Bootcamp для Drupal-разработчиков

Технологии - 2: Разработка сайтов

Новосибирск · Иркутск

Перед началом

- Микрофоны отключены у всех, кроме ведущего
- Назначены со-ведущие
- Запись включена

Все возникающие по ходу вебинара вопросы пишем в чат
Будут паузы для ответов на вопросы

Запись вебинара будет доступна в течение 3 дней
Ссылка на запись и эту презентацию будут в Telegram-группе после окончания вебинара

Цель и план вебинара

Цель:

Разобраться, что из себя представляют веб-приложения (и, как частный случай, — веб-сайты), какие технологии лежат в их основе, и каков процесс их разработки

О чём сегодня поговорим:

- об архитектуре приложений — в целом и веб-приложений — в частности
- о монолитных и распределённых приложениях
- о технологиях разработки веб-приложений
- о требованиях к интерфейсам веб-приложений
- о процессе разработки веб-приложений и вашем месте в этом процессе

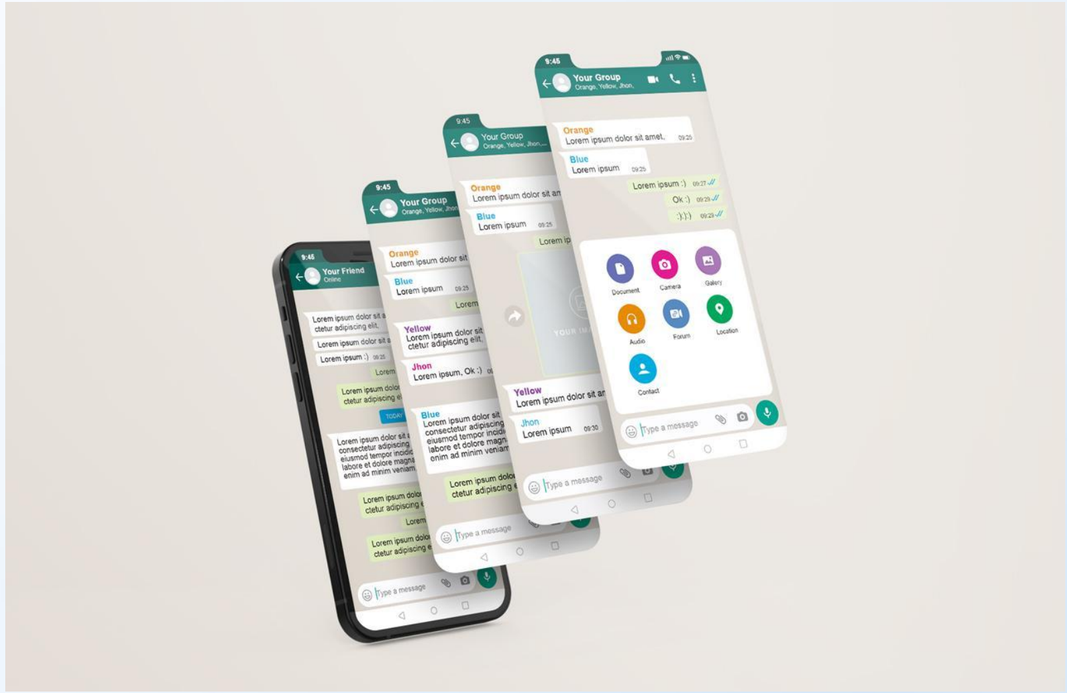
Архитектура приложений

Любое приложение состоит максимум из трёх компонентов:

Интерфейс — то, что мы видим и с чем взаимодействуем

Бизнес-логика — функциональность приложения

Данные — информация, которой приложение оперирует



Изображение: Freepik.com

Архитектура приложений (продолжение)

Любой компонент может быть или на вашем устройстве или на удалённом сервере

Рассмотрим некоторые конфигурации:

- Все компоненты на вашем устройстве — офлайн-приложения. Например, 2ГИС, офлайн-игры
- Интерфейс и логика — на вашем устройстве, а данные — на удалённом сервере. Например, 1С:Бухгалтерия, онлайн-игры
- Интерфейс — у вас, а логика и данные — на удалённом сервере. Это — конфигурация **классического веб-приложения**, о ней будем говорить дальше. «Браузерные» игры тоже здесь
- Интерфейс и «клиентская» логика — на вашем устройстве, а «серверная» логика и данные — на удалённом сервере. Это — конфигурация современного **распределённого веб-приложения**, её мы так же рассмотрим подробно

