из внешнего метода `user.sayHi()`:

Листинг 5. JS this.

```
1. let user = {
2.   firstName: "ИУ7-86",
3.   sayHi() {
4.     let arrow = () => alert(this.firstName);
5.     arrow();
6.   }
7. };
8.
9. user.sayHi(); // ИУ7-86
```

Это является особенностью стрелочных функций. Они полезны, когда мы на самом деле не хотим иметь отдельное значение **this**, а хотим брать его из внешнего контекста.

Билет №4.

1. Основные идеи и принципы современного CI/CD. Инструменты. Докеризация.
2. Технологии толстого клиента. Организация дуплексной связи клиент-сервер (Poling, Long poling, WebSocket).
3. Что такое браузер? Какие функции он выполняет? Место в Web-стеке. Современное состояние рынка браузеров.

1. Концепция непрерывной интеграции и доставки (CI/CD) – основа тестирований. Для тех, кто не знает: CI/CD – концепция, которая реализуется как конвейер, облегчая слияние только что закомиченного кода в основную кодовую базу. Концепция позволяет запускать различные типы тестов на каждом этапе (выполнение интеграционного аспекта) и завершать его запуском с развертыванием закомиченного кода в фактический продукт, который видят конечные пользователи (выполнение доставки). Инструметы – Buddy, GitLab, Jenkins и т. д.

CI/CD необходимы для разработки программного обеспечения с применением Agile-методологии, которая рекомендует использовать автоматическое тестирование для быстрой наладки рабочего программного обеспечения. Автоматическое тестирование дает заинтересованным лицам доступ к вновь созданным функциям и обеспечивает быструю обратную связь.

1. Первый принцип CI/CD: сегрегация ответственности заинтересованных сторон.
2. Второй принцип CI/CD: снижение риска.
3. Третий принцип CI/CD: короткий цикл обратной связи.

Docker – программное обеспечение для автоматизации развёртывания и управления приложениями в средах с поддержкой контейнеризации. Позволяет «упаковать» приложение со всем его окружением и зависимостями в контейнер, который может быть перенесён на любую Linux-систему с поддержкой cgroups в ядре, а также предоставляет среду по управлению контейнерами. Изначально использовал возможности LXC, с 2015 года применял собственную библиотеку, абстрагирующую виртуализационные возможности ядра Linux – libcontainer.

2. Poling (голосование) Опрос может быть определен как метод, который выполняет периодические запросы независимо от данных, которые существуют в передаче. Периодические запросы отправляются синхронно. Клиент периодически отправляет запрос на Сервер. Ответ сервера включает в себя доступные данные или некоторые предупреждения.

Long poling (Длинный опрос), как следует из названия, включает в себя подобную технику, как опрос. Клиент и сервер поддерживают соединение активным, пока не будут получены некоторые данные или не истечет время ожидания. Если соединение потеряно по каким-либо причинам, клиент может начать заново и выполнить последовательный запрос. Длинный опрос – не что иное, как улучшение производительности по сравнению с процессом опроса, но постоянные запросы могут замедлять процесс.

Web Socket. Этот протокол определяет полнодуплексную связь с нуля. Вебсокеты делают шаг вперед в обеспечении функциональности настольных компьютеров в веб-браузерах. Он представляет собой эволюцию, которая долгое время ожидалась в веб-технологии клиент/сервер.

Основные особенности веб-сокетов следующие:

- ✔ Протокол веб-сокетов стандартизирован, что означает, что с помощью этого протокола возможна связь между веб-серверами и клиентами в режиме реального времени.
- ✔ Веб-сокеты превращаются в кроссплатформенный стандарт для связи в реальном времени между клиентом и сервером.
- ✔ Этот стандарт допускает новый вид приложений. С помощью этой технологии предприятия, работающие в режиме реального времени, могут ускорить работу.
- ✔ Самым большим преимуществом Web Socket является то, что он обеспечивает двустороннюю связь (полный дуплекс) по одному TCP-соединению.

3. Браузер, или веб-обозреватель – прикладное программное обеспечение для просмотра 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 (gif, jpeg, png, svg), аудио-видео форматов (mp3, mpeg), текстовых форматов (pdf, djvu) и других файлов.

Состояние рынка браузеров [05.02.21][3]:

- ★ Chrome 63.63%
- ★ Safari 19.37%

3. Источник https://gs.statcounter.com/browser-market-share

- Firefox 3.65%
- Samsung Internet 3.49%
- Edge 3.24%
- Opera 2.16%

Билет №5.

1. Стек TCP/IP. Назначение, структура, применение.
2. Протокол HTTP. Расшифровка, назначение, структура. Формат пакета.
3. CSS. Основные свойства отображения.

1. TCP/IP – Transmission Control Protocol/Internet Protocol (Протокол Управления Передачей Данных/Межсетевой Протокол). Стек TCP/IP – совокупность протоколов организации взаимодействия между структурами и программными компонентами сети; представляет собой программно реализованный набор протоколов межсетевого взаимодействия.

Рисунок 5. OSI.

I-й должен обеспечить интеграцию в составную сеть любой др. сети, независимо от технологии передачи данных этой сети.

II-й должен обеспечить возможность передачи пакетов через составную сеть, используя разумный (оптимальный) на данный момент маршрут.

III-й решает задачу обеспечения надежной передачи данных между источником и адресатом.

IV-й объединяет все сетевые службы и услуги, предоставляемые сетью пользователю.

В TCP/IP достаточно хорошо развит первый уровень, соответствующий 1 и 2 уровням OSI. Второй уровень TCP/IP – IP.

Структура связей протокольных модулей. Рассмотрим потоки данных, проходящие через стек. В случае использования протокола TCP, данные передаются между прикладным процессом и модулем TCP. Типичным прикладным процессом, использующим протокол TCP, является модуль FTP. Стек протоколов в этом случае будет FTP/TCP/IP/Ethernet. При использовании протокола UDP, данные передаются между прикладным процессом и модулем UDP. Например, SNMP пользуется транспортными услугами UDP. Его стек протоколов выглядит так: SNMP/UDP/IP/Ethernet.

Модули TCP, UDP и драйвер Ethernet являются мультиплексорами n x 1. Действуя как мультиплексоры, они переключают несколько входов на один выход. Они также являются демультиплексорами 1 x n. Как демультиплексоры, они переключают один вход на один из многих выходов в соответствии с полем типа в заголовке протокольного блока данных. Другими словами, происходит следующее:

1. Когда Ethernet-кадр попадает в драйвер сетевого интерфейса Ethernet, он может быть направлен либо в модуль ARP, либо в модуль IP. На то, куда должен быть направлен Ethernet-кадр, указывает значение поля типа в заголовке кадра.

2. Если IP-пакет попадает в модуль IP, то содержащиеся в нем данные могут быть переданы либо модулю TCP, либо UDP, что определяется полем "протокол" в заголовке IP-пакета.
3. Если UDP-датаграмма попадает в модуль UDP, то на основании значения поля "порт" в заголовке датаграммы определяется прикладная программа, которой должно быть передано прикладное сообщение.
4. Если TCP-сообщение попадает в модуль TCP, то выбор прикладной программы, которой должно быть передано сообщение, осуществляется на основе значения поля "порт" в заголовке TCP-сообщения.

Мультиплексирование данных в обратную сторону осуществляется довольно просто, так как из каждого модуля существует только один путь вниз. Каждый протокольный модуль добавляет к пакету свой заголовок, на основании которого машина, принявшая пакет, выполняет демультиплексирование.

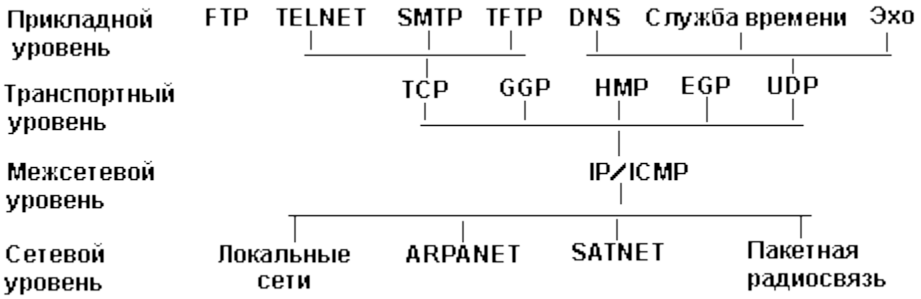

Рисунок 6. Структура взаимосвязей протоколов семейства TCP/IP.

2. HTTP (HyperText Transfer Protocol – протокол передачи гипертекста) – символьно-ориентированный клиент-серверный протокол прикладного уровня без сохранения состояния, используемый сервисом World Wide Web.

Основным объектом манипуляции в HTTP является ресурс, на который указывает URI (Uniform Resource Identifier – уникальный идентификатор ресурса) в запросе клиента. Основными ресурсами являются хранящиеся на сервере файлы, но ими могут быть и другие логические (напр. каталог на сервере) или абстрактные объекты (напр. ISBN). Протокол HTTP позволяет указать способ представления (кодирования) одного и того же ресурса по различным параметрам: mime-типу, языку и т. д. Благодаря этой возможности клиент и веб-сервер могут обмениваться двоичными данными, хотя данный протокол является текстовым.

Структура протокола определяет, что каждое HTTP-сообщение состоит из трёх частей , которые передаются в следующем порядке:

- Стартовая строка (англ. Starting line) – определяет тип сообщения;
- Заголовки (англ. Headers) – характеризуют тело сообщения, параметры передачи и прочие сведения;
- Тело сообщения (англ. Message Body) – непосредственно данные сообщения. Обязательно должно отделяться от заголовков пустой строкой.

