WEB

- SPA vs MPA
  - pros/cons
- SSR
  - pros/cons

1. Оптимизация:

производительность

- repaint/reflow
- Критические этапы рендеринга critical render path
  - Загрузка веб-страницы или приложения начинается с запроса HTML
  - HTML преобразуется в DOM-дерево
  - Ссылки на внешние ресурсы порождают новые запросы HTML: файлы стилей, скриптов, ссылки на изображения.
  - Некоторые запросы являются блокирующими
  - браузер конструирует CSSOM модель.
  - CSS-правила ниспадают каскадом. вложенные узлы наследуют стили от родительских.
  - Если формирование DOM инкрементально, CSSOM - нет. CSS блокирует рендер. Инкрементальная обработка недоступна для CSS, потому что набор следующих правил может перезаписать предыдущие. Это может привести к лишнему вызову компоновки и перерасчёта стилей.
  - браузер строит дерево рендера (render tree), в котором вычисляет стили для каждого видимого элемента страницы.
  - компоновка (layout), которая определяет положение и размеры элементов этого дерева.
  - страница рендерится. Или "отрисовывается" (paint) на экране.
- профилирование и оптимизация
  - JS/CSS/DOM/SVG/Шрифты/Файлы
  - https://developer.mozilla.org/en-US/docs/Learn/Performance/CSS
  - https://developer.mozilla.org/en-US/docs/Learn/Performance/Multimedia
  - https://developer.mozilla.org/en-US/docs/Learn/Performance/JavaScript
  - https://developer.mozilla.org/en-US/docs/Learn/Performance/HTML
  - сеть
  - утечки памяти
- отладка
  - JS/CSS/DOM
  - сеть
- обфускация и минификация
  - JS/CSS/HTML
- Долгие вычисления Long computations
- Framework optimization techniques (Angular, React, or others)
- RAIL
- SVG vs canvas (with prior experience)

- Service workers / Web workers

Сеть

- DNS
- Тело HTTP-запроса
- Передача данных
- Отправка форм
- User-agent
- Transfer-Encoding
- Перенаправления
- Cookies
- Протоколы
    - HTTP vs HTTPS vs HTTP/2
    - WS vs Long Polling
    - RESTful vs RPC (JSON RPC) vs GraphQL
- long pooling - держим соединение с сервером, пока он не ответит, затем переподключаемся
- quick pooling - пингуем сервер
- https://html.spec.whatwg.org/multipage/comms.html#the-eventsource-interface Messages in server-sent events, cross-document messaging, channel messaging, broadcast channels, and WebSockets use the MessageEvent interface for their message events
- WEBSOCKET - двунаправленный, с сохранением соединения, с подписками на события, быстрый, есть шифрование wss
- gRPC - преимущества двоичных сообщений, высокую производительность и низкий уровень использования сети, одно/двух направленный, синхронный/асинхронный, сохранение/сброс соединения, тип данных Buffer/stream
- GRAPHQL - язык запросов для формирования пакетов данных, требует доработок в тылу
- SOAP - обмен xml сообщениями с валидацией/схемами, атомарными - без хранения состояний. Подходит для ИБ, по сути MQ.
- RESTful HTTP - http/https однонаправленный pull открывает/закрывает соединение на каждый запрос, чаще всего текстовые данные, разные методы для разных видов запросов
- WEBRTC

HTTP REST

- CRUD
    - create - post
    - read - get
    - update - put
    - delete - delete
- https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods
- https://httpwg.org/specs/rfc9110.html#method.overview
- методы
    - GET Transfer a current representation of the target resource.
    - HEAD нет тела, Same as GET, but do not transfer the response content.
    - POST - можно много записей за раз, не идемпотентный(разный результат), добавление или замена значения