Классическое веб-приложение

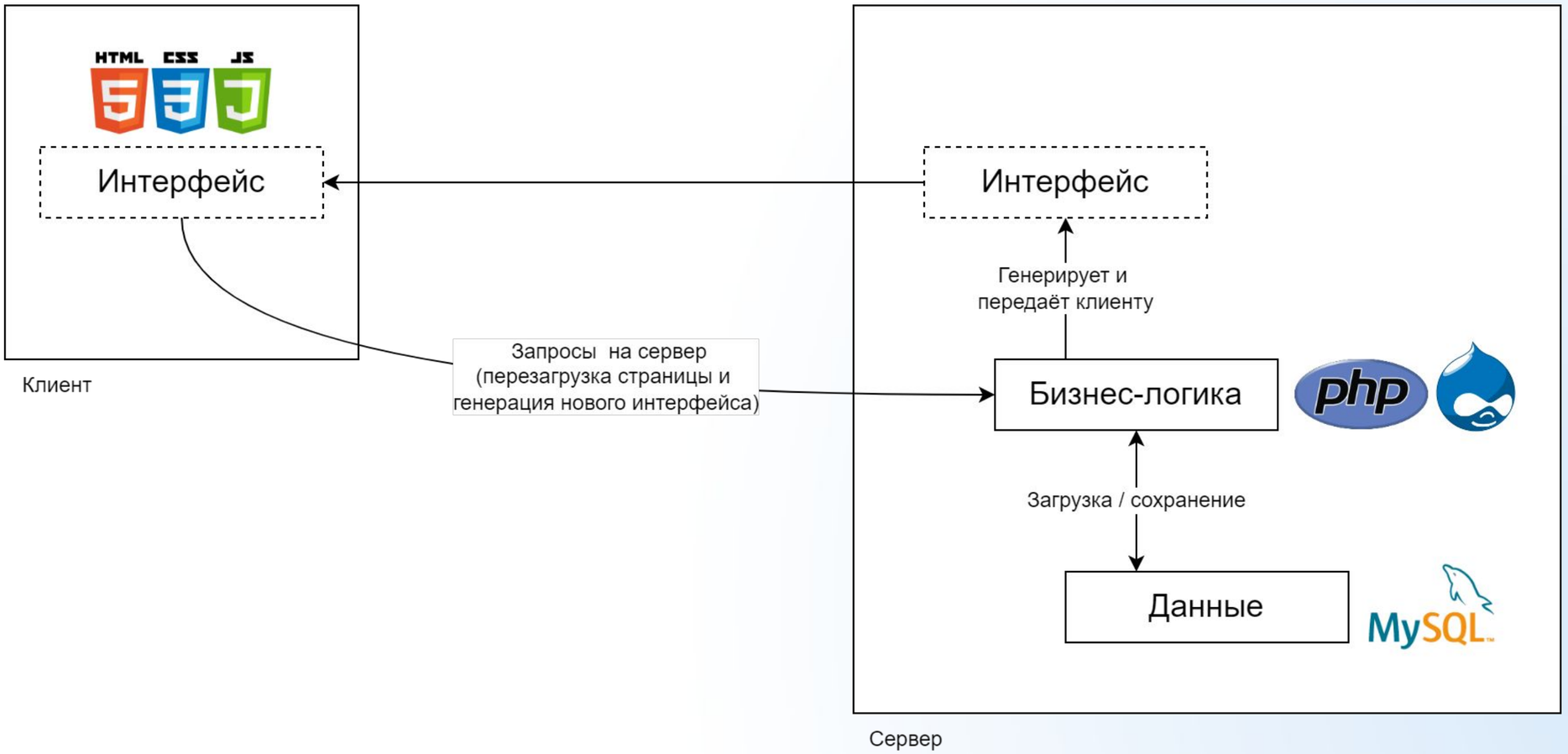
Бизнес-логика и данные хранятся на удалённом сервере, а клиентский интерфейс (дизайн, интерактив) генерируется серверной бизнес-логикой и отдаётся в браузер в виде готовых веб-страниц для отображения и работы на вашем устройстве

Раньше все веб-приложения были именно такие потому, что интернет был очень медленный, а клиентские устройства — очень слабые

Логичным было основную работу делать на сервере и лишь легковесный готовый результат передавать на сторону клиента

Небольшое количество клиентов, решаемые задачи, объём данных и мощность серверов позволяли веб-приложениям работать на приемлемом уровне