Методы:

Метод	Описание
OPTIONS	Используется для определения возможностей веб-сервера или параметров соединения для конкретного ресурса. Предполагается, что запрос клиента может содержать тело сообщения для указания интересующих его сведений. Формат тела и порядок работы с ним в настоящий момент не определён. Сервер пока должен его игнорировать. Аналогичная ситуация и с телом в ответе сервера. Для того чтобы узнать возможности всего сервера, клиент должен указать в URI звёздочку – «*». Запросы «OPTIONS * HTTP/1.1» могут также применяться для проверки работоспособности сервера (аналогично «пингованию») и тестирования на предмет поддержки сервером протокола HTTP версии 1.1. Результат выполнения этого метода не кэшируется.
GET	Используется для запроса содержимого указанного ресурса. С помощью метода GET можно

	также начать какой-либо процесс. В этом случае в тело ответного сообщения следует включить информацию о ходе выполнения процесса. Клиент может передавать параметры выполнения запроса в URI целевого ресурса после символа «?»: GET /path/resource?param1=value1¶m2=value2 HTTP/1.1 Согласно стандарту HTTP, запросы типа GET считаются идемпотентными – многократное повторение одного и того же запроса GET должно приводить к одинаковым результатам (при условии, что сам ресурс не изменился за время между запросами). Это позволяет кэшировать ответы на запросы GET. Кроме обычного метода GET, различают ещё условный GET и частичный GET. Условные запросы GET содержат заголовки If-Modified-Since, If-Match, If-Range и подобные. Частичные GET содержат в запросе Range. Порядок выполнения подобных запросов определён стандартами отдельно.
HEAD	Аналогичен методу GET, за исключением того, что в ответе сервера отсутствует тело. Запрос HEAD обычно применяется для извлечения метаданных, проверки наличия ресурса (валидация URL) и чтобы узнать, не изменился ли он с момента последнего обращения. Заголовки ответа могут кэшироваться. При несовпадении метаданных ресурса с соответствующей информацией в кэше копия ресурса помечается как устаревшая.
POST	Применяется для передачи пользовательских данных заданному ресурсу. Например, в блогах посетители обычно могут вводить свои комментарии к записям в HTML-форму, после чего они передаются серверу методом POST и он помещает их на страницу. При этом передаваемые данные (в примере с блогами – текст комментария) включаются в тело запроса. Аналогично с помощью метода POST обычно загружаются файлы. В отличие от метода GET, метод POST не считается идемпотентным, то есть многократное повторение одних и тех же запросов POST может возвращать разные результаты (например, после каждой отправки комментария будет появляться одна копия этого комментария). При результатах выполнения 200 (Ok) и 204 (No Content) в тело ответа следует включить сообщение об итоге выполнения запроса. Если был создан ресурс, то серверу следует вернуть ответ 201 (Created) с указанием URI нового ресурса в заголовке Location. Сообщение ответа сервера на выполнение метода POST не кэшируется.
PUT	Применяется для загрузки содержимого запроса на указанный в запросе URI. Если по заданному URI не существовало ресурса, то сервер создаёт его и возвращает статус 201 (Created). Если же был изменён ресурс, то сервер возвращает 200 (Ok) или 204 (No Content). Сервер не должен игнорировать некорректные заголовки Content-* передаваемые клиентом вместе с сообщением. Если какой-то из этих заголовков не может быть распознан или не допустим при текущих условиях, то необходимо вернуть код ошибки 501 (Not Implemented). Фундаментальное различие методов POST и PUT заключается в понимании предназначений URI ресурсов. Метод POST предполагает, что по указанному URI будет производиться обработка передаваемого клиентом содержимого. Используя PUT, клиент предполагает, что загружаемое содержимое соответствуют находящемуся по данному URI ресурсу. Сообщения ответов сервера на метод PUT не кэшируются.
PATCH	Аналогично PUT, но применяется только к фрагменту ресурса.
DELETE	Удаляет указанный ресурс.

TRACE	Возвращает полученный запрос так, что клиент может увидеть, что промежуточные сервера добавляют или изменяют в запросе.
LINK	Устанавливает связь указанного ресурса с другими.
UNLINK	Убирает связь указанного ресурса с другими.

Коды состояния:

1xx: Информационные сообщения

Набор этих кодов был введён в HTTP/1.1. Сервер может отправить запрос вида: Expect: 100-continue, что означает, что клиент ещё отправляет оставшуюся часть запроса. Клиенты, работающие с HTTP/1.0 игнорируют данные заголовки.

2xx: Сообщения об успехе

Если клиент получил код из серии 2xx, то запрос ушёл успешно. Самый распространённый вариант - это 200 OK. При GET запросе, сервер отправляет ответ в теле сообщения. Также существуют и другие возможные ответы:

202 Accepted: запрос принят, но может не содержать ресурс в ответе. Это полезно для асинхронных запросов на стороне сервера. Сервер определяет, отправить ресурс или нет.

204 No Content: в теле ответа нет сообщения.

205 Reset Content: указание серверу о сбросе представления документа.

206 Partial Content: ответ содержит только часть контента. В дополнительных заголовках определяется общая длина контента и другая инфа.

3xx: Перенаправление

Своеобразное сообщение клиенту о необходимости совершить ещё одно действие. Самый распространённый вариант применения: перенаправить клиент на другой адрес.

301 Moved Permanently: ресурс теперь можно найти по другому URL адресу.

303 See Other: ресурс временно можно найти по другому URL адресу. Заголовок Location содержит временный URL.

304 Not Modified: сервер определяет, что ресурс не был изменён и клиенту нужно задействовать закэшированную версию ответа. Для проверки идентичности информации используется ETag (хэш Сущности - Enttity Tag);

4xx: Клиентские ошибки

Данный класс сообщений используется сервером, если он решил, что запрос был отправлен с ошибкой. Наиболее распространённый код: 404 Not Found. Это означает, что ресурс не найден на сервере. Другие возможные коды:

400 Bad Request: вопрос был сформирован неверно.

401 Unauthorized: для совершения запроса нужна аутентификация. Информация передаётся через заголовок Authorization.

403 Forbidden: сервер не открыл доступ к ресурсу.

405 Method Not Allowed: неверный HTTP метод был задействован для того, чтобы получить доступ к ресурсу.

409 Conflict: сервер не может до конца обработать запрос, т.к. пытается изменить более новую версию ресурса. Это часто происходит при PUT запросах.

5xx: Ошибки сервера

Ряд кодов, которые используются для определения ошибки сервера при обработке запроса. Самый распространённый: 500 Internal Server Error. Другие варианты:

501 Not Implemented: сервер не поддерживает запрашиваемую функциональность.

503 Service Unavailable: это может случиться, если на сервере произошла ошибка или он перегружен. Обычно в этом случае, сервер не отвечает, а время, данное на ответ, истекает.

3.CSS. Вообще свойств дофига поэтому тык сюда.

CSS-свойства влияющие на цвет и фон

color: CSS-свойство влияющее на цвет текста: заголовка, абзаца, ссылки и т.д. В качестве его значения выступает имя цвета или шестнацатеричный код в RGB-формате.

background-color: – CSS-свойство влияющее на цвет фона заголовка, абзаца, ссылки и т.д. В качестве его значения выступает имя цвета или шестнацатеричный код в RGB-формате.

background-image: – CSS-свойство устанавливающее изображение, в качестве фона фона заголовка, абзаца, ссылки и т.д. Его значением выступает URL (адрес) изображения.

CSS-свойства влияющие на шрифт

font-size: – CSS-свойство влияющее на размер шрифта заголовка, абзаца, ссылки и т.д. В качестве его значения выступает единица измерения: пиксель (px), процент (%), пункт (pt) и т.д.

font-family: – CSS-свойство устанавливающее имя шрифта для заголовка, абзаца, ссылки и т.д. В качестве его значения выступает имя шрифта: Arial, Courier, Impact, Garamond, Georgia, Helvetica, Tahoma, Times (установлен по-умочанию), Verdana и т.д.

font-style: – CSS-свойство влияющее на начертание (стиль) шрифта заголовка, абзаца, ссылки и т.д. В качестве его значения выступает: normal, oblique.

font-weight: – CSS-свойство влияющее на толщину шрифта заголовка, абзаца, ссылки и т.д. В качестве его значения выступет: normal, bold.

CSS-свойства влияющие на текст

text-align: – CSS-свойство влияющее на выравнивание текста в заголовке, абзаце и других блочных элементах. В качестве его значения выступает: left, center, right, justify.

text-indent: – CSS-свойство влияющее на отступ первой строки (красная строка) текста в закаловке, абзаце и других блочных элементах. В качестве его значения выступает единица измерения: пиксель (px), процент (%), пункт (pt) и т. д.

line-height: – CSS-свойство влияющее на расстояние между строками текста (интерлиньяж) заголовка, абзаца, ссылки и т.д. В качестве его значения выступает единица измерения: множитель, пиксель (px), процент (%), пункт (pt) и т. д.

letter-spacing: – CSS-свойство влияющее на расстояние между символами в словах заголовка, абзаца ссылки и т.д. В качестве его значения выступает единица измерения: пиксель (px), процент (%), пункт (pt) и т. д.

word-spacing: – CSS-свойство влияющее на расстояние между словами в тексте заголовка, абзаца, ссылки и т.д. В качестве его значения выступает единица измерения: пиксель (px), процент (%), пункт (pt) и т. д.

text-transform: – CSS-свойство влияющее на вид букв в тексте, делая их прописными или строчными. В качестве его значения выступает: capitalize, lowercase, uppercase.

text-shadow: – CSS-свойство создающее тень текста. В качестве его значения выступает: цвет ось-x ось-y радиус.

CSS-свойства влияющие на границу (обводку)

border-color: – CSS-свойство влияющее на цвет границ заголовка, абзаца, ссылки и т.д. В качестве его значения выступает имя цвета или его шестнадцатеричный код в RGB-формате.

border-width: – CSS-свойство влияющее на толщину границ заголовка, абзаца, ссылки и т.д. В качестве его значения выступает единица измерения: пиксель (px).

border-style: – CSS-свойство влияющее на стиль границ заголовка, абзаца, ссылки и т.д. В качестве его значения выступает: solid, dashed, dotted, double.

CSS-свойства влияющие на внешние и внутренние отступы

margin: – CSS-свойство влияющее на отступ между краем фона одного блочного элемента (заголовки, абзацы и т.д.) и краем фона другого блочного элемента. В качестве его значения выступает единица измерения: пиксель (px), процент (%), пункт (pt) и т. д.

padding: – CSS-свойство влияющее на отступ между краем фона элемента и его содержимым (текстом, изображением). В качестве его значения выступает единица измерения: пиксель (px), процент (%), пункт (pt) и т.д.

Билет №6.

1. CSS3. Основные возможности стандарта (по сравнению с CSS2).
2. Язык Javascript. История. Основные идеи. Область применения. Структура программы.
3. Стек OSI/ISO. Структура. Применение. Протоколы.

1. Уже расписано, тык по гиперссылкам :)

2. JavaScript – мультипарадигменный язык программирования. Поддерживает объектно-ориентированный, императивный и функциональный стили. Является реализацией стандарта ECMAScript (стандарт ECMA-262).

JavaScript обычно используется как встраиваемый язык для программного доступа к объектам приложений. Наиболее широкое применение находит в браузерах как язык сценариев для придания интерактивности веб-страницам.

Основные архитектурные черты: динамическая типизация[4], слабая типизация[5], автоматическое управление памятью, прототипное программирование[6], функции как объекты первого класса[7].

