[**1. Обзор и сравнение популярных веб-фреймворков: React, Vue и Angular**](#_heading=h.ej5yd795cy3s) **3**

[**2. Библиотека React, ее характеристика и возможности**](#_heading=h.3xft7lusvj01) **5**

[**3. Фреймворк Vue.js**](#_heading=h.69rqi14n50u2) **6**

[**4. Среда Node.js**](#_heading=h.8s0tmm3jlb6h) **7**

[**5. V8, Rhino и другие движки (engines) JavaScript**](#_heading=h.2imd57akysoi) **8**

[**6. Фреймворк Express**](#_heading=h.q0xu5vo5maol) **9**

[**7. Метод Fetch**](#_heading=h.c65krtpix6i7) **11**

[**8. Библиотека Axios**](#_heading=h.xpfd6pihyd4x) **14**

[**9. Cheerio: предназначение, возможности и применение**](#_heading=h.ic3akya5gjeh) **16**

[**10. Сервис Ngrok**](#_heading=h.ln0homq9jly) **19**

[**11. Библиотека Socket.io: её предназначение, применение и возможности**](#_heading=h.yifzq6rfzjlo) **21**

[**12. Bootstrap – как и для чего применяется данная технология**](#_heading=h.ndbqpsfop8b6) **22**

[**13. Стрелочные функции языка JavaScript**](#_heading=h.m1hw5g5q85s9) **24**

[**14. Промис (promise) – понятие и примеры использования**](#_heading=h.krszck5a8mo2) **26**

[**15. Колбэки (callbacks) – понятие и примеры использования**](#_heading=h.eudwl242h8ua) **27**

[**16. Отладка программного кода JavaScript на стороне клиента и сервера**](#_heading=h.a9njm92ro289) **29**

[**17. GET и POST-запросы в веб-программировании**](#_heading=h.4muy9h57kel4) **30**

[**18. Фреймворк NestJS**](#_heading=h.6kcmwmx2006q) **31**

[**19. Язык программирования TypeScript, его отличия от JavaScript**](#_heading=h.1wlzn38ut90g) **32**

[**20. Браузерное окружение и DOM**](#_heading=h.g0asblpr0jl7) **33**

[**21. Браузерные события**](#_heading=h.ts4i8lvxc7a9) **35**

[**22. Куки, LocalStorage и IndexedDB**](#_heading=h.254udeswqglh) **36**

[**23. Протокол WebSocket («веб-сокет»)**](#_heading=h.dr1s3odqrwbq) **38**

[**24. Рендеринг на клиенте, на сервере и генерация статических сайтов**](#_heading=h.mf68mzzbt37o) **40**

[**25. Популярные CMS: 1С-Битрикс, WordPress, Joomla! и другие.**](#_heading=h.z6427gbo78ld) **43**

[**Примеры практических заданий в билетах**](#_heading=h.z170o57njw8i) **45**

# 1. Обзор и сравнение популярных веб-фреймворков: React, Vue и Angular

Веб-фреймворки представляют собой удобные инструменты для создания сложных веб-структур – сайтов, одностраничных приложений (Single Page Application(SPA)), микросервисов и т.д.

К основным преимуществам веб-фреймворков можно отнести удобную модель управления бизнес логикой сложных приложений, стандартизацию структуры проектов, с учетом заложенных в модели разработки общепринятых норм, большое количество оптимизированного экосистемного программного обеспечения и многое другое.

Стандартизация процесса разработки на базе фреймворка, упрощает поддержку проектов для опытных разработчиков, и позволяет начинающим специалистам быстрее освоиться. Крупным компаниям с долгосрочными, высоконагруженными проектами выгоднее использовать именно фреймворки.

**React**

React представляет собой JavaScript библиотеку с открытым исходным кодом.

Относительно конкурентов React является самым «гибким» вариантом в плане настройки и оптимизации. Приложения, созданные с помощью React, имеют произвольную структуру, что является большим плюсом для опытного разработчика. Практически неограниченный доступ к низкоуровневым объектам позволяет качественнее оптимизировать сложные приложения и управлять ими. Однако, гибкость и глубина в плане низкоуровневости являются не только преимуществом React, но и недостатком.

Начинающий разработчик с большой долей вероятности не сможет качественно использовать эти возможности, а создаст менее эффективное решение с громоздким и неоптимизированным кодом, чем если бы использовал аналоги.

**Angular**

Angular официально позиционируется как фреймворк с открытым исходным кодом, а языком для разработки на нём служит TypeScript. Это позволяет достичь высокого комфорта в работе для опытных разработчиков, однако вызывает большие трудности у начинающих. Именно поэтому Angular признан самым сложным в освоении фреймворком, среди React, Vue и Angular.

Основной разработки на Angular является классово-модульный подход и использование готовых встроенных решений для масштабирования, тестирования и оптимизации приложений. У Angular очень строгая структура проектов, в отличии от React, а благодаря статической типизации TypeScript, разработчик уже на этапе разработки избавляется от многих ошибок, допустимых на ранних этапах в JavaScript-фреймворках.

Однако, подход «всё включено» обладает недостатком в виде большого размера фреймворка относительно конкурентов. Так, например, при создании небольшого сайта, разработчику не требуется широкий функционал, по умолчанию присутствующий во-фреймворке, однако приходится устанавливать весь пакет.

**Vue**

VueJS – прогрессивный JavaScript фреймворк с открытым исходным кодом, который может использоваться и как библиотека.

Первым плюсом Vue считается его относительное удобство за счёт сокрытия определённых абстракций от разработчика и таким образом получение достаточно мощного функционала при минимальном количестве кода.

К очередному достоинству можно отнести высокую производительность за счёт использования модели Virtual DOM, поддержка которой была полностью переработана во Vue 3.0, что позволило достичь ещё большей скорости работы даже в сравнении с React, аналогично использующим Virtual DOM.

В Vue, как и в React, существует возможность использования сторонних библиотек управления состоянием приложения, например, Redux.

# 2. Библиотека React, ее характеристика и возможности

React — JavaScript-библиотека для работы с пользовательскими интерфейсами (UI), которую создали разработчики Facebook. Библиотеку начали использовать на сайте этой социальной сети в 2011 году. А в 2013 году Facebook открыл исходный код React.

С помощью React разработчики создают веб-приложения, которые изменяют отображение без перезагрузки страницы. Благодаря этому приложения быстро реагируют на действия пользователя, например, заполнение форм, применение фильтров, добавление товаров в корзину и так далее.

Компонент React — это участок кода, который представляет часть веб-страницы. Каждый компонент — это JavaScript-функция, которая возвращает кусок кода, представляющего фрагмент страницы.

Для формирования страницы мы вызываем эти функции в определённом порядке, собираем вместе результаты вызовов и показываем их пользователю.

React использует язык программирования, называемый JSX, который похож на HTML, но работает внутри JavaScript, что отличает его от HTML.

Компоненты React можно помещать в другие компоненты.

Классы компонентов должны содержать функцию, называемую render(). Эта функция возвращает JSX-код компонента.

Компоненты, основанные на классах, могут хранить информацию о текущей ситуации. Эта информация называется состоянием (state), она хранится в JS-объекте.

React использует виртуальный DOM (Virtual DOM). Виртуальный DOM — объект, в котором хранится информация о состоянии интерфейса. При изменении состояния, например, после отправки формы или нажатия кнопки, React рассчитывает разницу между старым и новым состоянием. После этого библиотека отрисовывает новое состояние. Использование виртуального DOM позволяет библиотеке эффективно обновлять реальный DOM.