Схема классического веб-приложения



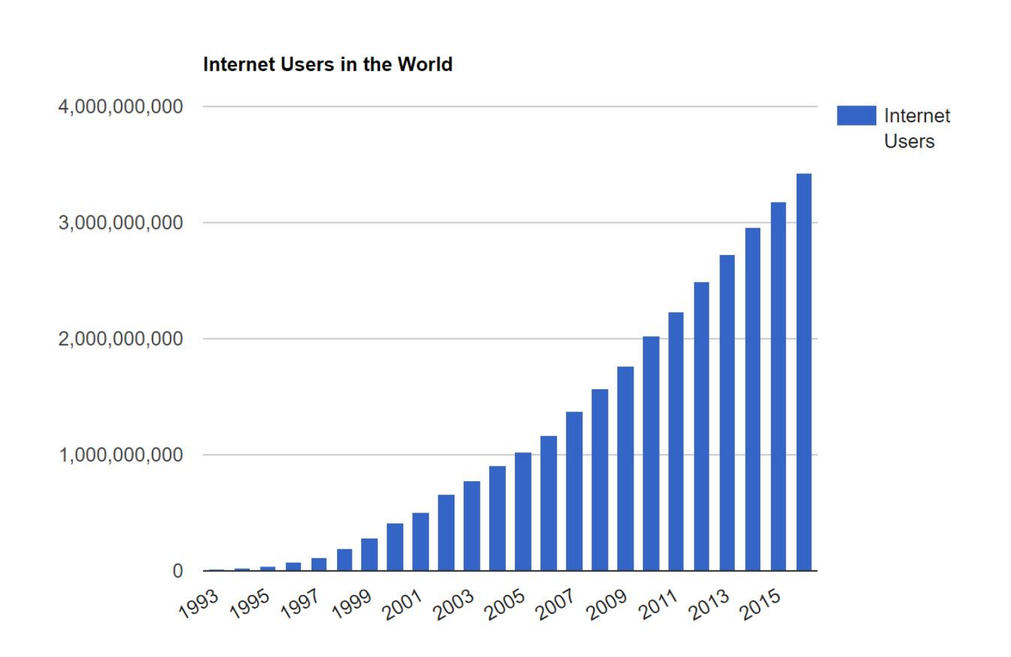
Бурное развитие интернета

Веб постоянно развивается:

- количество пользователей / устройств ▲
- пропускная способность каналов ▲
- сложность решаемых с помощью веба задач ▲
- объём данных и, как следствие, объём трафика ▲
- мощность серверов и **пользовательских устройств** ▲
- Развиваются браузеры и язык JavaScript

Растёт требовательность пользователей к скорости обслуживания

Классическая архитектура веб-приложений неспособна решить целый ряд современных задач с приемлемым уровнем качества



Распределённое веб-приложение

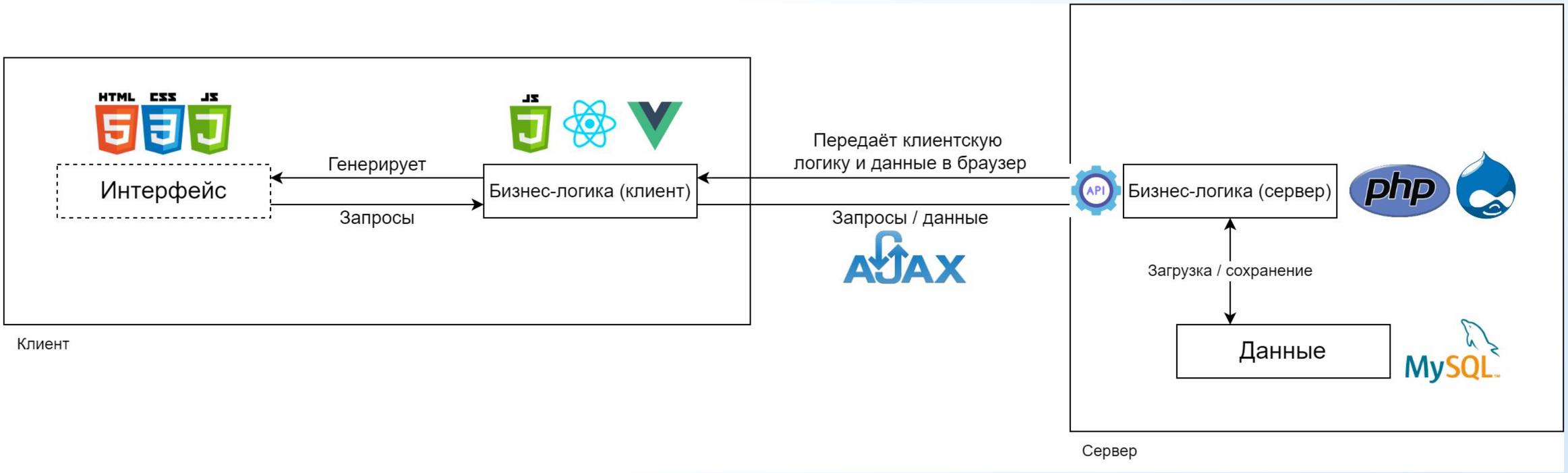
Одним из логичных решений по оптимизации стало движение в сторону переноса части бизнес-логики на сторону клиента, программируя её на JavaScript