История – В 1992 году компания Nombas начала разработку встраиваемого скриптового языка Cmm (Си-минус-минус), который, по замыслу разработчиков, должен был стать достаточно мощным, чтобы заменить макросы, сохраняя при этом схожесть с Си, чтобы разработчикам не составляло труда изучить его. Глав-

4. Приём, при котором переменная связывается с типом в момент присваивания значения, а не в момент объявления переменной. (Python, JS, Ruby и т. д.)
5. Система типов называется «сильной», если она исключает возможность возникновения ошибки согласования типов времени выполнения. По этому типу C и C++ слабые.
6. Стиль ООП, при котором отсутствует понятие класса, а наследование производится путём клонирования существующего экземпляра объекта – прототипа. (JS)
7. Элементы, которые могут быть переданы как параметр, возвращены из функции, присвоены переменной. (JS)

ным отличием от Си была работа с памятью. В новом языке всё управление памятью осуществлялось автоматически: не было необходимости создавать буферы, объявлять переменные, осуществлять преобразование типов. В остальном языки сильно походили друг на друга: в частности, Cmm поддерживал стандартные функции и операторы Си. Cmm был переименован в ScriptEase, поскольку исходное название звучало слишком негативно, а упоминание в нём Си «отпугивало» людей. На основе этого языка был создан проприетарный продукт CEnvi. В конце ноября 1995 года Nombas разработала версию CEnvi, внедряемую в веб-страницы. Страницы, которые можно было изменять с помощью скриптового языка, получили название Espresso Pages – они демонстрировали использование скриптового языка для создания игры, проверки пользовательского ввода в формы и создания анимации.

Структура языка – Структурно JavaScript можно представить в виде объединения трёх чётко различимых друг от друга частей:

- ядро (ECMAScript),
- объектная модель браузера (Browser Object Model или BOM[8]),
- объектная модель документа (Document Object Model или DOM[9]).

Если рассматривать JavaScript в отличных от браузера окружениях, то объектная модель браузера и объектная модель документа могут не поддерживаться.

Объектную модель документа иногда рассматривают как отдельную от JavaScript сущность, что согласуется с определением DOM как независимого от

8. Дополнительные объекты, предоставляемые браузером (окружением), чтобы работать со всем, кроме документа.
9. Независящий от платформы и языка программный интерфейс, позволяющий программам и скриптам получить доступ к содержимому HTML, XHTML и XML-документов, а также изменять содержимое, структуру и оформление таких документов.

языка интерфейса документа. В противоположность этому ряд авторов находит BOM и DOM тесно взаимосвязанными.

3. Протоколы – связи позволяют структуре на одном хосте взаимодействовать с соответствующей структурой того же уровня на другом хосте. И ещё тык сюда.

Прикладной уровень – верхний уровень модели, обеспечивающий взаимодействие пользовательских приложений с сетью:

- позволяет приложениям использовать сетевые службы:
  - удалённый доступ к файлам и базам данных,
  - пересылка электронной почты;
- отвечает за передачу служебной информации;
- предоставляет приложениям информацию об ошибках;
- формирует запросы к уровню представления.

Протоколы прикладного уровня: HTTP, SMTP, POP3, FTP, и другие.

Сетевой уровень – модели предназначен для определения пути передачи данных. Отвечает за трансляцию логических адресов и имён в физические, определение кратчайших маршрутов, коммутацию и маршрутизацию, отслеживание неполадок и «заторов» в сети.

Протоколы сетевого уровня маршрутизируют данные от источника к получателю. Работающие на этом уровне устройства (маршрутизаторы) условно называют устройствами третьего уровня (по номеру уровня в модели OSI).

Протоколы сетевого уровня: IP/IPv4/IPv6 (Internet Protocol), IPX (Internetwork Packet Exchange, протокол межсетевого обмена), IPsec (Internet Protocol Security).

Протоколы маршрутизации – RIP (Routing Information Protocol), OSPF (Open Shortest Path First).

И ещё ~~много~~ дофига протоколов, я надеюсь не будут всё спрашивать.

Но если что, тык на Wiki.

Билет №7.

1. Основные идеи и возможности HTML5. Область применения.
2. MVC-фреймворки. Понятие фреймворка. Примеры. Типовая структура.
3. Веб-сервер. Назначение, область применения, типы. Примеры.

1. HTML5 – язык для структурирования и представления содержимого всемирной паутины. Это пятая версия HTML. **Цель разработки** HTML5 – улучшение уровня поддержки мультимедиа-технологий с одновременным сохранением обратной совместимости, удобности читаемости кода для человека и простоты анализа для парсеров. Он 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.

Отличия от предыдущего (4.01):

- Изменён синтаксис
- Встраивание SVG и MathML в text/html
- Новые элементы (важные): <audio>, <canvas>, <figure>, <footer>, <header>, <nav>, <source>, <video> и д.р.
- 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)
- Элементы, которые будут исключены (не описал – лишнее…)

Билет №8.

1. Варианты организации клиент-серверной архитектуры приложения в веб. MPA, MPA+AJAX, SPA. Расшифровка, определение, принципы.
2. Структура типового MVC web-приложения на примере одного из фреймворков (Express, ASP, Django, Spring, Rails, etc.).
3. JS и ООП. Реализация ООП в ES5 и ES2015+.

1. SPA – singlepage, MPA – multipage.

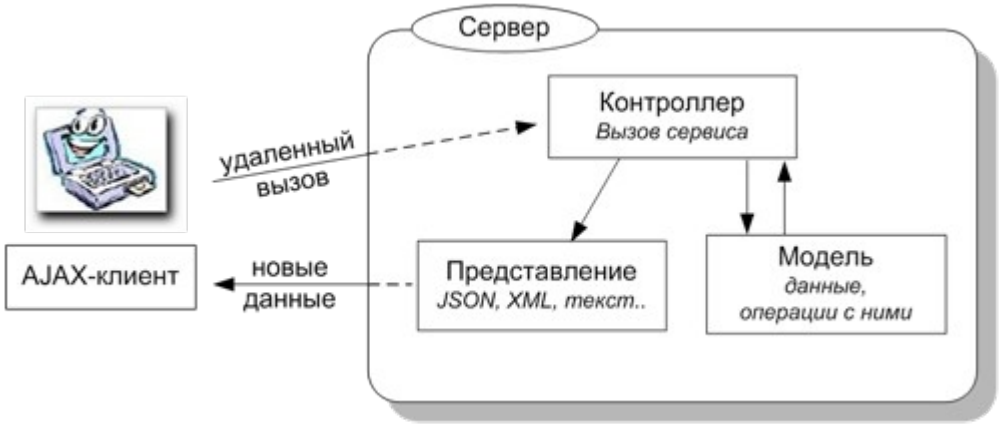

Рисунок 7. AJAX клиент-сервер.

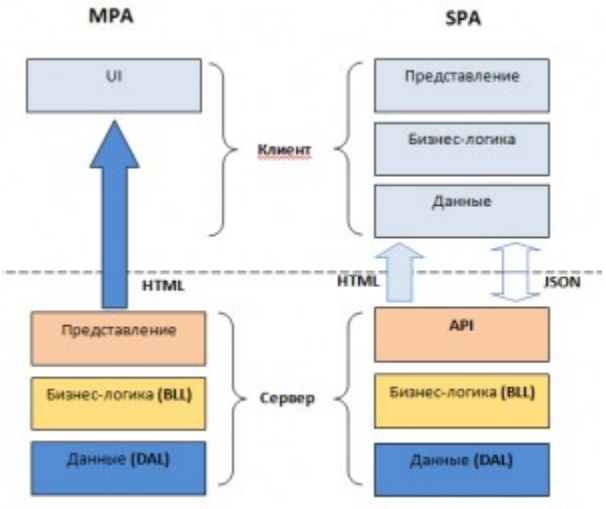

Рисунок 8. SPA и MPA

2. Шаблон проектирования MVC предполагает разделение данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента: Модель, Представление и Контроллер – таким образом, что модификация каждого компонента может осуществляться независимо.

Пример, но он общий:

- **View**, его видит пользователь. Допустим там есть кнопка, при нажатии на которую вызывается **Controller**, где также присутствует валидация – проверка права пользователя.
- **Контроллер** получает пользовательский ввод и обрабатывает данные. Если валидация проходит успешно, данные передаются в **Model**.
- **Модель** проводит с полученными данными необходимые операции, а затем вызывает метод обновления вида.

3. JS ООП – прототипное наследование. Этот простой концепт является гибким и мощным. Он позволяет сделать наследование и поведение сущностями первого класса, так же как и функции являются объектами первого класса в функциональных языках (включая JavaScript).

ES3 –> ES5 (4я версия была неудачной, поэтому 3я была популярна, но даже с приходом 5й, в 2009м году из-за Int.Exp. был тормоз по внедрению) Переход на ES5, вероятно, может немного ускорить процесс разработки и помочь сделать код надежным и оптимизированным. Некоторые из ключевых моментов:

- Методы поиска и управления содержимым массива: indexOf, map, filter, reduce, forEach и т. Д.
- Стандарт представления дат в виде строк (ISO 8601).

- Функции для преобразования объектов в JSON и обратно
- 'use strict' - строгий режим изменяет некоторые скрытые ошибки JavaScript на выдачу ошибок, исправляет некоторые ошибки Javascript для лучшей оптимизации и т. д.

Билет №9.

1. JS. Область определения. Объявление переменных (ES5 и ES2015). Замыкание. Примеры.
2. Унифицированный идентификатор ресурса (URI). URL. URN. Структура, применение.
3. Верстка. Типы Верстки. Дизайн. Виды дизайна. CSS-фреймворки.

1. JS – Переменная представляет собой идентификатор, которому присвоено некое значение. К переменной можно обращаться в программе, работая таким образом с присвоенным ей значением.

Сама по себе переменная в JavaScript не содержит информацию о типе значений, которые будут в ней храниться. Это означает, что записав в переменную, например, строку, позже в неё можно записать число. Такая операция ошибки в программе не вызовет. Именно поэтому JavaScript иногда называют «нетипизированным» языком.

Прежде чем использовать переменную, её нужно объявить с использованием ключевого слова var или let. Если речь идёт о константе, применяется ключевое слово const. Объявить переменную и присвоить ей некое значение можно и не используя эти ключевые слова, но делать так не рекомендуется.

var – до появления стандарта ES2015 использование ключевого слова **var** было единственным способом объявления переменных. [`var a = 0`] Если в этой конструкции опустить var, то значение будет назначено необъявленной переменной. Результат этой операции зависит от того, в каком режиме выполняется программа.

