

Микрофронты 101

Мой доклад появился в ответ на распространённый на собеседах заявление "Аааа... микрофронты? да, знаю, module federation". Это достаточно узкое представление о том, что такое микрофронт и я постараюсь сделать онбординг в эту концепцию и в то, как она используется в Леруа Мерлен Россия.

О себе

Всем привет, меня зовут Валентин Федяков. Я технический архитектор домена omnisaies, а до этого был лидером фронтенд разработки. В компании чуть больше 5 лет и видел некоторое...

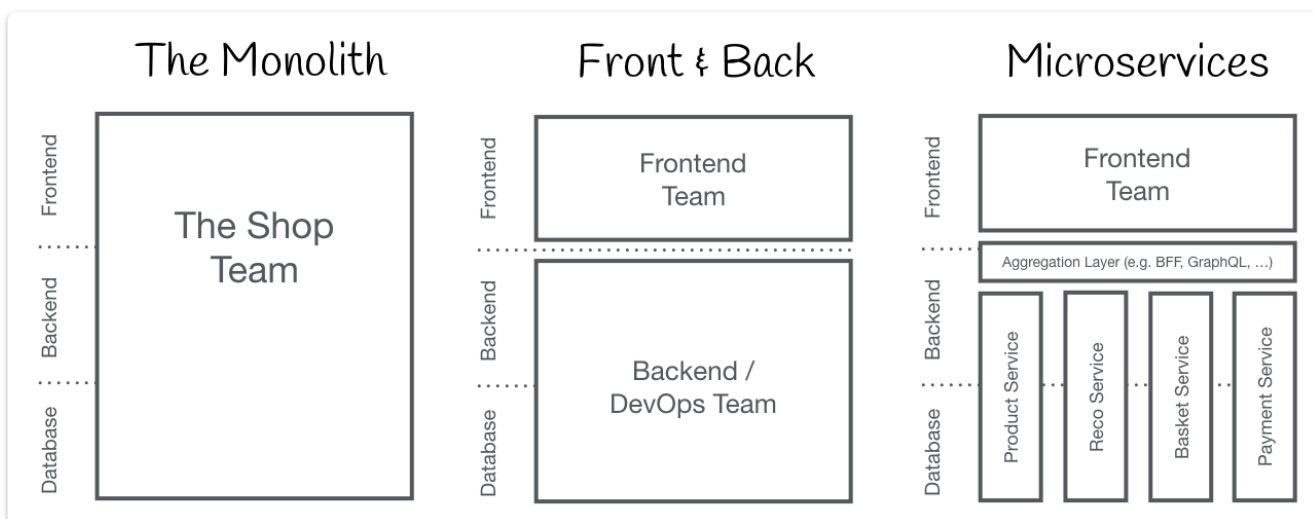
Участвовал в миграции с битрикс на АЕМ, с АЕМ на микрофронты.

Года 4 назад внедрил подход разработки фронта на вебкомпонентах, которые по сей день можно встретить в продакшен.

Помог 6 стажерам войти в профессию и провел более 150 технических собеседов фронтенд разработчиков в ЛМ (только за предыдущую неделю их было 9).

А еще я очень ленивый, потому некоторые картинки я буду брать прямо из референсов, а слайды собраны на коленке, чтобы меня не было скучно слушать. Сами референсы будут в конце.

Немного истории



Вероятно все из вас видели такую картину. По сути, это эволюция

приложений с целью достижения некоторой гибкости. На первых этапах все сильно зависело от Закона Конвея "организации проектируют системы, которые копируют структуру коммуникаций в этой организации". Компаниям было удобно вначале держать разработку в одном месте т.к. это понятно в рамках структуры. Просто есть "отдел разработки". Это породило монолитную архитектуру.

Проблемы монолита:

- высокая когнитивная сложность проекта мешала пониманию, как все работает
- изменения одних разработчиков ломали изменения других
- сильная связанность заставляла долго и тщательно тестировать релиз и отодвигать деплой на недели, месяца
- доступно только вертикальное масштабирование (через увеличение производительности компьютеров, где запускался монолит)
- слабая отказоустойчивость
- сложно проводить быстрое тестирование бизнес-идей

Необходимость стать гибче (независимые релизы, сокращение коммуникаций), вынудило перейти к "функциональным командам" (фронт, бек, данные, инфраструктура...), т.к. это так же отвечало на вопрос понятности структуры и решало некоторые проблемы монолита.