Эта «клиентская» логика веб-приложений работает в вашем браузере, решая следующие задачи:

- полноценная генерация пользовательского интерфейса
- решение части бизнес-задач веб-приложения
- обработка и подготовка данных
- взаимодействие с «серверной» логикой

Такое приложение относится к классу **распределённых**. Вы, как архитекторы, сами решаете, какую часть вашего веб-приложения переложить на сторону клиента

Схема распределённого веб-приложения

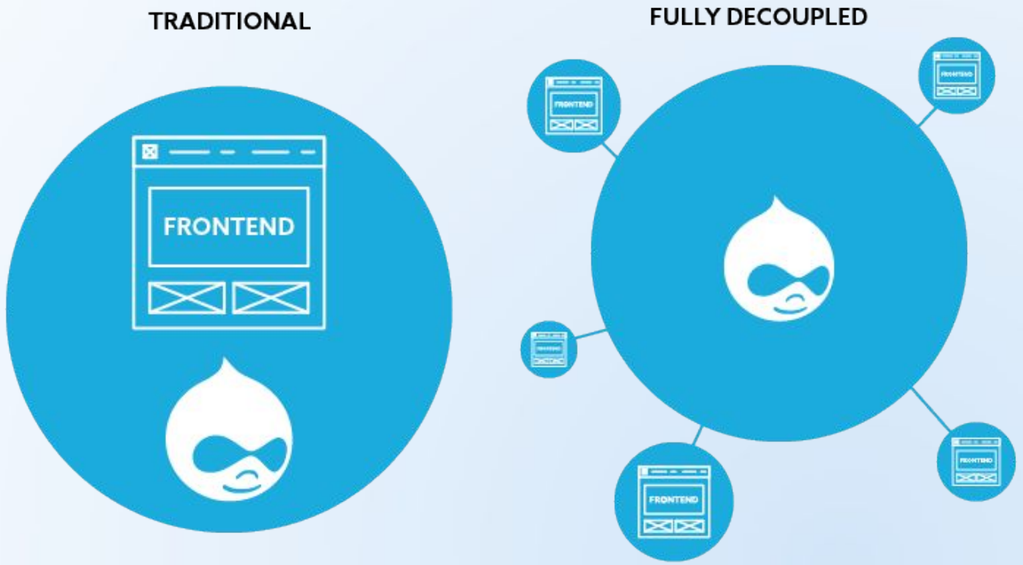


А как в Drupal?

Drupal позволяет реализовать любой из вариантов

Выбор варианта реализации зависит от нескольких параметров, о которых говорили выше

Классическое веб-приложение в любой момент может быть преобразовано в распределённое ~~и наоборот, но так никогда никто не делает~~



Клиентские технологии

HTML — (HyperText Markup Language — язык гипертекстовой разметки) — служит для представления информации на веб-страницах. Определяет структуру документа. В HTML не определяется внешнего вида элементов документа

CSS — (Cascading Style Sheets — каскадные таблицы стилей) — формальный язык описания внешнего вида документа, написанного с использованием языка разметки. Используется для оформления веб-страниц

JavaScript — алгоритмический язык программирования, который может быть выполнен в браузере. Изначально задумывался специально для того, чтобы создавать интерактивные сайты, а затем эволюционировал в полноценный язык для реализации клиентских приложений

Клиентские технологии (продолжение)

Bootstrap — готовая библиотека CSS-компонентов и сопутствующего инструментария для быстрого создания сайтов и приложений

jQuery — библиотека функций для JavaScript, фокусирующийся на взаимодействии JavaScript и HTML. Когда-то давно очень помогала ускорить разработку JavaScript-кода. В настоящее время потеряла актуальность и популярна только благодаря legacy-проектам

React, Angular, Vue.js — полноценные фреймворки для построения клиентских веб-приложений на JavaScript

Браузеры

Что такое браузер, знают все. Их много, они от разных производителей и под разные платформы

Наша задача, как разработчиков, — сделать универсальный интерфейс, который везде отображается и работает одинаково

Разберём такие понятия, как кроссбраузерность, адаптивность, валидность и семантичность