Так, если включён так называемый строгий режим (**strict mode**), подобное вызовет ошибку. Если строгий режим не включён, произойдёт неявное объявление переменной и она будет назначена глобальному объекту. В частности, это означает, что переменная, неявно объявленная таким образом в некоей функции, окажется доступной и после того, как функция завершит работу. Обычно же ожидается, что переменные, объявляемые в функциях, не «выходят» за их пределы. Выглядит это так:

Листинг 6. JS Неявное объявление переменных.

```
1. function notVar() {
2.   bNotVar = 1 //лучше так не делать
3. }
4. notVar()
5. console.log(bNotVar)
```

В консоль попадёт 1, такого поведения от программы обычно никто не ждёт, выражение bNotVar = 1 выглядит не как попытка объявления и инициализации переменной, а как попытка обратиться к переменной, находящейся во внешней по отношению к функции области видимости (это – вполне нормально). Как результат, неявное объявление переменных сбивает с толку того, кто читает код и может приводить к неожиданному поведению программ.

Если в этом примере тело функции переписать в виде var bNotVar = 1, то попытка запустить вышеприведённый фрагмент кода приведёт к появлению сообщения об **ошибке** (его можно увидеть в консоли браузера). Выглядеть оно, например, может так: `Uncaught ReferenceError: bNotVar is not defined`. Смысл его сводится к тому, что программа не может работать с несуществующей переменной.

Переменные, объявленные с помощью ключевого слова var, можно многократно объявлять снова, назначая им новые значения. В одном выражении можно объявить несколько переменных.

Замыкание – это функция, которая запоминает свои внешние переменные и может получить к ним доступ. В некоторых языках это невозможно, или функция должна быть написана специальным образом, чтобы получилось замыкание. Но, как было описано выше, в JavaScript, все функции изначально являются замыканием. То есть, они автоматически запоминают, где были созданы, с помощью скрытого свойства `[[Environment]]` и все они могут получить доступ к внешним переменным.

Листинг 7. JS пример замыкания.

```
1. var createHi = function(name) {
2.    return function() {
3.       alert('Merhaba arkadaş, ' + name);
4.    }
5. }
6. var sayHi = createHi('ИУ7-86');
7. sayHi(); //«Merhaba arkadaş, ИУ7-86»
```

2. Расшифровка аббревиатур

- URL – Uniform Resource Locator (унифицированный определитель местонахождения ресурса)
- URN – Unifrorm Resource Name (унифицированное имя ресурса)
- URI – Uniform Resource Identifier (унифицированный идентификатор ресурса)

Листинг 8. Структура URI.

```
1. URI = [ схема ":" ] иерархическая-часть [ "?" запрос ] [ "#" фрагмент ]
```

В этой записи:

- **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 URI-схемы. 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.

Для парсинга URI, то есть для разложения URI на составные части и их последующей идентификации, удобнее всего использовать систему регулярных выражений[10], доступную почти во всех современных языках программирования.

Листинг 9. Шаблон для парсинга.

```
1. ^(([^:/?#]+):)?(//([^/?#]*))?([^?#]*)(\?([^#]*))?(#(.*))?
2.  12             3   4           5        6   7         8 9
```

Этот шаблон включает в себя 9 обозначенных выше цифрами групп (подробнее о шаблонах и группах см. Регулярные выражения), которые наиболее полно и точно разбирают типичную структуру URI, где:

10. Формальный язык поиска и осуществления манипуляций с подстроками в тексте, основанный на использовании метасимволов. Для поиска используется строка-образец (pattern), состоящая из символов и метасимволов и задающая правило поиска. Для манипуляций с текстом дополнительно задаётся строка замены, которая также может содержать в себе специальные символы.

- группа 2 – схема;
- группа 4 – источник;
- группа 5 – путь;
- группа 7 – запрос;
- группа 9 – фрагмент.

Таким образом, если при помощи данного шаблона разобрать, например, такой типичный идентификатор URI: http://www.ics.uci.edu/pub/ietf/uri/#Related – то 9 вышеуказанных групп шаблона дадут следующие результаты соответственно:

1. http:
2. http
3. //www.ics.uci.edu
4. www.ics.uci.edu
5. /pub/ietf/uri/
6. нет результата
7. нет результата
8. #Related
9. Related

URL: Исторически возник самым первым из понятий и закрепился как синоним термина веб-адрес. URL определяет местонахождение ресурса в сети и способ его (ресурса) извлечения. Это позволяет нам полностью узнать: как, кому и где можно достать требуемый ресурс, вводя понятия схемы, данных авторизации и местонахождения.

URN: Неизменяемая последовательность символов определяющая только имя некоторого ресурса. Смысл URN в том, что им единоразово и уникально именует-

ся какая-либо сущность в рамках конкретного пространства имен (контекста), либо без пространства имен, в общем (что не желательно). Таким образом, URN способен преодолеть недостаток URL связанный с возможным будущим изменением и перемещением ссылок, однако, теперь для того, чтобы знать местонахождение URN ресурса необходимо обращаться к системе разрешения имен URN, в которой он должен быть зарегистрирован.

URI: Это лишь обобщенное понятие (множество) идентификации ресурса, включающее в нашем случае как URL, так и URN, как по отдельности, так и совместно. Т.е. мы можем считать, что: URI = URL или URI = URN или URI = URL + URN

Если коротко – **URI** – это абстракция концепции идентификации, а **URL** и **URN** – это конкретные реализации – полного адреса ресурса и уникального контекстного имени соответственно.

Примеры:

URI,URL = http://www.example.com/some-shit.html[11] (полная идентификация файла, место(Сайт(сервер)), Каталоги, подкаталоги, имя файла, расширение)

URL,URI = http://www.example.com/ (место(Сайт(сервер))

URN,URI = /article/ru/some-shit.html (Название пространства имен и идентификатора в этом пространстве, иначе можно записать %systemroot%/article/ru/some-shit.html)

3. Существует два типа **верстки** веб-страниц:

11. Ссылки написаны от балды, чекать не нужно.

1. блочная, предполагающая формирование страницы из отдельных компонентов (блоков) и таблиц стилей (CSS). Такую верстку ещё называют семантичной;
2. табличная, то есть основанная на применении таблиц, верстка позволяет выравнивать элементы дизайна, создавать рамки, модульные сетки, создавать цветовой фон для страниц.

Принято считать, что табличная верстка гораздо проще блочной в работе. Однако, современные тенденции в веб-дизайне всё же направлены в сторону отказа от таблиц в пользу блочного варианта вёрстки.

В рамках верстки применяют различные виды разметки:

- логическую, то есть ориентированную логически на дальнейшее предназначение элемента страницы;
- физическую, ориентированную на эстетическую гармонию оформления страниц.

Логическая разметка позволяет применять отдельные файлы стилей для верстки, тем самым давая возможность для её использования в печати, на мобильных устройствах и стационарных ПК.

Достоинства и недостатки типов вёрстки

Табличная вёрстка имеет свои недостатки. Так, для вывода содержимого страницы на экран ей требуется больше времени. Именно табличная верстка нередко утяжеляет сайты, делая их загрузку чрезмерно долгой, перегружая браузер, особенно если страницы с табличной версткой оказываются "вложенными" друг в

друга. Также табличная вёрстка не предполагает семантической ориентированности, то есть - смыслового соответствия используемых элементов их назначению.

Блочная вёрстка предполагает размещение слоёв (структурных элементов) на страницах с высокой степенью точности (до 1 пикс). В рамках блочной верстки с применением XHTML и HTML 4 слои создаются с помощью специального тега - .Таблицы в блочной верстке используются только как элемент оформления и несут чисто утилитарную функцию.

При этом блочная верстка позволяет использовать скрипты, заметно разнообразящие её оформительский арсенал. А используя настройку стилей (CSS), можно создавать дизайн любой сложности без лишних затрат времени и сил. Послойная вёрстка позволяет ускорить загрузку страниц, уменьшает "вес" документов в браузере. При этом главной проблемой блочной вёрстки остаётся создание универсального кода, удобного для работы с любыми браузерами и типами экраном. Но и эти сложности сегодня вполне по силам разработчику, знакомому с принципами работы и особенностями различных типов браузеров.

CSS frameworks – Bootstrap, Skeleton и т. д.

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:

- ✔ Кроссбраузерность;
- ✔ Возможность создать корректный HTML макет даже не очень опытному специалисту;
- ✔ Единообразие кода;
- ✔ Увеличение скорости разработки.

Минусы:

- ✗ Привязанность к стилю CSS библиотеки;
- ✗ Избыточный код.

Билет №10.

1. Эволюция HTTP. Протоколы HTTP2 и HTTP3. Основные идеи, решаемые проблемы. Тенденции дальнейшего развития.
2. Клиент в Web. Браузер. Алгоритм исполнения типового GET-запроса на web-ресурс.
3. История создания WWW. Основные технологии и принципы[12].

1. HTTP.

HTTP2. Цели

- Добавить механизмы согласования протокола, клиент и сервер могут использовать HTTP 1.1, 2.0 или, гипотетически, иные, не HTTP-протоколы.
- Поддержать совместимость с многими концепциями HTTP 1.1, например по набору методов доступа (GET, PUT, POST и т. п.), статусным кодам, формату URI, большому количеству заголовков
- Уменьшение задержек доступа для ускорения загрузки страниц, в частности путём:
  - Сжатия данных в заголовках HTTP
  - Использования push-технологий на серверной стороне
  - Конвейеризации запросов
  - Устранения проблемы блокировки «head-of-line» протоколов HTTP 1.0/1.1
  - Мультиплексирования множества запросов в одном соединении TCP
- Сохранение совместимости с широко внедрёнными применениями HTTP, в том числе с веб-браузерами (полноценными и мобильными), API, использу-

12. Сорян, но тут писать уже для даунов (без негатива).

мыми в Интернете, веб-серверами, прокси-серверами, обратными прокси-серверами, сетями доставки контента

HTTP3 – предлагаемый последователь HTTP2, который уже используется в Веб на основе UDP вместо TCP в качестве транспортного протокола. Как и HTTP2, он не объявляет устаревшими предыдущие основные версии протокола. Поддержка HTTP3 была добавлена в Cloudflare и Google Chrome и уже может быть включена в стабильных версиях Chrome и Firefox.

2. В контексте технологии и вычислений клиент – это получатель сервера или тот, который запрашивает конкретную услугу в системе типа сервера. В большинстве случаев клиент располагается на другом компьютерном терминале, доступ к которому возможен по сети.