# 3. Фреймворк Vue.js

Vue.js — это JavaScript библиотека для создания веб-интерфейсов с использованием шаблона архитектуры [MVVM](https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93viewmodel) (Model-View-ViewModel).

Поскольку Vue работает только на «уровне представления» и не используется для промежуточного программного обеспечения и бэкэнда, он может легко интегрироваться с другими проектами и библиотеками. Vue.js содержит широкую функциональность для уровня представлений и может использоваться для создания мощных одностраничных веб-приложений.

Функции Vue.js:

* Реактивные интерфейсы;
* Декларативный рендеринг;
* Связывание данных;
* Директивы (все директивы имеют префикс «V-». В директиву передается значение состояния, а в качестве аргументов используются html атрибуты или Vue JS события);
* Логика шаблонов;
* Компоненты;
* Обработка событий;
* Свойства;
* Переходы и анимация CSS;
* Фильтры.

Основная библиотека Vue.js 2 очень маленькая (всего 17 кБ). Это гарантирует, что нагрузка на ваш проект, реализованный с помощью Vue.js, минимальна, а ваш сайт будет быстро загружаться.

Vue подходит для небольших проектов, которым необходимо добавить немного реактивности, представить форму с помощью AJAX, отобразить значения при вводе данных пользователем, авторизация или другие аналогичные задачи. Vue легко масштабируется и хорошо подходит для объемных проектов, поэтому его называют прогрессивным фреймворком.

# 4. Среда Node.js

Node.js — это опенсорсная кроссплатформенная среда выполнения для JavaScript, которая работает на серверах. С момента выпуска этой платформы в 2009 году она стала чрезвычайно популярной и в наши дни играет весьма важную роль в области веб-разработки.

Node.js представляет среду выполнения кода на JavaScript, которая построена на основе движка JavaScript Chrome V8, который позволяет транслировать вызовы на языке JavaScript в машинный код. Node.js прежде всего предназначен для создания серверных приложений на языке JavaScript. Хотя также существуют проекты по написанию десктопных приложений (Electron) и даже по созданию кода для микроконтроллеров.

Основные особенности Node.js.

* *Скорость:* JavaScript-код, выполняемый в среде Node.js, может быть в два раза быстрее, чем код, написанный на компилируемых языках. Причиной подобного является неблокирующая архитектура платформы.
* *Простота:* Платформа Node.js проста в освоении и использовании. Для старта работы необходимо несколько строк кода.
* *JavaScript:* В среде Node.js выполняется код, написанный на JavaScript. Это означает, что миллионы фронтенд-разработчиков, которые уже пользуются JavaScript в браузере, могут писать и серверный, и клиентский код на одном и том же языке программирования без необходимости изучать совершенно новый инструмент для перехода к серверной разработке.
* *Движок V8:* В основе Node.js, помимо других решений, лежит опенсорсный JavaScript-движок V8 от Google, применяемый в браузере Google Chrome и в других браузерах. Это означает, что Node.js пользуется наработками тысяч инженеров, которые сделали среду выполнения JavaScript Chrome невероятно быстрой.
* *Асинхронность:* JavaScript значительно упрощает написание асинхронного и неблокирующего кода с использованием единственного потока, функций обратного вызова (коллбэков) и подхода к разработке, основанной на событиях.
* *Библиотеки:* Благодаря простоте и удобству работы с менеджером пакетов для Node.js, который называется [npm](https://flaviocopes.com/npm/), экосистема Node.js процветает. Сейчас в [реестре npm](https://www.npmjs.com/) имеется более полумиллиона опенсорсных пакетов, которые может свободно использовать любой Node.js-разработчик.

# 5. V8, Rhino и другие движки (engines) JavaScript

JavaScript-движок — это программа, или интерпретатор, выполняющий код, написанный на JavaScript. Движок может быть реализован с использованием различных подходов: в виде обычного интерпретатора, в виде динамического компилятора (или JIT-компилятора), который, перед выполнением программы, преобразует исходный код на JS в байт-код некоего формата.

Пайплайн (pipeline) движка JavaScript:

1. Движок JavaScript обрабатывает исходный код и представляет его в виде абстрактного синтаксического дерева (АСТ).
2. Основываясь на построенном АСТ, интерпретатор может начать генерировать байткод. Именно в этот момент движок выполняет JavaScript код.
3. Чтобы он исполнялся быстрее, можно отправить байткод в оптимизирующий компилятор вместе с данными профилирования (profiling data). Оптимизирующий компилятор делает определенные предположения на основе данных профилирования, затем он генерирует высокооптимизированный машинный код.
4. Если в какой-то момент предположения оказываются неверными, оптимизирующий компилятор деоптимизирует код и возвращается на этап интерпретатора.

Вот список популярных реализаций JavaScript-движков.

V8 — движок с открытым исходным кодом, написан на C++, его разработкой занимается Google.

Rhino — этот движок с открытым кодом поддерживает Mozilla Foundation, он полностью написан на Java.

SpiderMonkey — это самый первый из появившихся JS-движков, который в прошлом применялся в браузере Netscape Navigator, а сегодня — в Firefox.

JavaScriptCore — ещё один движок с открытым кодом, известный как Nitro и разрабатываемый Apple для браузера Safari.

Nashorn — движок с открытым кодом, являющийся частью OpenJDK, которым занимается Oracle.

# 6. Фреймворк Express

**Express** — это простейшая, гибкая и оптимизированная платформа Node.js, упрощающая разработку веб-приложения, которое может обрабатывать различные типы запросов, например GET, PUT, POST и DELETE.

Что такое фреймворк? **Фреймворк** — заготовка, готовая модель в IT для быстрой разработки, на основе которой можно дописать собственный код. Он задает структуру, определяет правила и предоставляет необходимый набор инструментов для создания проекта. В основном фреймворки используются в веб-разработке.

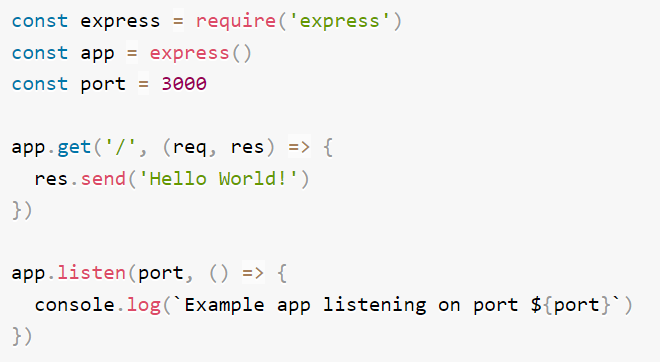
Express предоставляет следующие механизмы:

* Написание обработчиков для запросов с различными HTTP-методами в разных URL-адресах (маршрутах).
* Интеграцию с механизмами рендеринга «view», для генерации ответов, вставляя данные в шаблоны.
* Установка общих параметров веб-приложения, такие как порт для подключения, и расположение шаблонов, которые используются для отображения ответа.
* «промежуточное ПО» для дополнительной обработки запроса в любой момент в конвейере обработки запросов.

В то время как сам express довольно минималистичный, разработчики создали совместимые пакеты промежуточного программного обеспечения для решения практически любой проблемы с веб-разработкой. Существуют библиотеки для работы с куки-файлами, сеансами, входами пользователей, параметрами URL, данными POST, заголовками безопасности и многими другими.