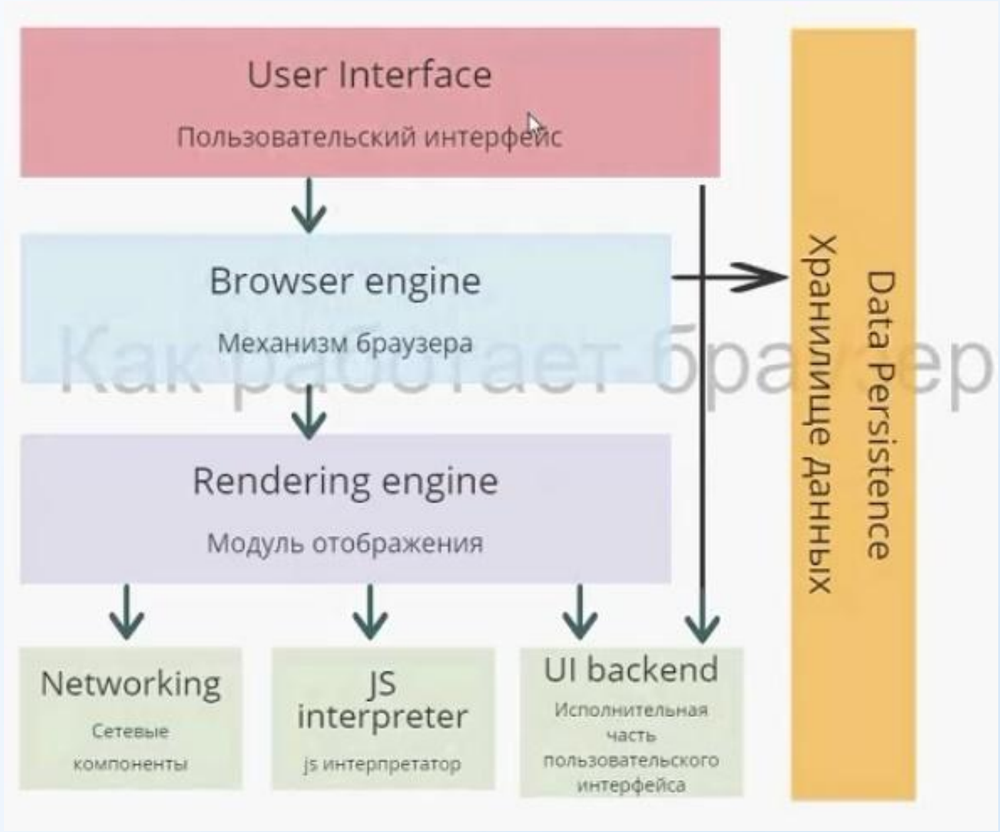
Кроссбраузерность

Браузер — это целый программный комплекс, имеющий следующую архитектуру

Нас прежде всего интересуют Модуль отображения и JS-интерпретатор. Они (увы) могут в разных браузерах вести себя по-разному

И если мы сделали интерфейс так, что он везде всё отображается и работает одинаково, то его можно назвать кроссбраузерным

А **кроссбраузерностью** называют способность интерфейса (в нашем случае — сайта) корректно отображаться и функционировать в разных браузерах



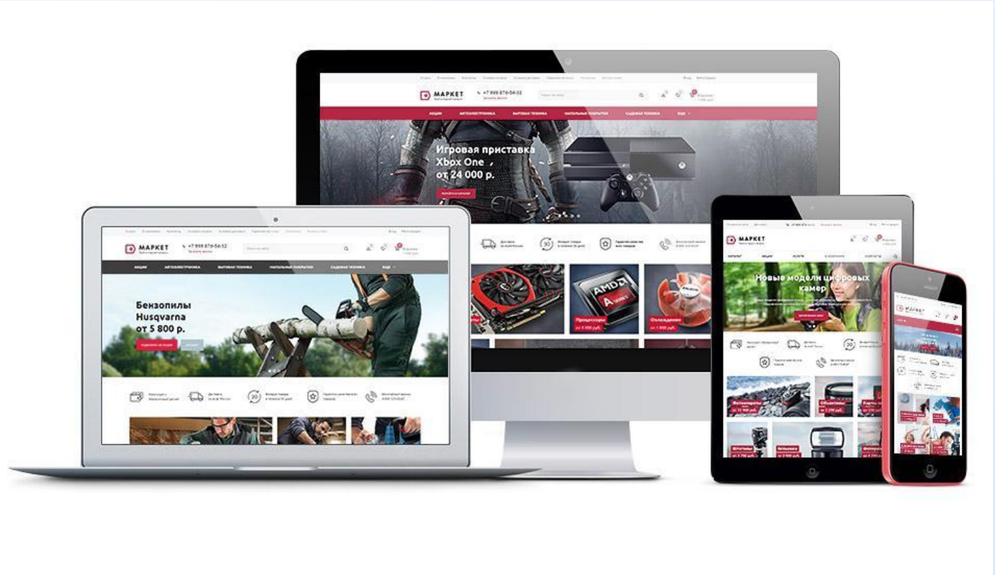
Адаптивный и отзывчивый дизайн

В современном мире мобильными устройствами для выхода в интернет пользуется уже гораздо больше половины жителей нашей страны