    - PUT - подходит для конкретного экземпляра, идемпотентный, замена значения
    - DELETE Remove all current representations of the target resource.
    - CONNECT Establish a tunnel to the server identified by the target resource.
    - OPTIONS Describe the communication options for the target resource.
    - TRACE Perform a message loop-back test along the path to the target resource.
- GET, HEAD, OPTIONS, and TRACE methods are defined to be safe

websocket

- Двунаправленность: и сервер, и клиент могут обмениваться сообщениями
- Бинарные и текстовые данные
- Протокол WebSocket
- https://learn.javascript.ru/websocket
-

```
    let socket = new
WebSocket("wss://javascript.info/article/websocket/demo/hello");

    socket.onopen = function(e) {
        alert("[open] Соединение установлено");
        alert("Отправляем данные на сервер");
        socket.send("Меня зовут Джон");
    };

    socket.onmessage = function(event) {
        alert(`[message] Данные получены с сервера: ${event.data}`);
    };

    socket.onclose = function(event) {
    if (event.wasClean) {
        alert(`[close] Соединение закрыто чисто, код=${event.code}
причина=${event.reason}`);
    } else {
        // например, сервер убил процесс или сеть недоступна
        // обычно в этом случае event.code 1006
        alert('[close] Соединение прервано');
    }
    };

    socket.onerror = function(error) {
        alert(`[error] ${error.message}`);
    };
```

-

quick/short pooling

- задержки между опросами
- много пустых запросов

long pooling

- алгоритм
  - Запрос на сервер
  - Сервер не закрывает соединение
  - сервер отвечает на запрос, Браузер немедленно делает новый
- плюсы
  - мало кода, не нужны библиотеки
  - хорошо подходит nodejs
- минусы
  - требует держать много соединений, оптимизировать под это
- https://learn.javascript.ru/long-polling
-

```
    async function subscribe() {
    let response = await fetch("/subscribe");

    if (response.status == 502) {
        // Статус 502 - это таймаут соединения;
        // возможен, когда соединение ожидало слишком долго
        // и сервер (или промежуточный прокси) закрыл его
        // давайте восстановим связь
        await subscribe();
    } else if (response.status != 200) {
        // Какая-то ошибка, покажем её
        showMessage(response.statusText);
        // Подключимся снова через секунду.
        await new Promise(resolve => setTimeout(resolve, 1000));
        await subscribe();
    } else {
        // Получим и покажем сообщение
        let message = await response.text();
        showMessage(message);
        // И снова вызовем subscribe() для получения следующего
сообщения
        await subscribe();
    }
    }

    subscribe();
```

-

server sent events/eventSource

- Однонаправленность: данные посылает только сервер
- Только текст
- Обычный HTTP
- https://learn.javascript.ru/server-sent-events

```
let eventSource = new EventSource("/events/subscribe");

eventSource.onmessage = function(event) {
console.log("Новое сообщение", event.data);
// этот код выведет в консоль 3 сообщения, из потока данных выше
};

// или eventSource.addEventListener('message', ...)
```

Сборщики

- NPM и YARN в чем разница?
- Как достать конфигурацию из Angular CLI с помощью Webpack.
- новинки webpack

CSS

- Пре/пост процессоры
  - LESS, SASS/SCSS, Stylus, PostCSS, tailwind
- Методологии
  - BEM, OOCSS, SMACSS, ITCSS, Atomic CSS
- семантичная вёрстка
- Медиа запросы
  - `@media media-type and (media-feature-rule) {`
  - media-type: all, print, screen
  - and/not/or
    - `@media not all and (min-width: 600px), screen and (orientation: landscape) {`
  - media-feature-rule
    - описывают свойства user-agent, устройства или окружения
- типовые раскладки
- [отзывчивый дизайн раскладок](https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout/Responsive_Design
  - статическая/overflow
  - резиновая/Responsive
  - адаптивная
  - Mobile/Desktop First
- Progressive Enhancement vs Graceful Degradation approaches
- Анимация
  - типы: js/css/framework
  - CSS анимации
- CSS переменные
- colors
- fonts
- Блочная модель box-model
- Позиционирование