**Пример:**

Приложение запускает сервер и слушает соединения на порте 3000. Приложение выдает ответ “Hello World!” на запросы, адресованные корневому URL (/) или ***маршруту***. Для всех остальных путей ответом будет **404 Not Found**.



Если не вдаваться в подробности, то можно выделить четыре стадии происходящего, которые мы можем проанализировать:

1. Создание нового приложения express.
2. Создание нового маршрута.
3. Запуск HTTP-сервера на заданном номере порта.
4. Обработка поступающих к серверу запросов.

# 7. Метод Fetch

**Fetch()** позволяет делать запросы, схожие с XMLHttpRequest (XHR), то есть легко получать ресурсы асинхронно. Основное отличие заключается в том, что Fetch API использует Promises (Обещания), которые позволяют использовать более простое и чистое API, избегать катастрофического количества callback'ов и необходимости помнить API для XMLHttpRequest. **Fetch()** предоставляет улучшенный интерфейс для осуществления запросов к серверу: как по части возможностей и контроля над происходящим, так и по синтаксису (из-за промисов).

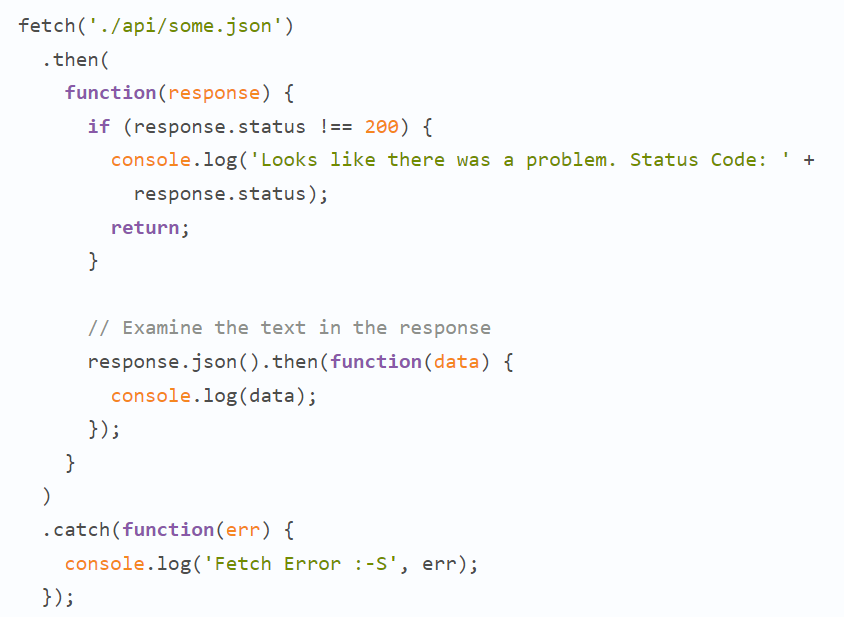
Синтаксис метода:



Свойства options:

* method – метод запроса,
* headers – заголовки запроса (объект),
* body – тело запроса: FormData, Blob, строка и т.п.
* mode – одно из: «same-origin», «no-cors», «cors», указывает, в каком режиме кросс-доменности предполагается делать запрос.
* credentials – одно из: «omit», «same-origin», «include», указывает, пересылать ли куки и заголовки авторизации вместе с запросом.
* cache – одно из «default», «no-store», «reload», «no-cache», «force-cache», «only-if-cached», указывает, как кешировать запрос.
* redirect – можно поставить «follow» для обычного поведения при коде 30x (следовать редиректу) или «error» для интерпретации редиректа как ошибки.

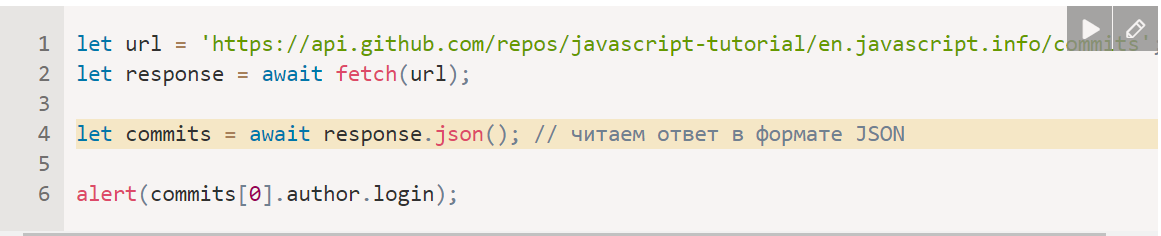
В этом примере — сделаем запрос на URL, получим ответ и распарсим его как JSON:



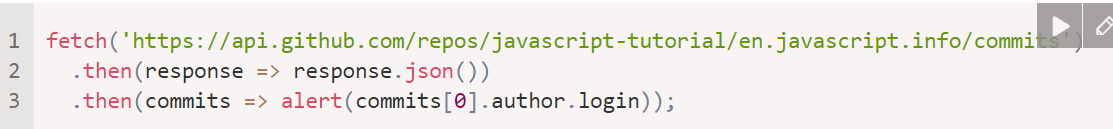
**Response** предоставляет несколько методов, основанных на промисах, для доступа к телу ответа в различных форматах:

* **response.text()** – читает ответ и возвращает как обычный текст,
* **response.json()** – декодирует ответ в формате JSON,
* **response.formData()** – возвращает ответ как объект FormData,
* **response.blob()** – возвращает объект как [Blob](https://learn.javascript.ru/blob) (бинарные данные с типом),
* **response.arrayBuffer()** – возвращает ответ как [ArrayBuffer](https://learn.javascript.ru/arraybuffer-binary-arrays) (низкоуровневое представление бинарных данных),
* помимо этого, response.body – это объект ReadableStream, с помощью которого можно считывать тело запроса по частям. Мы рассмотрим и такой пример несколько позже.

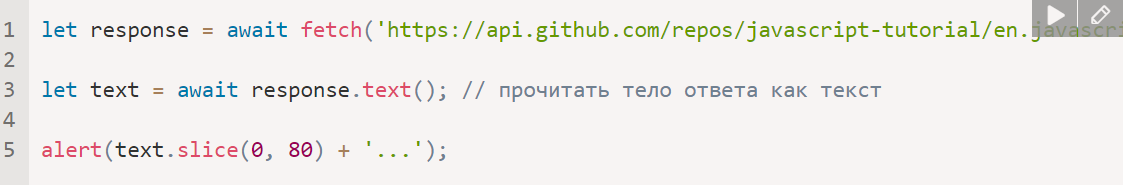
Например, получим JSON-объект с последними коммитами из репозитория на GitHub:



То же самое без await, с использованием промисов:



Для получения ответа в виде текста используем await response.text() вместо .json():



# 8. Библиотека Axios

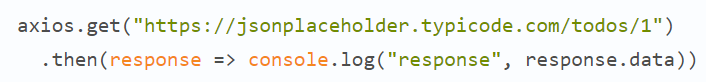
**Axios** – это библиотека с открытым исходным кодом, позволяющая делать HTTP-запросы. Она предоставляет методы .get(), .post() и .delete(). Axios работает на основе промисов для выполнения асинхронных HTTP-запросов в JavaScript. Axios работает как в браузере, так и в Node.js приложения. Axios предоставляет единый API для работы как с XHR в браузере, так и с HTTP-интерфейсом узла.