А значит, сайты, которые мы разрабатываем, должны быть удобными как для пользователей ПК, так и для пользователей мобильных устройств

Одним из способов адаптировать веб-сайт под мобильные устройства является создание **адаптивного** (или лучше — **отзывчивого**) дизайна

Адаптивный сайт «подстраивается» под ширину экрана устройства, на котором мы его открываем



Адаптивный дизайн / Альтернативы

Подход с адаптивным (отзывчивым) дизайном не лишён недостатков:

- в случае если дизайн для мобильных сильно отличается от дизайна для ПК, то переделок становится слишком много и вся выгода теряется
- даже если отличия небольшие, блоки и контент, который скрывается в мобильном варианте, всё равно загружается с сервера, что расходует мобильный трафик впустую

Поэтому есть альтернативы, как адаптировать интерфейс под разные устройства:

- отдельные сайты под разные устройства (под ПК — своя, под мобильные — своя)
- динамическая загрузка контента
- отдельное мобильное приложение

Валидность кода

Валидность HTML-кода и CSS-стилей — это когда они написаны в соответствии с определёнными стандартами. Стандарты разрабатывает и поддерживает Консорциум Всемирной Паутины — World Wide Web Consortium (W3C)

По задумке, валидный код всегда даст идентичный результат во всех браузерах. Это — попытка мотивировать разработчиков браузеров соблюдать единые стандарты. По факту, увы, валидность залогом полной идентичности не является, нюансы есть всё равно. Но стремиться писать валидный код всё равно нужно, так как это способ максимально приблизиться к идеалу

Проверить свой код на валидность можно с помощью бесплатных инструментов: https://validator.w3.org/ и https://jigsaw.w3.org/css-validator/

Семантичность кода

Ещё одно свойство HTML-документа, суть которого заключается в возможности документа передавать своё смысловое и логическое содержание при отключенном оформлении

HTML содержит множество тегов и рекомендаций по их использованию, описанных в стандарте. Правильное их применение может позволить разработчикам, поисковым машинам, screen reader устройствам правильно интерпретировать смысловые блоки документа, выделять главное, расставлять акценты

Это также является шагом к ещё большему охвату кроссбраузерности

Клиентские технологии / Выводы

1. HTML, CSS и JavaScript — три кита, на которых держится frontend
2. «Мобильная» аудитория не менее важна, чем пользователи ПК
3. Кроссбраузерность необходимо стараться реализовывать в полной мере (при условии наличия у всех целевых браузеров соответствующих возможностей)
4. Семантичность и валидность очень желательны
5. Выбор варианта реализации мобильного сайта определяется индивидуально.
6. Библиотеки и фреймворки — это замечательно, а порой даже необходимо. Но, используя их, вы должны понимать принцип их работы

Серверные технологии

PHP — C-подобный скриптовый язык общего назначения, интенсивно применяемый для разработки веб-приложений. В настоящее время поддерживается подавляющим большинством хостинг-провайдеров и является одним из лидеров среди языков, применяющихся для создания динамических веб-сайтов

MySQL — свободная реляционная система управления базами данных. Является отличным решением для малых и средних приложений

Drupal (Дру́пал) — система управления содержимым (CMS), используемая также как фреймворк для веб-приложений (CMF), написанная на языке PHP и использующая в качестве хранилища данных реляционную базу данных (поддерживаются MySQL, PostgreSQL и другие). Является свободным программным обеспечением и развивается усилиями энтузиастов со всего мира

Начиная с Drupal 8, система была переписана на основе фреймворка Symfony. Файлы шаблонов изменили расширение с .tpl на .twig

Связь клиента и сервера / AJAX

AJAX, Ajax — (Asynchronous Javascript and XML — «асинхронный JavaScript и XML») — подход к построению веб-приложений, заключающийся в «фоновом» обмене данными браузера с веб-сервером. В результате при обновлении данных веб-страница не перезагружается полностью, и веб-приложения становятся быстрее и удобнее

Примеры, где применяется:

- бесконечная подгрузка ленты новостей
- обмен сообщениями в веб-мессенджерах
- обмен информацией между компонентами распределённого приложения