Изначально, фронт нес меньшую часть ответственности за общий продукт, потому он долгое время оставался не делим, не управляем. Простая статика. Менялись лишь подходы к его реализации.

На беке вопрос с разделением был решен через микросервисы, каждый из которых содержал некую связную функциональность (функциональную или предметную), которые можно было независимо деплоить, масштабировать.

На фронте же все было печально. Например, во времена http1.1 было круто создавать такие страницы и сайты, которые не делают никаких дополнительных запросов. Все в html, js, css, даже jpg. В итоге получили огромный HTML, который проседал по перформанс и не кэшировался у пользователя. Это связано с ограничениями http1.1 и его ограничению на одновременный запрос только 7 ресурсов из браузера.

Изменения наступили лишь в момент появления браузера chrome, с его инновационным подходом онлайн обновлений, высокопроизводительным

движком Chromium и эволюции EcmaScript до 5 версии. Так появились Single page application, как доминирующая парадигма. По сути, монолитный интерфейс. Отличный перформанс в лайтхаус (на тот момент), крошечный html и огромный, монолитный кусок js, который сложно разделить.

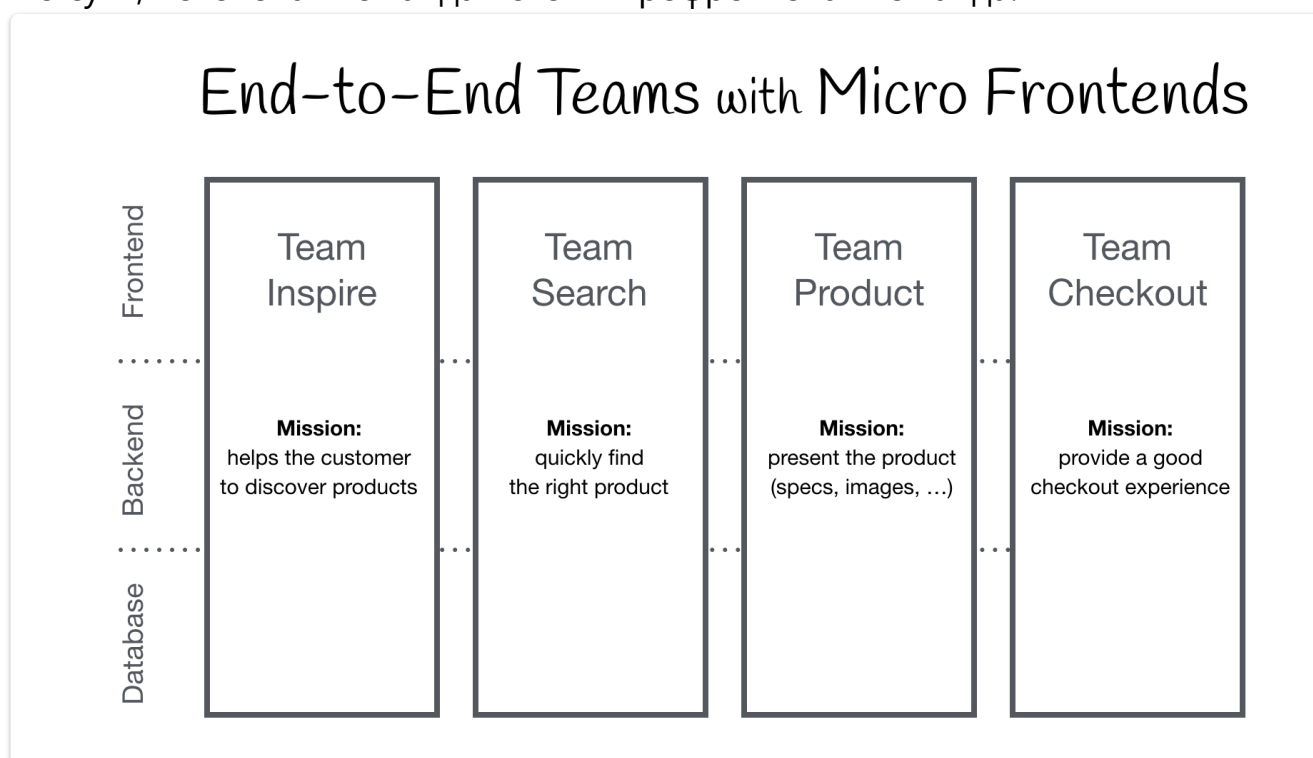
Это создавало проблемы для развития продуктов т.к. опять уперлись в монолит, один единственный интерфейс, поддерживаемый множеством команд с большим количеством взаимодействий. Были разные подходы к тому, как разделять фронт. Но все не то и всегда упиралось в межкомандную коммуникацию.