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

Запрос состоит из двух частей:

1. строка запроса (Request Line)

2. 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:

Листинг 10. GET запрос.

```
1. имя_параметра1=значение_параметра1&
```

```
   имя_параметра2=значение_параметра2&…
2.
3. Разделителем между параметрами служит знак «&».
4.
5. Пример:
6. http://www.example.com/some-shit.html?name=dude&age=69
```

3. История WWW – Мировой Информационной Паутины началась в Марте 1989, когда Тим Бернерс-Ли (Tim Berners Lee) из Европейской Лаборатории Физики Элементарных Частиц (известной как CERN), где работал коллектив исследователей-физиков, предложил новый способ обмена результатами исследований и идеями между организациями. К концу 1990 года исследователи CERN располагали программой просмотра, работающей в текстовом режиме, а также графической программой просмотра для компьютеров семейства NeXT. В 1991 году система WWW стала широко использоваться в CERN.

Первоначально пользователям WWW предоставлялся доступ только к гипертекстовым документам и к статьям телеконференций UseNet. По мере развития проекта добавился интерфейс к другим видам сервиса Internet (WAIS, анонимный FTP, Telnet и Gopher). До 1992 Tim Berners Lee продолжал выступать со своим проектом, до тех пор, пока не появились желающие продолжить работу над этой проблемой. Сотни людей со всего мира приняли участие в разработке этого проекта, одни писали программы и документы для WWW, другие просто рассказывали людям о WWW. Группа пионеров-проектировщиков WWW даже не могла предполагать тогда, что начатое ими дело достигнет таких масштабов.

Итак, 7 августа 1991 года британский ученый Тим Бернерс-Ли впервые опубликовал первые веб-страницы, которые стали результатом его работы над проектом World Wide Web, представлявшим собой метод сохранения знаний с использо-

ванием гипертекстовых документов. Тим разрабатывал свой проект с марта 1989 года и к тому времени, когда он опубликовал первые веб-страницы, он уже создал все необходимое для того чтобы интернет стал реальностью, в том числе первый браузер и сервер.

Тогда Тим, наверное, и представить себе не мог, как изменится мир после его изобретения. Созданный им язык HTML вдохнул жизнь в сеть Интернет, которая до этого существовала только в виде нескольких компьютерных сетей, соединенных между собой кабелем.

В свое время изобретатель Интернета намеренно не стал патентовать свое открытие. Он знал, что в этом случае Интернет никогда не станет всемирной Сетью, массовой и общедоступной.

Билет №11.

1. JS (ES5). Базовые типы данных.
2. HTML. Атрибуты.
3. Передача данных по HTTP. Сериализация. Форматы сериализации (XML, JSON, Protobuf).

2. Атрибуты – зарезервированные слова (как и теги, только без угловых скобок), значения же их могут быть разными. Так же, как и теги, атрибуты со значениями рекомендуется писать маленькими буквами, но разницы нет.

Листинг 11. Значения с атрибутами записываются в формате.

```
1. Атрибут=”значение”
2. lang=”en”
```

Листинг 12. Писать атрибуты всегда нужно внутри открывающего тега, после зарезервированного слова.

```
1. <p align=”center”>Абзац</p>
```

Обычно для одного тега доступно несколько атрибутов. В каком порядке они будут перечислены, неважно.

Универсальные атрибуты – Каждый HTML-тег наделён собственным набором атрибутов. Некоторые атрибуты могут быть доступны для нескольких тегов, другие же могут работать только с одним. Ещё есть группа универсальных (глобальных) атрибутов, которые можно использовать с любым тегом. Вкратце ознакомимся с атрибутами этой категории.

- **accesskey** позволяет задать сочетание клавиш для доступа к определённому объекту страницы. Например, вы можете сделать так, чтобы с помощью комбинации клавиш Alt+1 пользователь переходил по определённой ссылке. Таким образом разработать систему клавишной навигации.

В качестве значения атрибута могут выступать цифры 0-9 или буквы латинского алфавита: `<a acceskey="1" href="http://www.example.com/some-shit.html">Ссылка будет открываться по нажатию сочетания клавиш с единицей</a>`

- **class** позволяет связать тег с заранее заданным с помощью CSS оформлением. Использование атрибута позволяет существенно уменьшить код, ведь вместо того, чтобы повторять ввод одного и того же блока CSS, можно просто ввести имя соответствующего ему класса.
- С помощью **contenteditable** можно разрешить пользователю редактировать любой элемент HTML-страницы: удалять, вставлять, изменять текст. Этот же атрибут даёт возможность редактирование и запретить. Значения имеет всего два: true – правку разрешить, false – запретить.
- При помощи атрибута **contextmenu** вы можете наделить любой элемент документа уникальными пунктами контекстного меню на своё усмотрение. Само меню создаётся в теге <menu>, а атрибуту contextmenu присваивается его идентификатор.
- **dir** определяет направление текста: слева направо (ltr) или справа налево (rtl).
- **draggable** позволяет запретить (false) или разрешить (true) пользователю перетаскивать наделённый этим атрибутом элемент страницы.
- **dropzone** указывает браузеру, что делать с перетаскиваемым элементом: копировать (значение copy), перемещать (move) или создать на него ссылку (link).

- **hidden** – атрибут, позволяющий скрыть содержимое элемента, чтобы оно не отображалось в браузере. Если атрибуту задано значение false, объект отображается, true – скрывается.
- **id** задаёт идентификатор элемента – своего рода имя, которое нужно для простой смены стиля объекта, а также для того, чтобы к нему могли обращаться скрипты. Значением атрибута и будет его имя. Начинаться оно должно обязательно с латинской буквы, и может содержать цифры, буквы всё того же латинского алфавита (большие и маленькие), а также символы дефиса (-) и подчёркивания (_). Русских букв содержать не может.
- **lang** помогает браузеру понять, на каком языке написан контент, и задать ему соответствующий стиль (например, в языках могут использоваться разные кавычки). Значениями выступают коды языков (русский – ru, английский – en и т. п.).
- **spellcheck** включает (true) или отключает (false) проверку правописания. Особенно полезно использовать атрибут в тегах полей форм, куда текст будет вводить пользователь.
- **style** позволяет задать оформление элемента с помощью CSS-кода.
- **tabindex** даёт возможность определить, сколько раз пользователю придётся нажать клавишу Tab, чтобы фокус получил объект с этим атрибутом. Количество нажатий определяет значение атрибута – целое положительное число.
- **title** – всплывающая подсказка, которая появится, если подвести мышку к элементу и на некоторое время оставить её неподвижной. Строка в значении и будет подсказкой. `<a title="Ссылка, ведущая в ***" href="http://www.example.com/some-shit.html">Тык</a>`
- **translate** разрешает (yes) или запрещает (no) перевод содержимого тега.

- **align** задаёт выравнивание элемента. Например, с его помощью можно выровнять текст по левому краю (значение left), по правому краю (right), по центру (center) или по ширине (justify). Для изображений (тег <img>) также доступно выравнивание по верхней границе самого высокого элемента строки (top), по нижней границе (bottom), а значение middle делает так, что средняя линия картинки совпадает с базовой линией строки. (*Но лучше выравнивать с CSS*)

Типы данных. Значение в JavaScript всегда относится к данным определённого типа. Например, это может быть строка или число.

В JavaScript есть 8 основных типов.

- **number** для любых чисел: целочисленных или чисел с плавающей точкой; целочисленные значения ограничены диапазоном $\pm(2^{53}-1)$.
- **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.
- **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.

3.HTTP – широко распространённый протокол передачи данных, изначально предназначенный для передачи гипертекстовых документов (то есть документов, которые могут содержать ссылки, позволяющие организовать переход к другим документам).

Протокол HTTP предполагает использование клиент-серверной структуры передачи данных. Клиентское приложение формирует запрос и отправляет его на сервер, после чего серверное программное обеспечение обрабатывает данный запрос, формирует ответ и передаёт его обратно клиенту. После этого клиентское приложение может продолжить отправлять другие запросы, которые будут обработаны аналогичным образом. Задача, которая традиционно решается с помощью протокола HTTP – обмен данными между пользовательским приложением, осуществляющим доступ к веб-ресурсам (обычно это веб-браузер) и веб-сервером.

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

Сериализация предоставляет несколько полезных возможностей:

- ✔ метод реализации сохраняемости объектов, который более удобен, чем запись их свойств в текстовый файл на диск и повторная сборка объектов чтением файлов;

- ✔ метод осуществления удалённых вызовов процедур, как, например, в SOAP[13];
- ✔ метод распространения объектов, особенно в технологиях компонентно-ориентированного программирования, таких как COM[14] и CORBA[15];
- ✔ метод обнаружения изменений в данных, изменяющихся со временем.

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). Для сериализации всех полей и свойств объекта, как открытых, так и закрытых, используйте **DataContractSerializer** вместо XML-сериализации.

JSON-сериализация. Основная функциональность по работе с **JSON** сосредоточена в пространстве имен **System.Text.Json**.

Ключевым типом является класс **JsonSerializer**, который и позволяет сериализовать объект в **json** и, наоборот, десериализовать код json в объект C#.

---

13. Simple Object Access Protocol – простой протокол доступа к объектам) – протокол обмена структурированными сообщениями в распределённой вычислительной сред. Протокол используется для обмена произвольными сообщениями в формате XML, а не только для вызова процедур.
14. Component Object Model – технологический стандарт от компании Microsoft, предназначенный для создания ПО на основе взаимодействующих компонентов объекта, каждый из которых может использоваться во многих программах одновременно. Стандарт воплощает в себе идеи полиморфизма и инкапсуляции ООП.
15. Common Object Request Broker Architecture – типовая архитектура опосредованных запросов к объектам – технологический стандарт написания распределённых приложений, продвигаемый рабочей группой OMG и соответствующая ему информационная технология. CORBA, детально задает свои форматы сериализации.

Объект, который подвергается десериализации, должен иметь конструктор без параметров. Можно также явным образом определить подобный конструктор в классе.

Сериализации подлежат только публичные свойства объекта (с модификатором **public**).

Почему JSON такой медленный?

Если вы работали с XML, то вы в курсе, что есть два подхода к десериализации – это DOM (Document Object Model) и SAX(Simple API for XML). Напомню, что в случае DOM текст преобразуется в дерево узлов, которое можно исследовать, используя соответствующее API. А SAX парсер работает по-другому – он сканирует документ и генерирует те или иные события, которые обрабатывает код пользователя, реализованный, как правило, в виде функций обратного вызова. Так или иначе, большинство текстовых десериализаторов используют один из этих подходов, либо комбинируют их.

Основной недостаток **DOM** в необходимости строить дерево, а это всегда накладно. Процесс построения такого дерева на этапе десериализации может происходить существенно дольше, чем исполнение прикладных алгоритмов. Но кроме этого, необходимо произвести поиск необходимых полей и преобразовать их во внутренние структуры данных. А эта задача ложится уже на плечи программиста, которую он может реализовать чрезвычайно неэффективно. На самом деле это уже не столь важно, потому что именно построение деревьев DOM съедает основные ресурсы.