Связь клиента и сервера / API

API — (Application Programming Interface — «программный интерфейс приложения») — описание способов (набор классов, процедур, функций, структур или констант), которыми одна компьютерная программа может взаимодействовать с другой программой

REST — (Representational State Transfer — «передача репрезентативного состояния» или «передача „самоописываемого“ состояния») — архитектурный стиль взаимодействия компонентов распределённого приложения в сети. Работает «поверх» HTTP(S)-протокола

Есть понятие endpoint — URL, на который приходят запросы

Есть 4 доступных метода: GET, POST, DELETE, PUT, реализующих CRUD-модель

Через наличие этих двух составляющих клиенты могут взаимодействовать с серверами и обмениваться данными по сети

Процесс разработки сайтов

Классический процесс состоит из нескольких **этапов**:

1. Проектирование:

- сбор требований
- прототипирование
- разработка технического задания (и технического проекта)

2. Разработка:

- разработка дизайна
- вёрстка и создание интерактива
- программирование клиентской бизнес-логики *(для распределённых приложений)*
- интеграция вёрстки в CMS, настройка *(для классических приложений)*
- программирование серверной бизнес-логики
- программирование API *(для распределённых приложений)*
- сборка *(для распределённых приложений)*

Процесс разработки сайтов / Продолжение

3. Ввод в эксплуатацию:

- документирование
- бета-тестирование
- запуск в эксплуатацию

4. Гарантийная поддержка

5. Техническое обслуживание и поддержка

Результат каждого из этапов может являться входными данными для последующих этапов. Зелёным цветом выделены этапы, где участвует backend-разработчик

Что дальше

Домашнее задание:

Понять всё то, что не поняли на вебинаре.

Если поняли всё, самостоятельнао поработать с открытыми источниками на углубление полученных знаний

Ближайший план работы:

- с 04.10.2022 — теоретический зачёт по 1 блоку
- с 04.10.22 переходим ко 2 блоку, начинаем осваивать клиентские технологии. Дополнительные инструкции будут в основной группе

Спасибо за внимание!

Новосибирск · Иркутск

Заголовок в одну строку

Научимся как правильно проектировать и планировать разработку функций, как вести инструкции и согласовывать с командой. Научимся как правильно проектировать и планировать разработку функций, как вести инструкции и согласовывать с командой.

Подзаголовок:

- Научимся как правильно проектировать;
- Планировать разработку функций;
- Вести инструкции и согласовывать с командой.

Заголовок в две строки: Архитектура крупных корпораций

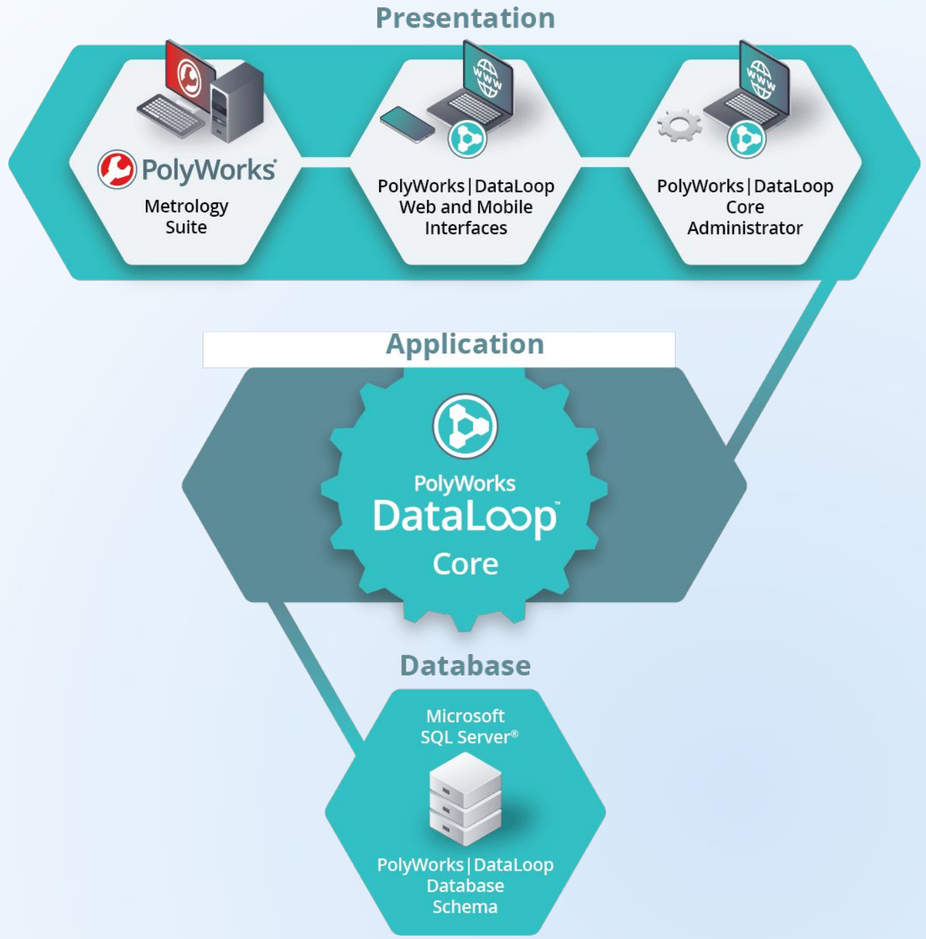
Научимся как правильно проектировать и планировать разработку функций, как вести инструкции и согласовывать с командой. Научимся как правильно проектировать и планировать разработку функций, как вести инструкции и согласовывать с командой.