Использование Axios имеет ряд преимуществ:

* поддерживает старые браузеры
* есть способ прервать запрос
* есть способ установить время ожидания ответа
* имеет встроенную защиту от CSRF (межсайтовая подделка запроса)
* поддерживает прогресс загрузки
* выполняет автоматическое JSON преобразование данных
* работает в Node.js

ПРИМЕРЫ:

1. При работе с Axios у нас есть прямой доступ к JSON-результату в свойстве data объекта ответа.



1. Использование Axios позволяет избежать написания больших объёмов шаблонного кода и сделать код чище и понятнее. Сравними Fetch и Axios:

Fetch



Axios



1. Библиотека Axios выдаёт сведения о сетевых ошибках



# 9. Cheerio: предназначение, возможности и применение

Node.js предоставляет несколько простых методов скачивания данных из интернета через HTTP и HTTPS, но их нужно обрабатывать отдельно, не говоря уже о редиректах и других задачах, которые возникают в процессе парсинга. Модуль **Request** объединяет эти методы, позволяя абстрагироваться от рутины, и предоставляет интерфейс для создания запросов. Этот модуль используется, чтобы скачивать страницы непосредственно в память.

**Cheerio** позволяет работать со скачанными из сети данными, используя синтаксис, аналогичный jQuery (набор функций JavaScript, фокусирующийся на взаимодействии JavaScript и HTML. Библиотека jQuery помогает легко получать доступ к любому элементу DOM, обращаться к атрибутам и содержимому элементов DOM, манипулировать ими. Также библиотека **jQuery** предоставляет удобный API для работы с AJAX.).

**Cheerio** — это быстрый, гибкий и надежный порт jQuery, разработанный специально для сервера. Использование Cheerio позволяет сконцентрироваться непосредственно на работе с полученными данными, а не на их парсинге.

Что реализовано в **cheerio**:

* **.attr()** — стандартный синтаксис attr, как в jQuery:

$().attr(prop) = val или $().attr(prop, val) устанавливает свойству prop значение val;

val = $().attr(prop) читает значение свойства prop в переменную val.

Установку свойств можно комбинировать: $().attr({prop1: val1, prop2: val2}).

Дополнительно функция $().attr(prop, val) возвращает сам этот элемент, можно тут же прочитать, например, его html-код: elem\_html = $().attr(prop, val).html();

* **.html()** возвращает html-код, а .html(‘some html code’) заменяет содержимое элемента DOM-дерева переданным кодом.
* **.data()** возвращает объект с перечнем всех свойств элемента, а .data(prop) — значение конкретного свойства.
* **.addClass() и .removeClass()** работают полностью как в jQuery — добавляют и удаляют указанный класс к элементу.
* **.next() и .prev()** возвращают соседей — предыдущий и следующий элементы дерева.

Код ниже — это простое приложение, которое получает температуру воздуха с сайта с погодой.



Так, и что мы тут делаем? Сначала подключаем модули, которые будем использовать, затем сохраняем в переменной URL, содержимое которого собираемся спарсить.

После этого используем модуль Request, чтобы скачать страницу, находящуюся по URL, переданному функции request. Мы передаём в качестве аргументов URL, содержимое которого хотим скачать, и колбек, который обработает результаты запроса. Когда мы получим данные, будет вызван колбек, в который в качестве аргументов будут переданы три переменные: error, response и body. Если Request не сможет скачать страницу и получить данные, он передаст в функцию объект в переменной error и null, в качестве аргумента body. Прежде чем начинать работать с данными, проверяем не произошла ли ошибка; если произошла — выводим сообщение в консоль, чтобы увидеть, что именно пошло не так.

Если всё хорошо, передаём Cheerio полученные данные. В результате мы можем работать с данными как на обычном сайте, используя стандартный синтаксис jQuery. Чтобы найти на странице интересующие нас данные, надо написать селектор, который получит нужный элемент (или набор элементов).

**ВКРАТЦЕ,**

В браузере

1. Откройте страницу, которую хотите спарсить, сохраните URL.
2. Определите элемент или элементы, данные из которых вы хотите получить, и jQuery-селектор, который позволяет получить эти элементы.

В коде

1. Используйте Request, чтобы скачать страницу, находящуюся по выбранному вами URL.
2. Передайте полученные данные Cheerio, чтобы использовать jQuery-подобный интерфейс.
3. Используйте селектор, который вы написали перед этим, чтобы «сграбить» данные со страницы.

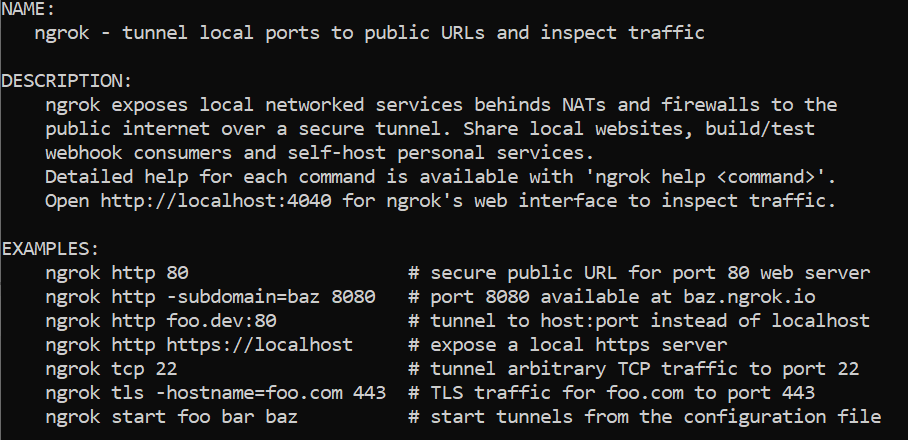
# 10. Сервис Ngrok

**Ngrok** — это платформа, которая с помощью установленной утилиты, позволяет, организовать удаленный доступ на веб-сервер или какой-то другой сервис, запущенный ПК. Доступ организуется через созданный при запуске ngrok безопасный туннель. Совсем не обязательно тащить тестовый проект куда-то ещё, можно показать его заказчику прямо с локальной машины, или, например, с помощью Ngrok можно очень легко расшарить файлы лежащие на ПК.

Туннель подключения, установленный ngrok, безопасен и может передавать данные только на localhost-открытый вами порт. Было бы сложно нанести какой-либо ущерб, но ngrok безопасен настолько, насколько безопасно приложение, которое вы тестируете.

**Для работы необходимо:**

* скачать бинарный файл для вашей системы — [ngrok](https://ngrok.com/download);
* зарегистрировать аккаунт на ngrok для получения токена;
* удостовериться, что ваш локальный сервис запущен и ожидает HTTP запросов;
* запустить ngrok (ngrok http {port}).



Для создания туннеля достаточно выполнить следующую команду: ngrok http 80, где 80 — это порт, на котором запущен ваш веб-сервер.

Ответ:



5057493e.ngrok.io — это адрес, по которому ваш локальный сервер стал доступен в интернете.

**Ngrok**, возможно, самый простой и известный localhost-сервис туннелирования, но альтернативные варианты включают:

* LocalXpose : коммерческий сервис с бесплатными опциями. Регистрация обязательна, но доступны терминальные клиенты и клиенты с графическим пользовательским интерфейсом.
* localhost.run : бесплатная служба, работающая через SSH, поэтому клиент или регистрация не требуются.
* localtunnel : клиент Node.js с открытым исходным кодом. Регистрация не требуется.