Решение пришло от куда не ждали, не технологии, а организация команд. Мэтью Скелтон и Мэтью Элиас в книге "Team Topologies: Organizing Business and Technology Teams for Fast Flow" описали идею о том, как делать команды более адаптивными и гибкими в рамках всего продукта.

Потоковая команда - это коллектив, ориентированный на единый, значимый поток работы. Команда уполномочена создавать и предоставлять потребительские или пользовательские ценности как можно быстрее, безопаснее и независимей, не требуя передачи другим командам части работы для выполнения.

Skelton M., Pais M. Team Topologies. — IT Revolution, 2019

По сути, потоковая команда - это микрофронтонная команда.



Концепция микрофронтов

Концепция микрофронтонтов разделена на 2 элемента:

- организация команды
- техническая реализация

Организация команды

На мой взгляд, про организационный подход микрофронтонтовых команд, говорить немного не мне. Я чуть больше про технику, но хотел бы, что бы вы не забывали, что микрофронтонты на прямую связаны с организацией команд внутри компании (при этом я помню, что технологическая эволюция стала причиной почему микросервисный подход стал возможен).

Ну и одна микрофронтонтовая команда реализует множество микрофронтонтов, но один микрофронтон принадлежит только одной команде.

Техническая реализация

На первый взгляд, технически идея микрофронтонтов выглядит так. И на второй тоже...



Заранее скажу, что у микрофронтонтов есть альтернатива, например Автономные системы ([SCS: Self-contained Systems \(scs-architecture.org\)](https://scs-architecture.org)). Разница с микрофронтами в основном в том, что SCS предписывает иметь автономное веб-приложение, что в большинстве случаев избыточно.

Прежде всего я бы хотел выделить вопрос "что именно должно быть микрофронтонтом и почему?". Любые решения несут за собой последствия, мы не можем сделать ВСЕ микрофронтонтом т.к. это подразумевает некую абстракцию на уровне api и интеграций с другими микрофронтами. Ведь микрофронтон становится микрофронтонтом только в рамках некоего макрофронта (общего продукта).

Границы

Как и в микросервисах, есть 2 архитектурных типа выделения границ:

- предметная (корзина, личный кабинет, продуктовая страница и т.д.)
- техническая (авторизация/аутентификация, веб-аналитика, роутинг, ui-kit и т.д.)

Гранулярность

Далее идет вопрос про гранулярность. Есть два способа декомпозиции микрофронтонтов:


- на основе страниц
- на основе виджетов

Декомпозиция на основе страниц


Простота декомпозиции на основе страниц с точки зрения технической реализации, очень привлекательная. Пользователь нажимает на ссылку и запрашивается новая страница с нужного микрофронта.

Декомпозиция на основе виджетов

The Model Store






Tractor Porsche-Diesel Master 419



buy for 66,00 €

basket: 0 item(s)

Related Products

Team Product

Team Checkout

Team Inspire

Этот подхода в декомпозиции, чуть сложнее, за счет того, что должен быть некий механизм оркестрации, который будет управлять, что именно должно подключиться на странице.

Интеграция

Выделяют 2 подхода `buildtime` и `runtime`.

Buildtime

Интеграция микрофронтонтов, их связь, происходит в момент сборки проекта. Как пример - `npm` пакеты. На мой взгляд, это наиболее безопасный подход т.к. еще на этапе сборки у нас есть юнит тесты, типизация, статический анализ. Команда контролирует версиюность и не столкнется с неожиданным поведением другого микрофронтонта.

Runtime

Интеграция микрофронтонтов происходит в браузере или в момент запуска `JS` на беке. Это наиболее удобный способ для независимой поставки инкремента, команда, с мелкой гранулярностью не будет ждать релиза от команды, которая интегрирует этот микрофронтон у себя. Но тут появляется проблема во

временной связанности, SLO команды поставщика, а также необходимость в доверии, что другая команда тебе ничего не сломает в глобальной области.