Парсеры типа **SAX** гораздо быстрее. По большому счету они проходят текст один раз, вызывая соответствующие хандлеры. Реализовать подобный парсер для **JSON** тривиальная задача, потому как сам **JSON** прост до безобразия, и в этом его

прелесть. Но требует от программиста, его использующего, гораздо больше усилий по извлечению данных. А это – больше работы программисту, больше ошибок и неэффективного кода, что может свести на нет эффективность **SAX**.

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.

По замыслу разработчиков, сначала должна быть описана структура данных, которая затем компилируется в классы. Вместе с классами идёт код их сериализации в компактном формате представления. Чтение и запись данных доступна в высокоуровневых языках программирования – таких как Java, C++ или Python.

Protocol Buffers не предназначен для чтения пользователем и представляет собой двоичный формат. Для десериализации данных необходим отдельный ***.proto****-файл, в котором определяется формат сообщения.*

Билет №12.

1. Паттерны проектирования MVC. Примеры паттернов MVC-семейства. Расшифровка аббревиатуры. Описание архитектуры.
2. CSS. Правила каскадирования и наследования.
3. HTML. Элементы. Виды элементов.

1. MVC – это фундаментальный паттерн, который нашел применение во многих технологиях, дал развитие новым технологиям и каждый день облегчает жизнь разработчикам.

Наиболее распространенные виды MVC-паттерна, это:

- Model-View-Controller
- Model-View-Presenter
- Model-View-View Model

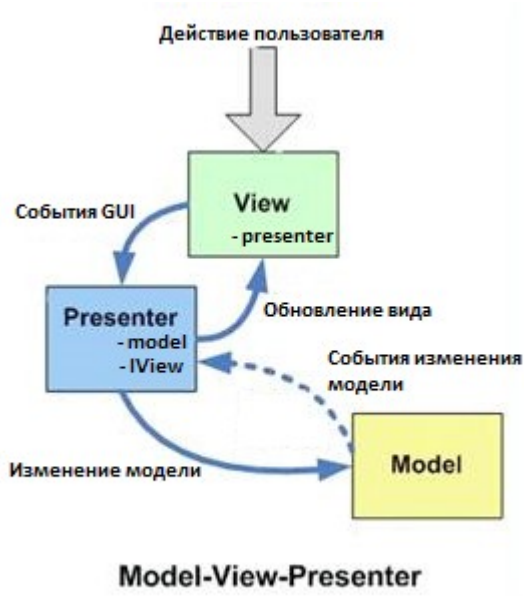

Рисунок 9. MVP.

Данный подход позволяет создавать абстракцию представления. Для этого необходимо выделить интерфейс представления с определенным набором свойств и методов. Презентер, в свою очередь, получает ссылку на реализацию интерфейса, подписывается на события представления и по запросу изменяет модель.

Признаки **презентера**:

- Двухсторонняя коммуникация с представлением;
- Представление взаимодействует напрямую с презентером, путем вызова соответствующих функций или событий экземпляра презентера;
- Презентер взаимодействует с View путем использования специального интерфейса, реализованного представлением;
- Один экземпляр презентера связан с одним отображением.

Реализация:

Каждое представление должно реализовывать соответствующий интерфейс. Интерфейс представления определяет набор функций и событий, необходимых для взаимодействия с пользователем (например, IView.ShowErrorMessage(string msg)). **Презентер** должен иметь ссылку на реализацию соответствующего интерфейса, которую обычно передают в конструкторе.

Логика представления должна иметь ссылку на экземпляр презентера. Все события представления передаются для обработки в презентер и практически никогда не обрабатываются логикой представления (в т.ч. создания других представлений).

Пример использования: **Windows Forms**.

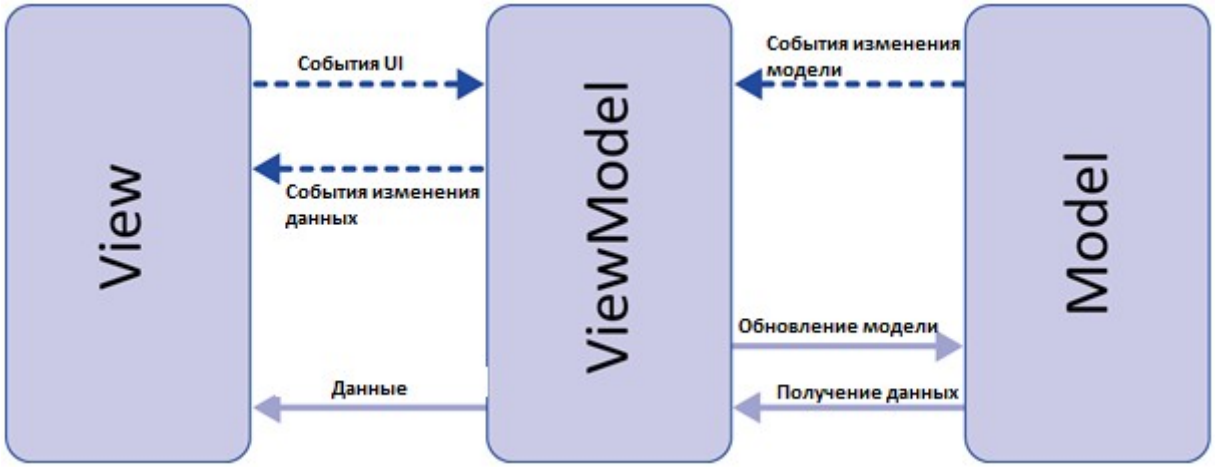

Рисунок 10. MVVM.

Данный подход позволяет связывать элементы представления со свойствами и событиями View-модели. Можно утверждать, что каждый слой этого паттерна не знает о существовании другого слоя.

Признаки View-модели:

- Двухсторонняя коммуникация с представлением;
- View-модель – это абстракция представления. Обычно означает, что свойства представления совпадают со свойствами View-модели / модели
- View-модель не имеет ссылки на интерфейс представления (IView). Изменение состояния View-модели автоматически изменяет представление и наоборот, поскольку используется механизм связывания данных (Bindings)
- Один экземпляр View-модели связан с одним отображением.

Реализация:

При использовании этого паттерна, представление не реализует соответствующий интерфейс (IView).

Представление должно иметь ссылку на источник данных (DataContex), которым в данном случае является View-модель. Элементы представления связаны (Bind) с соответствующими свойствами и событиями View-модели.

В свою очередь, View-модель реализует специальный интерфейс, который используется для автоматического обновления элементов представления. Примером такого интерфейса в WPF может быть INotifyPropertyChanged.

Пример использования: **WPF (Windows Presentation Foundation)**

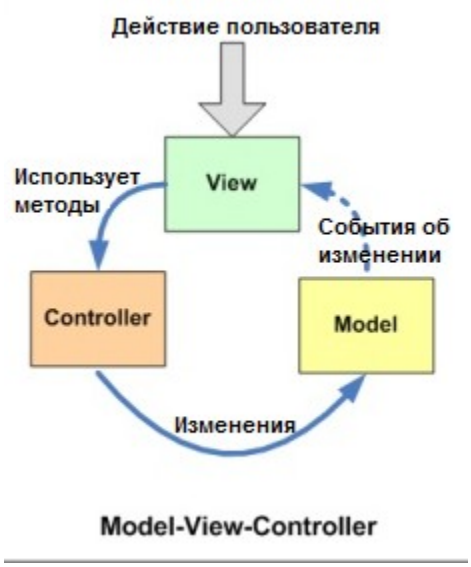

Рисунок 11. MVC

Основная идея этого паттерна в том, что и контроллер и представление зависят от модели, но модель никак не зависит от этих двух компонент.

Признаки контроллера

- Контроллер определяет, какие представление должно быть отображено в данный момент;

- События представления могут повлиять только на контроллер.контроллер может повлиять на модель и определить другое представление.
- Возможно несколько представлений только для одного контроллера;

Реализация:

Контроллер перехватывает событие извне и в соответствии с заложенной в него логикой, реагирует на это событие изменяя Модель, посредством вызова соответствующего метода. После изменения Модель использует событие о том что она изменилась, и все подписанные на это события Представления, получив его, обращаются к Модели за обновленными данными, после чего их и отображают.

Пример использования: **MVC ASP.NET**

Общие правила выбора паттерна (коротко о важном)

MVVM Используется в ситуации, когда возможно связывание данных без необходимости ввода специальных интерфейсов представления (т.е. отсутствует необходимость реализовывать IView);

MVP Используется в ситуации, когда невозможно связывание данных (нельзя использовать Binding);

MVC Используется в ситуации, когда связь между представление и другими частями приложения невозможна (низя использовать MVVM или MVP);

2. Cascading Style Sheets (каскадные таблицы стилей), где одним из ключевых слов выступает «каскад». Под каскадом в данном случае понимается одновременное применение разных стилевых правил к элементам документа – с помощью подключения нескольких стилевых файлов, наследования свойств и других методов. Чтобы в подобной ситуации браузер понимал, какое в итоге правило приме-

нять к элементу, и не возникало конфликтов в поведении разных браузеров, введены некоторые приоритеты.

Ниже приведены приоритеты браузеров, которыми они руководствуются при обработке стилевых правил. Чем выше в списке находится пункт, тем ниже его приоритет, и наоборот.

- Стиль браузера.
- Стиль автора.
- Стиль пользователя.
- Стиль автора с добавлением !important.
- Стиль пользователя с добавлением !important.

Самым низким приоритетом обладает стиль браузера – оформление, которое по умолчанию применяется к элементам веб-страницы браузером. Это оформление можно увидеть в случае «голого» HTML, когда к документу не добавляется никаких стилей.

3. Элемент HTML – это основная структурная единица веб-страницы, написанная на языке 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.

Типы элементов:

Аудио и видео. Элементы, с помощью которых можно добавлять и управлять аудио и видеороликами в браузере.

Документ. Элементы, формирующие структуру документа.

Изображения. Элементы для добавления картинок на страницу.

Объекты. Добавление на страницу апплетов или объектов, которые браузер понимает с помощью плагинов.

Скрипты. Вставка на страницу программных скриптов, обычно на языке JavaScript.

Списки. Нумерованные и маркированные списки.

Ссылки на другие страницы и навигация по сайту.

Таблицы. Создание и управление табличными данными.

Текст. Элементы предназначенные для форматирования текста.

Формы. Интерактивные элементы для взаимодействия с пользователем и отправки данных на сервер для их последующей обработки.

Фреймы разделяют окно браузера на отдельные области, расположенные вплотную друг к другу. В каждую из таких областей загружается самостоятельная веб-страница.