# 11. Библиотека Socket.io: её предназначение, применение и возможности

Socket.IO – это библиотека JavaScript для веб-приложений реального времени . Он обеспечивает двустороннюю связь в реальном времени между веб-клиентами и серверами. Он состоит из двух частей: клиентской библиотеки, которая запускается в браузере, и серверной библиотеки для node.js. Оба компонента имеют идентичный API.

***Применяется в:***

Мессенджеры – приложения для чата, такие как Whatsapp, Facebook Messenger и т. Д. Вам не нужно обновлять свое приложение / веб-сайт для получения новых сообщений.

Push-уведомления – когда кто-то помечает вас на фотографии в Facebook, вы сразу получаете уведомление.

Приложения для совместной работы – приложения, такие как Google Docs, которые позволяют нескольким людям одновременно обновлять одни и те же документы и вносить изменения во все экземпляры людей.

Это одна из самых мощных JavaScript-фреймворков на GitHub и наиболее зависимая от модуля NPM (Node Package Manager). Socket.IO также имеет огромное сообщество, что означает, что найти помощь довольно легко.

Сокеты работают на основе событий. Существуют некоторые зарезервированные события, доступ к которым можно получить с помощью объекта сокета на стороне сервера.

Объект сокета на стороне клиента также предоставляет некоторые зарезервированные события.

Вещание означает отправку сообщения всем подключенным клиентам. Вещание может быть сделано на нескольких уровнях.

# 12. Bootstrap – как и для чего применяется данная технология

Bootstrap — свободный набор инструментов для создания сайтов и веб-приложений. Включает в себя HTML- и CSS-шаблоны оформления для типографики, веб-форм, кнопок, меток, блоков навигации и прочих компонентов веб-интерфейса, включая JavaScript-расширения.

Основные инструменты Bootstrap:

* **Сетки** — заранее заданные размеры колонок, которые можно сразу же использовать, например, ширина колонки 140 px относится к классу .span2 (.col-md-2 в третьей версии фреймворка), который можно использовать в CSS-описании документа.
* **Шаблоны** — фиксированный или резиновый шаблон документа.
* **Типографика** — описания шрифтов, определение некоторых классов для шрифтов, таких как код, цитаты и т. п.
* **Медиа** — предоставляет некоторое управление изображениями и видео.
* **Таблицы** — средства оформления таблиц, вплоть до добавления функциональности сортировки.
* **Формы** — классы для оформления форм и некоторых событий, происходящих с ними.
* **Навигация** — классы оформления для панелей, вкладок, перехода по страницам, меню и панели инструментов.
* **Алерты** — оформление диалоговых окон, подсказок и всплывающих окон.

Пример добавления дефолтного заголовка:

<header class="d-flex flex-wrap justify-content-center py-3 mb-4 border-bottom">

<a href="/" class="d-flex align-items-center mb-3 mb-md-0 me-md-auto text-dark text-decoration-none">

<svg class="bi me-2" width="40" height="32"><use xlink:href="#bootstrap"></use></svg>

<span class="fs-4">Simple header</span>

</a>

<ul class="nav nav-pills">

<li class="nav-item"><a href="#" class="nav-link active" aria-current="page">Home</a></li>

<li class="nav-item"><a href="#" class="nav-link">Features</a></li>

<li class="nav-item"><a href="#" class="nav-link">Pricing</a></li>

<li class="nav-item"><a href="#" class="nav-link">FAQs</a></li>

<li class="nav-item"><a href="#" class="nav-link">About</a></li>

</ul>

</header>

# 13. Стрелочные функции языка JavaScript

Существует ещё один очень простой и лаконичный синтаксис для создания функций, который часто лучше, чем Function Expression. Он называется «стрелочные функции» (arrow functions), т.к. выглядит следующим образом:

let func = (arg1, arg2, ...argN) => expression;

Это создаёт функцию func, которая принимает аргументы arg1..argN, затем вычисляет expression в правой части с их использованием и возвращает результат. Другими словами, это сокращённая версия:

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

return expression;

};

пример:

let sum = (a, b) => a + b;

(a, b) => a + b задаёт функцию, которая принимает два аргумента с именами a и b. И при выполнении она вычисляет выражение a + b и возвращает результат.

Если только один аргумент, то круглые скобки вокруг параметров можно опустить, сделав запись ещё короче:

let double = n => n \* 2;

Если аргументов нет, круглые скобки будут пустыми, но они должны присутствовать:

let sayHi = () => alert("Hello!");

sayHi();

Многострочные стрелочные функции

Иногда нам нужна более сложная функция, с несколькими выражениями и инструкциями. Это также возможно, нужно лишь заключить их в фигурные скобки. При этом важное отличие – в том, что в таких скобках для возврата значения нужно использовать return (как в обычных функциях).

let sum = (a, b) => { // фигурная скобка, открывающая тело многострочной функции let result = a + b;

return result; // если мы используем фигурные скобки, то нам нужно явно указать "return"};

alert( sum(1, 2) ); // 3

# 14. Промис (promise) – понятие и примеры использования

Часто встречаются задачи, для выполнения которых требуется значительное время. Если их реализовать с помощью синхронного кода, то это может привести к тому, что страницы будут подвисать. Чтобы этого избежать необходимо использовать асинхронный код. Он в отличие от синхронного выполняется в фоновом режиме и не блокирует основной поток. Асинхронный код в JavaScript может быть написан разными способами: с помощью обратных вызовов, ключевых слов async/await, а также с помощью Promise (обещаний).

Promise (промис) - это объект, представляющий результат успешного или неудачного завершения асинхронной операции.

const promise = new Promise((resolve, reject) => {

// асинхронный код

});

Конструктор промиса принимает 2 аргумента, которые являются функциями. Внутрь промиса помещают асинхронный код. Промис завершает своё выполнение, когда вызывается функция resolve() или reject(). Функцию resolve() вызывают обычно в том месте кода, в котором асинхронная операция должна завершиться успешно. А функцию reject() – там, где она должна завершиться с ошибкой. Во время выполнения кода внутри промиса он получает статус Pending (состояние ожидания).

Также с промисами используются методы:

* then – выполняется, когда промис завершился успешно (после вызова функции resolve());
* catch – вызывается, если промис завершается ошибкой (после вызова reject());
* finally – выполняется в любом случае после завершения промиса, вне зависимости от конечного состояния.

Ещё небольшой пример:

let promise = new Promise((resolve, reject) => {

setTimeout(() => {

// переведёт промис в состояние fulfilled с результатом "result"

resolve("result");

}, 1000); });

# 15. Колбэки (callbacks) – понятие и примеры использования

Колбэк-функция (или обратный вызов) - это функция, переданная в другую функцию в качестве аргумента, которая затем вызывается по завершению какого-либо действия.

Вот краткий пример:

function greeting(name) {

alert('Hello ' + name);

}

function processUserInput(callback) {

var name = prompt('Please enter your name.');

callback(name);

}

processUserInput(greeting);

Выше приведён пример синхронного колбэка, поскольку функция processUserInput выполняется синхронно.