    - static
        - не работают top, right, bottom, left, z-index
        - позволяет элементу находиться в обычном его состоянии, расположенном на своём месте в документе
        - значение по умолчанию.
    - relative
        - работают top, right, bottom, left, z-index
        - новый стыкующийся контекст если z-index !== auto
    - absolute
        - относительно relative или absolute
        - Если элемент имеет поля, они добавляются к смещению
        - выход из основного потока
    - fixed
        - как absolute
        - игнорирует прокрутку
    - sticky
        - прилипает(игнорирует прокрутку) на значение не менее установленных top, right, bottom, left
        - гибрид относительного и фиксированного позиционирования
    - Позиционируемый элемент — все, кроме static
    - Относительно позиционируемый элемент relative
    - Абсолютно позиционируемый элемент — absolute или fixed
    - Элемент с липкой позицией — sticky
- display
    - none
    - inline
    - inline-block
    - block
    - flex
    - grid
    - table
- подходы к кроссбраузерной вёрстке: сброс и нормализация(более предсказуемый вид ранее не использованных тэгов)
- https://developer.mozilla.org/en-US/docs/Web/Web_Components/Using_shadow_DOM

HTML

- iframe - браузер в браузере, browsing context, своя история сессии
- семантичная вёрстка
- [https://developer.mozilla.org/en-US/docs/Web/API/Intersection_Observer_API](Intersection Observer API) lets code register a callback function that is executed whenever an element they wish to monitor enters or exits another element (or the viewport)
- mutation observer
    - https://developers.google.com/web/updates/2012/02/Detect-DOM-changes-with-Mutation-Observers

```
    //Here's an example of listing inserted nodes with Mutation
Events:

    var insertedNodes = [];
    document.addEventListener("DOMNodeInserted", function(e) {
        insertedNodes.push(e.target);
    }, false);
    console.log(insertedNodes);

    //And here's how it looks with Mutation Observers:

    var insertedNodes = [];
    var observer = new MutationObserver(function(mutations) {
        mutations.forEach(function(mutation) {
        for (var i = 0; i < mutation.addedNodes.length; i++)
            insertedNodes.push(mutation.addedNodes[i]);
        })
    });
    observer.observe(document.documentElement, { childList: true
});
    console.log(insertedNodes);
```

ИБ

- CORS
- CSP
- Авторизация
  - JWT
  - OAuth
  - Basic
  - Cookie based
  - SSO
- HTTP заголовки
  - strict-transport-security
  - x-frame-options
  - HttpOnly cookies
  - content-security-policy
  - x-xss-protection
  - HTTP Strict Transport Security (HSTS)
  - HTTP Public Key Pinning (HPKP)
- Атаки
  - OWASP Top 10
  - SQL Injection
  - XSS
  - CSRF
  - DoS
  - Password Attack
  - Man-in-the-Middle

    - Eavesdropping Attack
    - Insider Threats
    - Drive-By Download Attack
    - theft of credentials or an escalation of permissions
    - CSRF
    - Phishing
    - Malicious redirects
- Разрешение инцидентов
    - monitoring
    - logging
    - firewalls
    - headers
    - packages i.e. "helmet" package
- JWT
    - Где лучше хранить JWT в cookie или localstorage. Почему?