Глобальная область

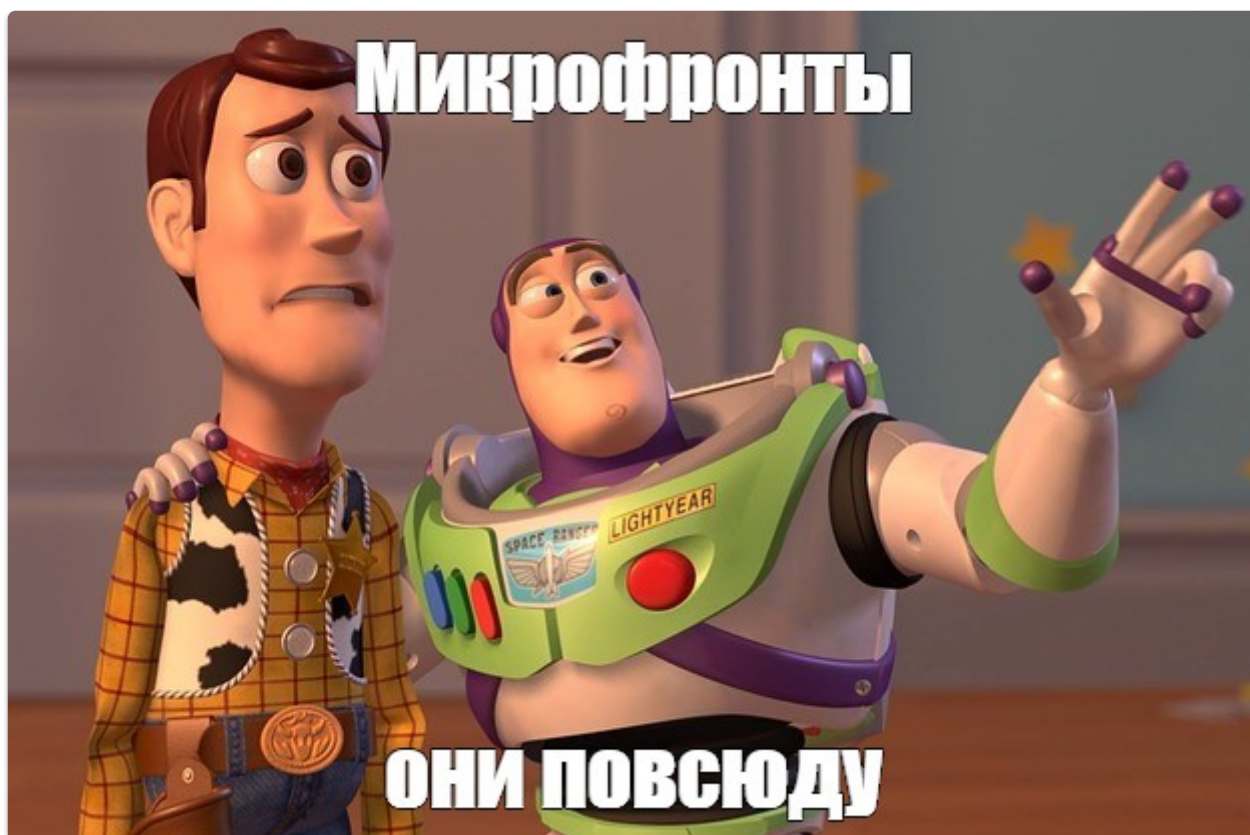
На мой взгляд, современный WEB не предназначен для микрофронтонтов из-за разделения ресурсов в браузере. Даже в микросервисах, не всегда удастся разделить ресурсы так, чтобы это не аффектило на другие микросервисы. При декомпозиции микрофронтонтов на основе страниц эта проблема заметна в меньшей степени (там общие лишь в рамках Storage API). А вот при декомпозиции на основе виджетов (за исключением некоторых случаев)... я бы сказал так:

"Микрофронты это не про технологии, а про то, как жить в коммунальной квартире и не поубивать друг друга из-за не смытого в унитазе"

Коммуникации, следование правилам, уважение к труду коллег и их нефункциональных требований - это основа успешного микрофронтонтового подхода в компании. Из личного опыта скажу, что опускаются руки, когда ты занимаешься перформанс, потому что от тебя требуют хорошие цифры для SEO, сокращаешь кодовую базу на 200кб и параллельно, на твою страницу прилетает кода еще на 500кб в виджете другой команды и он "очень нужный и мы его не выпилим!".