Колбэки часто используются для продолжения выполнения кода после завершения асинхронной операции - они называются асинхронными колбэками.

async function pageLoader(callback) {

const data = await fetch('/ru/docs/Glossary/Callback\_function')

callback(data)

}

function onPageLoadingFinished(pageData) {

console.log('Page was sucessfully loaded!')

console.log('Response:')

console.log(pageData)

}

pageLoader(onPageLoadingFinished)

# 16. Отладка программного кода JavaScript на стороне клиента и сервера

Методы отладки в js:

1. Метод console.log()

2. Точки останова в ide

3. Ключевое слово **debugger** останавливает выполнение скрипта JavaScript и вызывает (если есть) функцию отладки. Если отладка не доступна, то выражение **debugger** ничего не делает.

4. Если вы не знаете точно, выполнен ли запрос к серверу, перейдите к вкладке Network инструментов разработчика Chrome. Посмотрите на список вызовов для запроса, в котором вы не уверены. Вы можете проверить код состояния запроса, просмотреть заголовки запроса и другие сведения о нём.

5. Node Inspect похож на инструменты разработчика Chrome, но предназначен он для серверного кода.

6. Если вы уверены в том, что запрос отправлен на сервер, но вы не знаете точно, как выглядит то, что пришло в ответ, или даже в том, пришло ли что-нибудь вообще, разобраться в ситуации вам поможет Postman

7. Если ваше приложение основано на React и нужно проверить значения свойств или состояний, вам стоит познакомиться с расширением React Developer Tools для Chrome. Добавив в Chrome это расширение и перейдя на страницу, созданную с помощью React, вы увидите в консоли разработчика вкладку React, которая выводит значения свойств и состояния для элемента, по которому вы щёлкнете.

# 17. GET и POST-запросы в веб-программировании

GET - это самый простой тип HTTP-запроса; которым браузер пользуется каждый раз, когда вы нажимаете ссылку или вводите URL-адрес в адресную строку. Он инструктирует сервер передавать клиенту данные, идентифицированные URL-адресом. Никогда не последует изменений данных на стороне сервера в результате запроса GET. В этом смысле GET-запрос доступен только для чтения, но, конечно, как только клиент получит данные, он может самостоятельно выполнять любые операции с ними, например, форматировать для отображения.

Запрос PUT используется, когда вы хотите создать или обновить ресурс, указанный URL-адресом. POST используется, когда обработка, которую вы хотите выполнить на сервере, должна повторяться, если запрос POST повторяется (то есть, они не являются *идемпотентными*)

# 18. Фреймворк NestJS

Nest (NestJS) - это фреймворк для построения эффективных, масштабируемых Node.js приложений на сервере. Он использует современный JavaScript, построен на основе и полностью поддерживает TypeScript (но при этом позволяет разработчикам писать на чистом JavaScript) и сочетает в себе элементы ООП (объектно-ориентированного программирования), ФП (функционального программирования) и ФРП (функционально-реактивного программирования).

Под капотом Nest использует надежные фреймворки HTTP-серверов, такие как Express (по умолчанию) и, по желанию, может быть настроен на использование Fastify

Nest обеспечивает уровень абстракции над этими распространенными фреймворками Node.js (Express/Fastify), но также предоставляет их API непосредственно разработчику. Это дает свободу в использовании огромного количества сторонних модулей, которые доступны для базовой платформы.

# 19. Язык программирования TypeScript, его отличия от JavaScript

JavaScript является одной из самых популярных основных технологий Интернета. С самого начала он был неотъемлемой частью веб-приложений, делающих веб-страницы интерактивными и динамичными. Это язык высокого уровня, с компилятором JIT (Just-in-Time) и динамической типизацией. Долгое время JS был только реализацией на стороне клиента, но некоторые новые движки JS также имеют реализации на стороне сервера.

TypeScript ничем не отличается от JavaScript по своему назначению, но используется для разработки больших приложений. TypeScript trans компилирует (компиляция из источника в источник) в JavaScript. Он следует структуре объектно-ориентированного языка программирования и поддерживает такие функции, как классы, интерфейсы, пространства имен и наследование. Статическая типизация возможна в TypeScript через аннотации типов (numbers, string и boolean).

TypeScript сильно типизирован. Это улучшает отладку (во время компиляции), что является более эффективным способом кодирования для больших проектов. Программы TypeScript обычно состоят из модулей, функций, переменных, комментариев, выражений и операторов – как и любой другой полноценный язык программирования.

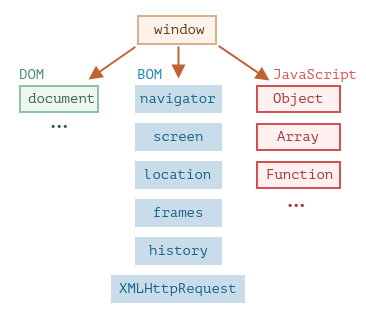
Когда код становится огромным, сложным в обработке и подвержен ошибкам, лучше, если ошибки будут обнаружены во время самой компиляции. Вот где помогает TypeScript.

# 20. Браузерное окружение и DOM

Сегодня JavaScript может использоваться в браузере, на веб-сервере или в какой-то другой среде, даже в кофеварке. Каждая среда предоставляет свою функциональность, которую спецификация JavaScript называет *окружением*.

Окружение предоставляет свои объекты и дополнительные функции, в дополнение базовым языковым. Браузеры, например, дают средства для управления веб-страницами. Node.js делает доступными какие-то серверные возможности и так далее.

На картинке ниже в общих чертах показано, что доступно для JavaScript в браузерном окружении:



Как мы видим, имеется корневой объект window, который выступает в 2 ролях:

1. Во-первых, это глобальный объект для JavaScript-кода
2. Во-вторых, он также представляет собой окно браузера и располагает методами для управления им.

Document Object Model, сокращённо DOM – объектная модель документа, которая представляет все содержимое страницы в виде объектов, которые можно менять.

Объект document – основная «входная точка». С его помощью мы можем что-то создавать или менять на странице. Основой HTML-документа являются теги. В соответствии с объектной моделью документа («Document Object Model», коротко DOM), каждый HTML-тег является объектом. Вложенные теги являются «детьми» родительского элемента. Текст, который находится внутри тега, также является объектом. Все эти объекты доступны при помощи JavaScript, мы можем использовать их для изменения страницы.

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

* Объект navigator даёт информацию о самом браузере и операционной системе. Среди множества его свойств самыми известными являются: navigator.userAgent – информация о текущем браузере, и navigator.platform – информация о платформе (может помочь в понимании того, в какой ОС открыт браузер – Windows/Linux/Mac и так далее).
* Объект location позволяет получить текущий URL и перенаправить браузер по новому адресу.

# 21. Браузерные события

*Событие* – это сигнал от браузера о том, что что-то произошло. Все DOM-узлы подают такие сигналы (хотя события бывают и не только в DOM). Бывают события мыши, события клавиатуры, события элементов управления, события документа и т.д. Событию можно назначить *обработчик*, то есть функцию, которая сработает, как только событие произошло. Именно благодаря обработчикам JavaScript-код может реагировать на действия пользователя.