Проектирование/шаблоны

- шаблоны проектирования
    - должны описывать решения, а не определять их
    - pattern не шаблон а закономерность http://sergeyteplyakov.blogspot.com/2015/04/difference-between-templates-and.html
- GRASP паттерны проектирования - 9шт
- GoF - gang of four - 23шт
    - порождающие - новый экземпляр
        - прототип - клонирование существующего объекта
        - одиночка(синглтон) - предотвращение создания копий класса
        - простая фабрика - создание с параметрами. функция или метод, возвращающая объекты разных прототипов
        - фабричный метод - создание с абстрактным методом/логикой. Объекты создаются посредством вызова не конструктора, а генерирующего метода, реализованного/переопределённого дочерними классами
        - строитель - многоэтапное создание/модификация/достраивание нового объекта, облегчение конструктора класса `one.two.three`
        - абстрактная фабрика - фабрика фабрик. связывание фабрик с разной логикой через интерфейс+абстрактный класс. позволяет создавать семейства объектов или взаимосвязанные объекты
    - структурные - модификация, способ взаимодействия, доступные извне методы
        - адаптер - конвейер для трансформации, один к одному. Связывание через общие методы в интерфейсе
        - декоратор - костыль. подключать к объекту дополнительное поведение (статически или динамически), не влияя на поведение других объектов того же класса. Шаблон часто используется для соблюдения принципа единственной обязанности (Single Responsibility Principle), поскольку позволяет разделить функциональность между классами для решения конкретных задач.
        - фасад - предоставляет упрощённый интерфейс для сложной подсистемы

        - мост - компоновки вместо наследованию, отделение абстракции от реализации. Логика передаётся между классами
        - компоновщик - ??? описывает общий порядок обработки группы объектов, словно это одиночный экземпляр объекта. Суть шаблона — компонование объектов в древовидную структуру для представления иерархии от частного к целому. Шаблон позволяет клиентам одинаково обращаться к отдельным объектам и к группам объектов.
        - приспособленец - совместное использование объектов, экономия ресурсов
        - заместитель - это класс, функционирующий как интерфейс к чему-либо. «Заместитель» может просто переадресовывать запросы настоящему объекту, а может предоставлять дополнительную логику: кеширование данных при интенсивном выполнении операций или потреблении ресурсов настоящим объектом; проверка предварительных условий (preconditions) до вызова выполнения операций настоящим объектом.
    - поведенческие - Они связаны с присвоением обязанностей (responsibilities) объектам. В отличие от структурных, добавляют логику обработки потоков данных.
        - Цепочка ответственности -позволяет создавать цепочки объектов. Запрос входит с одного конца цепочки и движется от объекта к объекту, пока не будет найден подходящий обработчик
        - Итератор - для перемещения по контейнеру и обеспечения доступа к элементам контейнера. Шаблон подразумевает отделение алгоритмов от контейнера.
        - Команда - словарь action. объект используется для инкапсуляции всей информации, необходимой для выполнения действия либо для его инициирования позднее. Информация включает в себя имя метода; объект, владеющий методом; значения параметров метода.
        - Посредник - подразумевает добавление стороннего объекта («посредника») для управления взаимодействием между двумя объектами («коллегами»). Шаблон помогает уменьшить связанность (coupling) классов
        - Хранитель - фиксирует и хранит текущее состояние объекта, чтобы оно легко восстанавливалось
        - Наблюдатель - В шаблоне «Наблюдатель» есть объект («субъект»), ведущий список своих «подчинённых» («наблюдателей») и автоматически уведомляющий их о любом изменении своего состояния, обычно с помощью вызова одного из их методов.
        - Посетитель - это способ отделения алгоритма от структуры объекта, в которой он оперирует. Результат отделения — возможность добавлять новые операции в существующие структуры объектов без их модифицирования. Это один из способов соблюдения принципа открытости/закрытости (open/closed principle).
        - Стратегия - позволяет переключаться между алгоритмами или стратегиями в зависимости от ситуации
        - Состояние - реализует машину состояний объектно ориентированным способом. Это достигается с помощью:
            - реализации каждого состояния в виде производного класса интерфейса шаблона «Состояние»,
            - реализации переходов состояний (state transitions) посредством вызова методов, определённых вышестоящим классом (superclass).

        - Шаблон «Состояние» — это в некотором плане шаблон «Стратегия», при котором возможно переключение текущей стратегии с помощью вызова методов, определённых в интерфейсе шаблона.
    - Шаблонный метод - определяет каркас выполнения определённого алгоритма, но реализацию самих этапов делегирует дочерним классам.

SQL

- http://sqlfiddle.com/
-

бесконечная прокрутка

- intersection observer
- scroll+debounce