Подзаголовок:

- Научимся как правильно проектировать;
- Планировать разработку функций;
- Вести инструкции и согласовывать с командой.

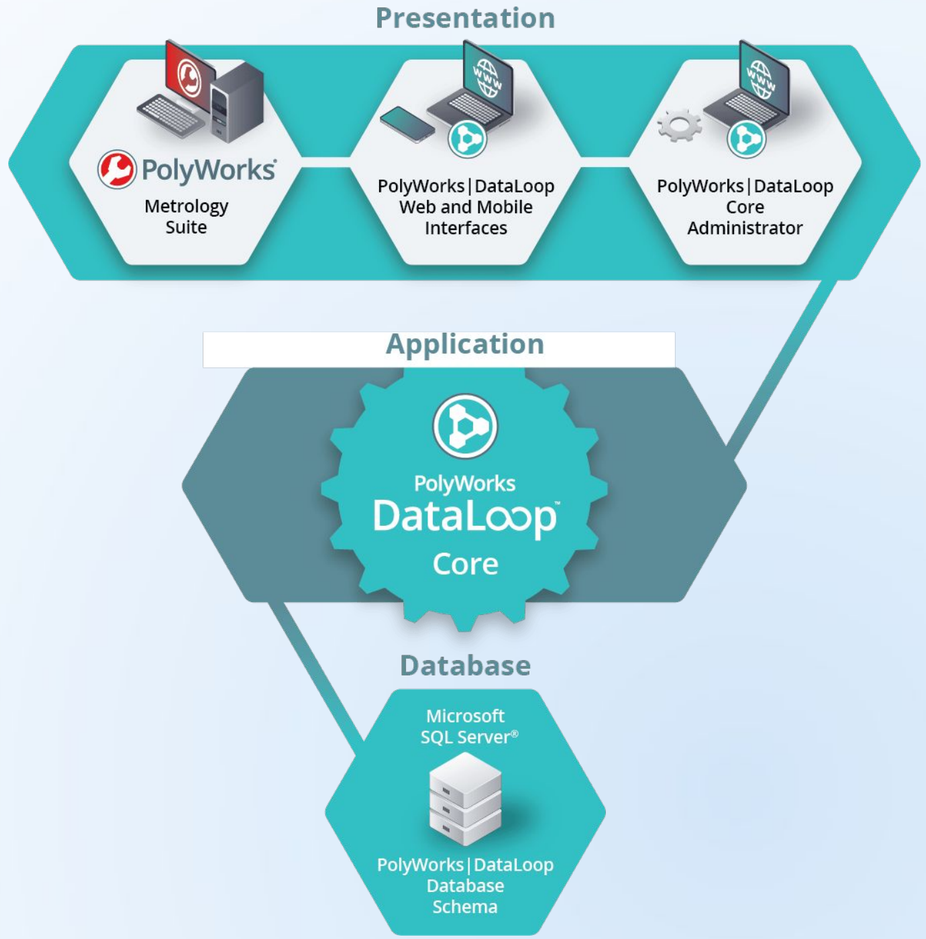
Заголовок в одну строку

- Научимся как правильно проектировать и планировать разработку функций.
- Как вести инструкции и согласовывать с командой.
- Научим правильно планировать работу.
- Правильно проектировать функции.



Заголовок в две строчки: Архитектура

- Научимся как правильно проектировать и планировать разработку функций.
- Как вести инструкции и согласовывать с командой.
- Научим правильно планировать работу.
- Правильно проектировать функции.



Заголовок в три строчки: Архитектура крупных корпораций

- Научимся как правильно проектировать и планировать разработку функций.
- Как вести инструкции и согласовывать с командой.
- Научим правильно планировать работу.
- Правильно проектировать функции.