Есть несколько способов назначить событию обработчик: использование атрибута html, свойства dom-объекта, доступ через this. Фундаментальный недостаток описанных выше способов назначения обработчика – невозможность повесить несколько обработчиков на одно событие. Разработчики стандартов достаточно давно это поняли и предложили альтернативный способ назначения обработчиков при помощи специальных методов addEventListener и removeEventListener. Чтобы хорошо обработать событие, могут понадобиться детали того, что произошло. Не просто «клик» или «нажатие клавиши», а также – какие координаты указателя мыши, какая клавиша нажата и так далее. Когда происходит событие, браузер создаёт *объект события*, записывает в него детали и передаёт его в качестве аргумента функции-обработчику. Мы можем назначить обработчиком не только функцию, но и объект при помощи addEventListener. В этом случае, когда происходит событие, вызывается метод объекта handleEvent.

# 22. Куки, LocalStorage и IndexedDB

LocalStorage:

* Хранилище ключей и значений, хранящее значения в виде строк
* Не имеет даты истечения срока действия (постоянное хранилище), если явно не очистить браузер с помощью настроек или Javascript
* Можно хранить до 10 МБ данных
* Следуйте политике того же происхождения, что означает, что протокол (Http / Https), порт и хост одинаковы. Только скрипты одного и того же происхождения могут получить доступ к данным LocalStorge
* Не отправлять на сервер, только для использования на стороне клиента

Может использоваться для хранения пользовательских данных

Session Storage:  
Хранилище ключей и значений, хранящее значения в виде строк

* Данные не сохраняются после закрытия таблицы / окна
* Можно хранить до 10 МБ данных
* Привязаны к вкладке
* Не отправлять на сервер, только для использования на стороне клиента

хранить пользовательские данные только для одного сеанса, например, выбор языка

Куки:

* Хранилище ключевых значений, которое хранит значения в виде строки
* Есть время истечения срока действия, если не указано время истечения срока действия, то cookie истечет в конце сеанса браузера.
* Можно хранить до 4 КБ данных
* Отправляйте на сервер для каждого запроса

Сервер может получать данные из cookie для отслеживания состояния сеанса.

IndexedDB:

* Может хранить как объекты, так и пары ключ-значение
* До 250 МБ для IE
* API IndexedDB является асинхронным, в отличие от локального хранилища и хранилища сеансов. Операции IndexedDB управляются различными событиями, такими как onsuccess, onerror, oncomplete и т. Д.
* У них нет времени истечения срока действия (постоянное хранилище), если не указано явное удаление

Когда необходимо хранить большое количество объектов, что отнимает много времени и снижает производительность, чтобы каждый раз преобразовывать в строку для локального хранилища.

# 23. Протокол WebSocket («веб-сокет»)

Протокол WebSocket («веб-сокет») обеспечивает возможность обмена данными между браузером и сервером через постоянное соединение. Данные передаются по нему в обоих направлениях в виде «пакетов», без разрыва соединения и дополнительных HTTP-запросов.

WebSocket особенно хорош для сервисов, которые нуждаются в постоянном обмене данными, например онлайн игры, торговые площадки, работающие в реальном времени, и т.д.

Как только объект WebSocket создан, мы должны слушать его события. Их всего 4:

* **open** – соединение установлено,
* **message** – получены данные,
* **error** – ошибка,
* **close** – соединение закрыто.

…А если мы хотим отправить что-нибудь, то вызов socket.send(data) сделает это.

Когда new WebSocket(url) создан, он тут же сам начинает устанавливать соединение.

Браузер, при помощи специальных заголовков, спрашивает сервер: «Ты поддерживаешь Websocket?» и если сервер отвечает «да», они начинают работать по протоколу WebSocket, который уже не является HTTP.

Могут быть дополнительные заголовки Sec-WebSocket-Extensions и Sec-WebSocket-Protocol, описывающие расширения и подпротоколы.

* Sec-WebSocket-Extensions: deflate-frame означает, что браузер поддерживает сжатие данных. Расширение – это что-то, связанное с передачей данных, расширяющее сам протокол WebSocket. Заголовок Sec-WebSocket-Extensions отправляется браузером автоматически со списком всевозможных расширений, которые он поддерживает.
* Sec-WebSocket-Protocol: soap, wamp означает, что мы будем передавать не только произвольные данные, но и данные в протоколах [SOAP](http://en.wikipedia.org/wiki/SOAP) или WAMP (The WebSocket Application Messaging Protocol" – «протокол обмена сообщениями WebSocket приложений»). То есть, этот заголовок описывает не передачу, а формат данных, который мы собираемся использовать. Официальные подпротоколы WebSocket регистрируются в [каталоге IANA](http://www.iana.org/assignments/websocket/websocket.xml).  
    
  Этот необязательный заголовок ставим мы сами, передавая массив подпротоколов вторым параметром new WebSocket. Сервер должен ответить перечнем протоколов и расширений, которые он может использовать.

Поток данных в WebSocket состоит из «фреймов», фрагментов данных, которые могут быть отправлены любой стороной, и которые могут быть следующих видов:

* «текстовые фреймы» – содержат текстовые данные, которые стороны отправляют друг другу.
* «бинарные фреймы» – содержат бинарные данные, которые стороны отправляют друг другу.
* «пинг-понг фреймы» используется для проверки соединения; отправляется с сервера, браузер реагирует на них автоматически.
* также есть «фрейм закрытия соединения» и некоторые другие служебные фреймы.

В браузере мы напрямую работаем только с текстовыми и бинарными фреймами.

# 24. Рендеринг на клиенте, на сервере и генерация статических сайтов

Существует два наиболее распространенных способа обработки данных (рендеринга) на сайтах:

* На сервере
* На клиенте

Также можно обойтись вообще без рендеринга, просто использовав статический сайт - простой текстовый .html документ

**Рендеринг на сервере**

Приложения с серверным рендерингом отдают готовые к рендеру HTML страницы. Такие приложения заранее выполняют все необходимые вызовы API и передают все необходимые данные в изначальном запросе. Это значит, что браузер сразу же получает все, что нужно для рендеринга приложения, что сокращает время до первого взаимодействия с пользователем.

***Преимущества рендеринга на сервере***

1. Ускоренная загрузка. Приложения с рендерингом на стороне сервера, загружаются быстрее, чем похожие приложения, которые рендерятся на клиенте. А поскольку сервер выполняет наиболее затратную часть работы, они также быстро загружаются на менее производительных устройствах.

***Недостатки рендеринга на сервере***

1. Дорого хостить, повышенная нагрузка на сервер  
   По сравнению с приложениями, которые рендерятся на стороне клиента, хостинг приложений с серверным рендерингом стоит дороже. В результате каждого запроса к вашему серверу он должен будет выполнять вызовы API, а затем рендерить HTML перед его передачей на сторону клиента.
2. Более сложная разработка и коммуникация между беком и фронтом  
   Самостоятельная настройка рендеринга на стороне сервера с использованием React может оказаться непростой задачей. Однако это становится намного проще, если использовать один из предназначенных для этого фреймворков, например NextJS.
3. Никто так в 2022 году не делает.

**Рендеринг на клиенте**

На клиенте все данные обрабатываются на машине пользователя, который зашел на определенный web-сайт с помощью JS (который, собственно, всегда отрабатывает на стороне клиента). В нынешнее время большинство сайтов используют рендеринг на стороне клиента (поэтому, если выключить выполнение JS в браузере, ничего работать не будет).

Такой способ рендеринга стал популярнее с расцветом технологии Single Page Application (Spa). Этот подход применяется во многих JavaScript фреймворках, например AngularJS, ReactJS, Backbone.JS и др. В приложениях с генерацией на клиенте сервер отправляет JS файлы и статичный HTML на сторону клиента. Клиент делает Http-запросы и рендерит приложение.