Билет №13.

1. История развития сети Интернет.
2. Технологии толстого клиента. Пример с отображением списка пользователей на JS/JQuery (??? хз чё писать)
3. Понятие "статического" и "динамического" веб-сервера.

1. История Интернета началась с разработки компьютеров в 1950-х годах и появления научных и прикладных концепций глобальных вычислительных сетей почти одновременно в разных странах, в первую очередь в научных и военных лабораториях в США, Великобритании и Франции. (дальше сами -_-)

2. ??? jQuery. Для того чтобы понимать как работает селектор Вам все-же необходимы базовые знания CSS, т.к. именно от принципов CSS отталкивает селектор jQuery:

- $("#header") – получение элемента с id=«header»
- $(«h3») – получить все <h3> элементы
- $(«div#content .photo») – получить все элементы с классом =«photo» которые находятся в элементе div с id=«content»
- $(«ul li») – получить все <li> элементы из списка
- $(«ul li:first») – получить только первый элемент <li> из списка

3. Грубо говоря, сервер может отдавать статическое или динамическое содержимое. «**Статическое**» означает «отдается как есть». Статические веб-сайты делаются проще всего.

«**Динамическое**» означает, что сервер обрабатывает данные или даже генерирует их на лету из базы данных. Это обеспечивает большую гибкость, но технически сложнее в реализации и обслуживании, из-за чего процесс создания сайта сильно усложняется.

На веб-сервере, где сайт хостится, есть сервер приложения, который извлекает содержимое статьи из базы данных, форматирует его, добавляет в HTML-шаблоны и отправляет вам результат.

Существует так много серверов приложений, что довольно трудно предложить какой-то один. Некоторые серверы приложений заточены под определенные категории веб-сайтов, такие как блоги, вики-страницы или интернет-магазины; другие, называемые CMSs (системы управления контентом), более универсальны.

Билет №14.

1. Методы и заголовки протокола HTTP 1.1
2. Клиент-серверная архитектура. Принципы построения. Понятие клиента и сервера в Web.
3. JS. Выражения. Операторы.

1. HTTP 1.1

С помощью URL, мы определяем точное название хоста, с которым хотим общаться, однако какое действие нам нужно совершить, можно сообщить только с помощью HTTP метода. Конечно же существует несколько видов действий, которые мы можем совершить. В HTTP реализованы самые нужные, подходящие под нужды большинства приложений.

Существующие методы:

- **GET**: получить доступ к существующему ресурсу. В **URL** перечислена вся необходимая информация, чтобы сервер смог найти и вернуть в качестве ответа искомый ресурс.
- **POST**: используется для создания нового ресурса. **POST** запрос обычно содержит в себе всю нужную информацию для создания нового ресурса.
- **PUT**: обновить текущий ресурс. **PUT** запрос содержит обновляемые данные.
- **DELETE**: служит для удаления существующего ресурса.

Данные методы самые популярные и чаще всего используются различными инструментами и фрэймворками. В некоторых случаях, **PUT** и **DELETE** запросы отправляются посредством отправки **POST**, в содержании которого указано действие, которое нужно совершить с ресурсом: создать, обновить или удалить.

Также **HTTP** поддерживает и другие методы:

- **HEAD**: аналогичен **GET**. Разница в том, что при данном виде запроса не передаётся сообщение. Сервер получает только заголовки. Используется, к примеру, для того чтобы определить, был ли изменён ресурс.
- **TRACE**: во время передачи запрос проходит через множество точек доступа и прокси серверов, каждый из которых вносит свою информацию: **IP**, **DNS**. С помощью данного метода, можно увидеть всю промежуточную информацию.
- **OPTIONS**: используется для определения возможностей сервера, его параметров и конфигурации для конкретного ресурса.

Заголовки запросов:

request-header = Accept
| Accept-Charset
| Accept-Encoding
| Accept-Language
| Authorization
| Expect
| From
| Host
| If-Match
| If-Modified-Since
| If-None-Match
| If-Range
| If-Unmodified-Since

| Max-Forwards

| Proxy-Authorization

| Range

| Referer

| TE

| User-Agent

2. «Клиент – сервер» – вычислительная или сетевая архитектура, в которой задания или сетевая нагрузка распределены между поставщиками услуг, называемыми серверами, и заказчиками услуг, называемыми клиентами. Фактически клиент и сервер – это программное обеспечение. Обычно эти программы расположены на разных вычислительных машинах и взаимодействуют между собой через вычислительную сеть посредством сетевых протоколов, но они могут быть расположены также и на одной машине. Программы-серверы ожидают от клиентских программ запросы и предоставляют им свои ресурсы в виде данных (например, загрузка файлов посредством HTTP, FTP, BitTorrent, потоковое мультимедиа или работа с базами данных) или в виде сервисных функций (например, работа с электронной почтой, общение посредством систем мгновенного обмена сообщениями или просмотр web-страниц во всемирной паутине). Поскольку одна программа-сервер может выполнять запросы от множества программ-клиентов, её размещают на специально выделенной вычислительной машине, настроенной особым образом, как правило, совместно с другими программами-серверами, поэтому производительность этой машины должна быть высокой. Из-за особой роли такой машины в сети, специфики её оборудования и программного обеспечения, её также называют сервером, а машины, выполняющие клиентские программы, соответственно, клиентами.

Существуют концепции построения системы **клиент-сервер**:

- **Слабый клиент – мощный сервер**. В такой модели вся обработка информации перенесена на сервер, а у клиента права доступа строго ограничены. Сервер отправляет ответ, который не требует дополнительной обработки. Клиент взаимодействует с пользователем: составляет и отправляет запрос, принимает результат и выводит информацию на экран.
- **Сильный клиент** – концепция, в которой часть обработки информации предоставляется клиенту. В таком случае сервер выступает хранилищем данных, а вся работа по обработке и представлению информации переносится на компьютер клиента.

Система (приложение), которая основана на клиент-серверном взаимодействии, включает три основных компонента: представление данных, прикладной компонент, компонент управления ресурсами и их хранения.

Двухуровневая архитектура состоит из двух узлов:

- сервер, который отвечает за получение запросов и отправку ответов клиенту, используя при этом только собственные ресурсы;
- клиент, который представляет пользовательский интерфейс.

Принцип работы заключается в том, что сервер получает запрос, обрабатывает его и отвечает напрямую, без использования сторонних ресурсов.

Трехуровневая архитектура состоит из следующих компонентов:

- представление данных – пользовательский интерфейс;

- прикладной компонент – сервер приложений;
- управление ресурсами – сервер базы данных, который предоставляет информацию.

Принцип работы заключается в том, что несколько серверов обрабатывают запрос клиента. Распределение операций снижает нагрузку на сервер.

Трехуровневую архитектуру можно расширить до многоуровневой (N-tier, Multi-tier) способом установки дополнительных серверов. Многоуровневая архитектура позволяет повысить эффективность работы информационной системы, а также оптимизировать распределение ее программно-аппаратных ресурсов.

3. JS – В JavaScript есть следующие типы операторов. Данный подраздел описывает каждый тип и содержит информацию об их приоритетах друг над другом.

- Операторы присваивания
- Операторы сравнения
- Арифметические операторы
- Бинарные операторы
- Логические операторы
- Строковые операторы
- Условный (тернарный) оператор
- Оператор запятая
- Унарные операторы
- Операторы отношения
- Приоритет операторов

JavaScript поддерживает бинарные и унарные операторы, а также ещё один специальный тернарный оператор – условный оператор. **Бинарная операция** использует два операнда, один перед оператором и другой за ним:

Листинг 13. JS операторы (1).

```
1. operand1 operator operand2
```

В свою очередь **унарная операция** использует один операнд, перед или после оператора:

Листинг 14. JS операторы (2)

```
1. operator operand
2. operand operator
```

Выражением является любой корректный блок кода, который возвращает значение. Концептуально, существуют **два типа выражений**: те которые присваивают переменной значение, и те, которые вычисляют значение без его присваивания.

Выражение x = 7 является примером выражения **первого типа**. Данное выражение использует оператор = для присваивания переменной x значения 7. Само выражение также равняется 7.

Код 3 + 4 является примером выражения **второго типа**. Данное выражение использует оператор "+" для сложения чисел 3 и 4 без присваивания переменной полученного результата 7.

Все выражения в JavaScript делятся на следующие категории:

- **Арифметические**: вычисляются в число, например: 3.14159 (Используют арифметические операторы).
- **Строковые**: вычисляются в текстовую строку, например: "Fred" или "234" (Используют строковые операторы).

- **Логические**: вычисляются в true или false (Используют логические операторы).
- **Основные выражения**: Базовые ключевые слова и основные выражения в JavaScript.
- **Левосторонние выражения**: Значениям слева назначаются значения справа.

Базовые ключевые слова и основные выражения в JavaScript.

Оператор **this** [ещё тык сюда]. Используйте ключевое слово **this** для указания на текущий объект. В общем случае **this** указывает на вызываемый объект, которому принадлежит данный метод.

Оператор группировки "скобки" ( ) контролирует приоритет вычисления выражений. Например, вы можете переопределить порядок - "умножение и деление, а потом сложение и вычитание", так чтобы, например, чтобы сложение выполнялось до умножения:

Листинг 15. Оператор группировки.

```
1. var a = 1;
2. var b = 2;
3. var c = 3;
4.
5. // обычный порядок
6. a + b * c     // 7
7. // выполняется, как обычно, так
8. a + (b * c)   // 7
9.
10. // теперь поменяем порядок
11. // сложение до умножения
12. (a + b) * c   // 9
13.
14. // что эквивалентно следующему
15. a * c + b * c // 9
```

Билет №15.

1. Понятие шаблонизатора и маршрутизатора в MVC-фреймворке на сервере и клиенте. (**не особо расписан**)
2. Система доменных имен (DNS). Назначение, расшифровка, алгоритм работы. Пример работы.
3. Алгоритм запуска и работы веб-сервера.

1. Маршрутизация URL позволяет настроить приложение на прием запросов с URL, которые не соответствуют реальным файлам приложения, а также использовать ЧПУ[16], которые семантически значимы для пользователей и предпочтительны для поисковой оптимизации. К примеру, для обычной страницы, отображающей форму обратной связи, URL мог бы выглядеть так: http://www.example.com/contacts.php?action=feedback

Основная цель использования **шаблонизаторов** – это отделение представления данных от исполняемого кода. Часто это необходимо для обеспечения возможности параллельной работы программиста и дизайнера-верстальщика. Использование шаблонизаторов часто улучшает читаемость кода и внесение изменений во внешний вид, когда проект целиком выполняет один человек.