***Преимущества рендеринга на клиенте***:

1. Приложения легко разрабатывать и хостить. Вам не нужен сервер для приложений, рендерящихся на стороне клиента. Можно просто разместить свое приложение на любом файловом хостинге по типу Github Pages или Amazon.
2. Не нужно перезагружать страницы для отображения обновленных данных. Пользователи могут переходить по страницам без серии обращений к серверу. Из-за этого создаётся ощущение высокой скорости работы, почти как у нативного приложения.

***Недостатки рендеринга на клиенте:***

1. Дольше грузятся. Приложения с рендерингом на стороне клиента выполняют дополнительный запрос к серверу с API для рендеринга, что увеличивает время открытия приложения
2. Медленная работа на медленных устройствах. На слабых по производительности устройствах рендеринг на клиенте может замедлить загрузку на несколько секунд.

**Генерация статических сайтов**

Генераторы статических сайтов работают, генерируя все HTML-файлы для сайта во время сборки. Сервер заранее выполняет вызовы API и генерирует статические HTML-файлы для каждой страницы вашего сайта. Это значит, что когда клиент запрашивает одну из веб-страниц, серверу не нужно обращаться к API или рендерить HTML, ему нужно только вернуть предварительно подготовленную HTML-страницу.

***Преимущества создания статических сайтов***

1. Быстрая загрузка. Поскольку HTML-файлы уже скомпилированы и готовы к использованию, статические сайты загружаются быстрее, чем сайты, с рендерингом на стороне клиента, и сайты с рендерингом на сервере.

***Недостатки создания статических сайтов***

1. Подойдет лишь только для сайтов, где данные меняются очень редко, например, для непопулярных блогов

# 25. Популярные CMS: 1С-Битрикс, WordPress, Joomla! и другие.

**1С-Битрикс**

«1С-Битрикс: Управление сайтом» - это профессиональная система управления веб-проектами, универсальный программный продукт для создания, поддержки и успешного развития: корпоративных сайтов, интернет-магазинов и пр.

Технология «Композитный сайт» объединяет все преимущества статического сайта с возможностями динамического. Статический сайт – это готовая HTML-страница, не содержащая программного кода. Обычно очень быстро загружается для пользователя, но в современном сайтостроении используется редко. Динамические сайты используют программный код, поэтому скорость загрузки сайта у пользователей значительно медленнее. «Композитный сайт» обеспечивает максимально высокую скорость отображения динамического сайта за счет отображение кешированных копий страниц сайта и отображения обновленных данных в местах, где они претерпели изменения с момента кеширования.

**WordPress**

От других систем управления содержимым сайта WordPress отличается предельно простым интерфейсом. Программисту это, возможно, не гарантирует всей полноты удобств, зато администратору веб-ресурса значительно облегчает работу. Публикации создаются моментально, сама платформа устанавливается не более чем за 5 минут.

WordPress распространяется в сети свободно и бесплатно, весит несколько мегабайт. Плюсы: интеграция сайта со сторонними сервисами (например, LiveJournal), простая установка и настройка, поддерживаются стандарты XHTML и CSS.

Недостаток платформы в том, что без предварительной установки специальных плагинов невозможно восстанавливать удаленные объекты.

**Joomla!**

Платформа предоставляет минимальный набор инструментов для администрирования. Расширить его можно, скачав дополнительное программное расширение (Joomla! предлагает более 8000 продуктов). Панель не дает тяжелой нагрузки на сервер и экономит место на хостинге.

Главные достоинства платформы: простота управления, надёжность, доступность. CMS написана на языках JavaScript и PHP. Бесплатно распространяется со стандартной общественной лицензией GPL.  
Возможности: Быстрая смена шаблонов, создание неограниченного количества страниц и т.д.

Преимущества:

* Простота настройки и управления.
* Поддержка более 60-и языков.
* Четкая структура, понятное главное меню.

Недостатки:

* Нужно регулярно обновлять.
* Слишком проста для некоторых задач

**Tilda**

Tilda Publishing — блочный конструктор сайтов, не требующий навыков программирования. Позволяет создавать сайты, интернет-магазины, посадочные страницы, блоги и email-рассылки.

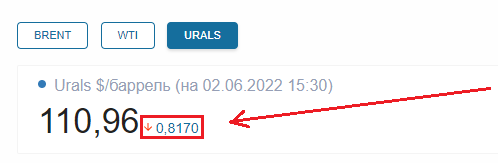
Cайты на платформе собираются из готовых блоков, которые автоматически адаптируются под мобильные устройства и выделены в смысловые категории (например, обложка сайта, меню, форма, текст, изображение). Также помощью встроенного редактора Zero Block можно спроектировать свой собственный блок, добавляя отдельные модули (текст, форма, геометрическая фигура, изображение), располагая их на экране в режиме Drag-and-drop.

# Примеры практических заданий в билетах

1. Создайте форму авторизации (поля «Логин» и «Пароль», кнопка «Войти»), отправляющая при нажатии на кнопку «Войти» данные (логин и пароль) на сервер по адресу http://localhost:8085/server. Если пользователь забыл ввести логин или пароль, попросить его сделать это. Добавить стилизацию элементов формы с помощью Bootstrap или другими средствами.
2. Разработайте серверное приложение, расположенное по адресу http:// localhost:8085/server, которое может:
   1. принимать от клиента данные (логин и пароль)
   2. сравнивать входящие логин и пароль с логином и паролем, хранящимися в базе данных MongoDB, PostgreSQL или SQLite (на выбор студента). Поиск в базе происходит по логину.
   3. возвращать на клиент строку «success» (в случае совпадения входящих и хранящихся логинов-паролей) или «fail» (в случае расхождения входящих и хранящихся логинов-паролей).

2

1. Создайте клиентскую часть сервиса для мониторинга актуальных цен на нефть различных марок: URALS, BRENT, WTI. Можно воспользоваться ресурсом <https://maanimo.ua/ru/oils> или подобными ему ресурсами. На форме должно быть поле выбора марки нефти (URALS, BRENT, WTI), поле для вывода текущей цены и поле для вывода динамики (разницы цен между текущей ценой и ценой предыдущего дня, выделен красным на рисунке, используется сайт <https://maanimo.ua/ru/oils>).



1. Реализуйте серверную часть сервиса для мониторинга актуальных цен на нефть различных марок: URALS, BRENT, WTI. При выборе марки нефти пользователем на клиентской форме сервер проверяет, есть ли уже загруженные данные в локальной базе (MongoDB, PostgreSQL или SQLite), если есть – текущая цена и динамика за последние сутки – передаются на клиент и выводятся в соответствующие поля на форме, если нет – данные загружаются напрямую с сайта, передаются на клиент и выводятся на форме.

3

1. Создайте клиентскую часть сервиса, загружающего данные курсов валют с сайта <https://www.cbr-xml-daily.ru/> и отображающего результат в таблице из 3-х столбцов: Валюта, Курс сегодня, Курс вчера. В таблице должно быть не менее 5 различных валют. Сделайте оформление таблицы в отдельном CSS-файле.
2. Реализуйте серверную часть сервиса, которая будет отправлять готовый html-файл из предыдущей задачи. Предусмотрите использование относительных ссылок на html-файл c помощью модуля PATH. Подключите Ngrok или ему подобные инструменты для вывода сервиса в глобальную сеть.