2. DNS (Domain Name System) – компьютерная распределённая система для получения информации о доменах. Чаще всего используется для получения IP-адреса по имени хоста (компьютера или устройства), получения информации о маршрутизации почты и/или обслуживающих узлах для протоколов в домене (SRV-запись).

Основой DNS является представление об иерархической структуре имени и зонах. Каждый сервер, отвечающий за имя, может передать ответственность за даль-

16. Человекопонятный URL.

нейшую часть домена другому серверу (с административной точки зрения – другой организации или человеку), что позволяет возложить ответственность за актуальность информации на серверы различных организаций (людей), отвечающих только за «свою» часть доменного имени.

DNS обладает следующими характеристиками:

- Распределённость администрирования. Ответственность за разные части иерархической структуры несут разные люди или организации.
- Распределённость хранения информации. Каждый узел сети в обязательном порядке должен хранить только те данные, которые входят в его зону ответственности, и (возможно) адреса корневых DNS-серверов.
- Кэширование информации. Узел может хранить некоторое количество данных не из своей зоны ответственности для уменьшения нагрузки на сеть.
- Иерархическая структура, в которой все узлы объединены в дерево, и каждый узел может или самостоятельно определять работу нижестоящих узлов, или делегировать (передавать) их другим узлам.
- Резервирование. За хранение и обслуживание своих узлов (зон) отвечают (обычно) несколько серверов, разделённые как физически, так и логически, что обеспечивает сохранность данных и продолжение работы даже в случае сбоя одного из узлов.

Пример. Предположим, мы набрали в браузере адрес ru.wikipedia.org. Браузер ищет соответствие этого адреса IP-адресу в файле hosts. Если файл не содержит соответствия, то далее браузер спрашивает у сервера DNS: «какой IP-адрес у ru.wikipedia.org»? Однако сервер DNS может ничего не знать не только о запро-

шенном имени, но и даже обо всём домене wikipedia.org. В этом случае сервер обращается к корневому серверу – например, 198.41.0.4. Этот сервер сообщает – «У меня нет информации о данном адресе, но я знаю, что 204.74.112.1 является ответственным за зону org.» Тогда сервер DNS направляет свой запрос к 204.74.112.1, но тот отвечает «У меня нет информации о данном сервере, но я знаю, что 207.142.131.234 является ответственным за зону wikipedia.org.» Наконец, тот же запрос отправляется к третьему DNS-серверу и получает ответ – IP-адрес, который и передаётся клиенту – браузеру.

3. Веб-сервер – это программа, которая принимает входящие HTTP-запросы, обрабатывает эти запросы, генерирует HTTP-ответ и отправляет его клиенту. Общий алгоритм работы веб-сервера можно представить следующим образом (зеленым цветом помечены действия, которые обрабатываются веб-сервером).

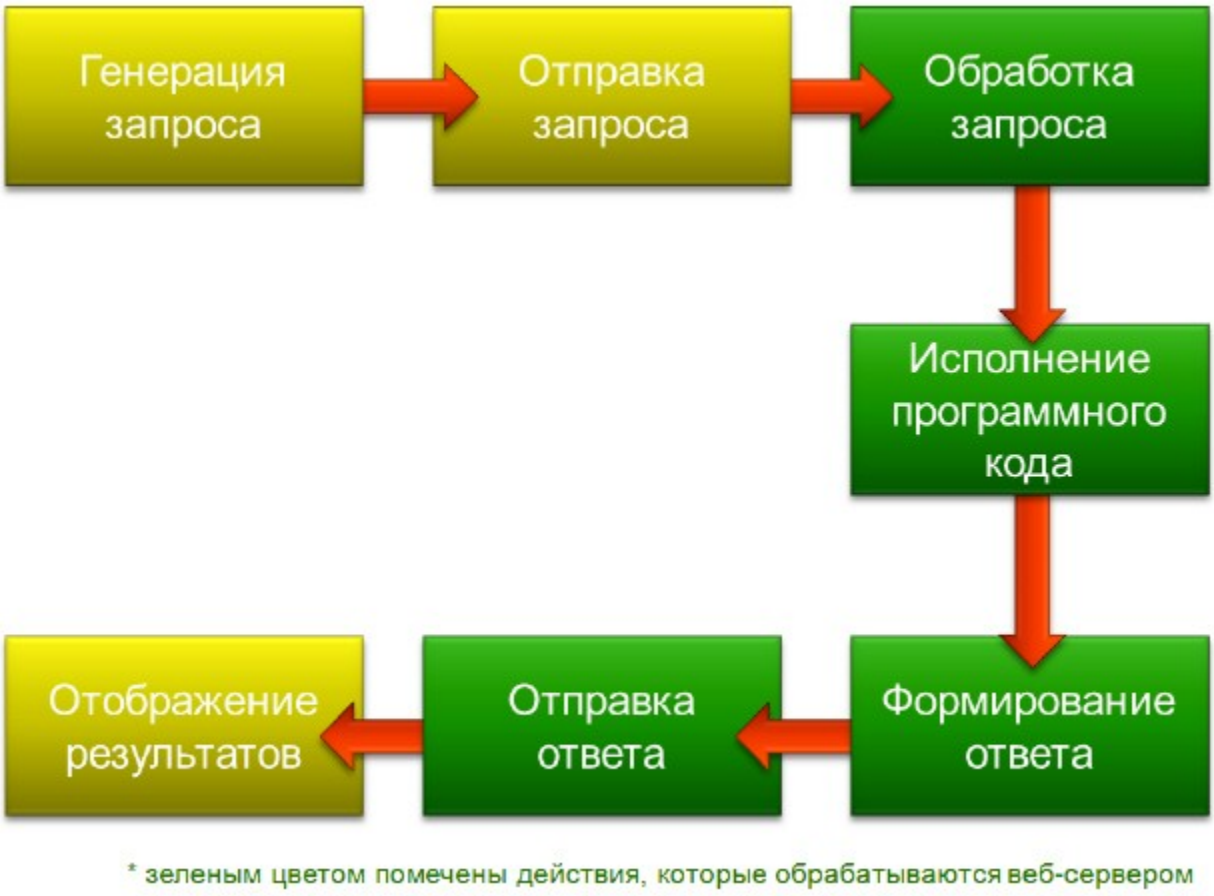

Рисунок 12. Алгоритм работы веб-сервера.

Билет №16.

1. Макетирование. Основные подходы и идеи. Инструменты (Figma).
2. Паттерны проектирования MVC. Примеры паттернов MVC-семейства. Расшифровка аббревиатуры. Описание архитектуры.
3. Технологии толстого клиента. Пример с отображением списка пользователей на Angular/React/Vue. (??? хз чё писать)

1. В **дизайн-макете** отсутствуют интерактивные элементы и анимация, что упрощает его разработку. В то же время позволяет наглядно изучить дизайн будущих страниц в отличие от блочной схемы исходного прототипа.

В **дизайн-проект** входят:

- ✔ Параметры и размеры страницы.
- ✔ Фон и цветовое оформление.
- ✔ Количество и расположение блоков на сайте.
- ✔ Дизайн элементов на странице.
- ✔ Параметры шапки сайта, футера и сайдбара.
- ✔ Границы, отступы между блоками и элементами.

При создании дизайн-макета учитывается фирменный стиль, логотип, корпоративная графика, а также данные из заполненного заказчиком брифа. В ходе разработки каждая деталь обговаривается между дизайнером, верстальщиком и маркетологом, после согласовывается с клиентом. Идеальный макет является точкой схождения мнения клиента и команды проекта, и полностью учитывает технические и визуальные особенности будущего сайта.

Фиксированная. Является одним из наиболее распространённых на сегодняшний день типов. Основная особенность: сайт имеет строго фиксированную ширину и высоту всех блоков и страницы в целом, и не зависит от диагонали монитора.

Достоинства:

- ✔ простота изготовления (а следовательно и меньшая цена)
- ✔ корректное и одинаковое отображение сайта на экранах с разной диагональю.

Недостатки:

- ✗ на маленьких экранах такой сайт будет просто масштабироваться. То есть вы получите уменьшенную точную копию вашей страницы для прочтения которой её нужно будет сначала увеличить. В эпоху мобильного интернета это не самый лучший вариант.
- ✗ на широкоформатных мониторах остаётся много пустого места по бокам
- ✗ при масштабировании страницы она просто становится больше и вылезает за пределы экрана как по высоте, так и по ширине.

Обычно данный тип вёрстки используют для создания **Landing Page**.

Резиновая (тянущаяся). Основная особенность: смысл этой вёрстки заключается в том что размеры всех элементов задаются не в пикселях, а в процентах. И таким образом одни и те же блоки на мониторах с разной диагональю будут иметь разный размер и возможно немного по другому отображаться.

Достоинства:

- ✔ Хорошо смотрится на экранах разной величины.
- ✔ Изменяется при масштабировании страницы.

Недостатки:

- ✗ Более сложный в реализации (а следовательно и более дорогой)
- ✗ Так как блоки сайта растягиваются и сжимаются под ширину экрана – некоторые элементы выглядят не очень красиво на планшетах и телефонах.
- ✗ На мобильный устройствах некоторые элементы за счёт сжатия становятся очень неудобными в использовании. Особенно это касается меню и форм обратной связи.

Такая вёрстка подходит для макетов сайтов, элементы которых размещены в одну колонку. Или в сочетании с фиксированной вёрсткой.

Адаптивная. Этот тип вёрстки наиболее популярен на сегодняшний день так как позволяет создавать «живые» сайты, способные перестраивать свои элементы и изменять их размер в зависимости от того на каком устройстве вы его просматриваете.

Адаптивная вёрстка базируется на резиновой вёрстке + использует медиазапросы или специальные скрипты, которые позволяют определять размер экрана пользователя и в зависимости от этого приписывать конкретным элементам те или иные стили (ширина элементов и положение блоков, размер шрифтов и картинок и даже цвета).

Достоинства:

- ✔ Адаптируется по размеры экрана, за счёт чего ваш сайт хорошо отображается на всех видах устройств.

- ✔ Такой сайт удобно просматривать на мобильных устройствах, так как не нужно ничего увеличивать (всё само подстраивается).
- ✔ При масштабировании страницы в окне браузера элементы так же перестраиваются и страница не теряет свой вид.

Недостатки:

- ✗ Очень трудоёмкий вид вёрстки, что отражается на его цене.
- ✗ Контроль качества выполнения такой вёрстки тоже требует не мало времени (особенно в том случае если сайт большой) так как нужно протестировать вёрстку на разных размерах экранов через онлайн сервисы либо через масштабирование и изменение размеров окна в браузере.

Figma – ну типо *figma* и всё ¯_(ツ)_/¯