Так же вопрос, а что делать, если микрофронтонт другой команды перестал отвечать? Как это повлияет на путь твоего пользователя? Оставляю эти вопросы открытыми.

Реализации



Предлагаю пробежаться по возможным реализациям микрофронтов, их плюсах, минусах. Это лишь базовые реализации и в большинстве случаев они используются не в чистом виде, а их сочетание или модификации под определенные условия.

Server-side include

Скажем это микрофронты до микрофронтов. Динамическая сборка страницы на основании директив сервера. Например, могут быть вызваны другие сервисы, которые вернут статическую разметку.

```
<!--#include virtual="/footer.html" -->
```

language-html

У этого подхода есть огромное преимущество в том, что тебе не важно, какая технология используется на удалённом сервере, главное, что ты получаешь разметку. Так же, результаты ответа хорошо кешируются. Очень простая реализация. А сама технология переболела всеми детскими болячками. Но современные фронтенд разработчики, на мой взгляд, боятся всего, что выходит за их стек Typescript и тем более осуществлять взаимодействие через пользовательские события. Так же могут быть сложности с дедупликацией кода js, наложением селекторов css.

Примером реализации может послужить Kobi наших французских коллег. У этого подхода есть еще некоторые вариации, в том числе и Client-side

include, но, если честно, я давно уже не видел такого.

iframe

Еще один динозавр. Не стареющая классика. Через разметку html ты показываешь на своей странице страницу из другого микрофронта.

```
<iframe                                     language-html
  id="inlineFrameExample"
  title="Inline Frame Example"
  width="300"
  height="200"
  src="https://www.openstreetmap.org/export/embed.html?
bbox=-0.004017949104309083%2C51.47612752641776%2C0.00030577182769775396
%2C51.478569861898606&layer=mapnik"
>
</iframe>
```

Самое большое преимущество в том, что не нужно никаких дополнительных соглашений, а ресурсы почти полностью изолированы т.е. сторонние правки не мешают работоспособности твоего микрофронта. Так же полная независимость от используемых технологий. Но пользователь расплачивается за это производительностью т.к. для каждого iframe "создается своя страница", которой выделяются ресурсы. Из опыта, 10 таких iframe на странице, сильно просаживали общую производительность компьютера (бек офис) до такой степени, что приходилось закрывать браузер. Так же, общение между такими микрофронтами доступно только через postMessage, с его сериализация/десериализация данных.

Сторонний скрипт подключенный к странице

Например - Google tag manager или Google Optimize. Отдельная команда, независимая поставка, свои беки для работы с данными. Звучит как микрофронт, организован как микрофронт - значит микрофронт.

Но на мой взгляд, за частую, это ящик Пандоры. Вне зависимости от того, как реализован твой микрофронт, эти скрипты начинают на прямую управлять DOM, что часто приводит к ошибкам.

npm packages

Самый распространенный, контролируемый, тестируемый способ внедрения другого функционала в свой микрофронт. Изменение в другом пакете не

станут для тебя неожиданностью, если ты фиксируешь версию пакета у себя в зависимостях. При этом код стороннего микрофронта максимально оптимизируется в твоём бандле и в процессе разработки. А при необходимости можно использовать ленивую подгрузку. Проблема у этого подхода лишь одна: отсутствие независимой поставки инкремента в продакшен. Любое изменение стороннего микрофронта поставляется через новый релиз твоего.

Module federation

Можно представить, как альтернативу npm пакетам. Ты получаешь независимое развертывание, отсутствие версионности. Но получаешь отсутствие оптимизации бандла (дедупликация, тришейкинг) и постоянную асинхронность виджетов. Так же, довольно долгое время была полноценная реализация только в webpack 5. Но на сегодня существуют плагины, которые и для других сборщиков, правда они еще не все свои болячки получили.

WebComponents

Вебкомпоненты это описание 3 стандартов, применяемых к одному виджету

- Custom Elements
- ShadowDOM
- html template

Это стандарт веб разработки, который не пользуется популярностью из-за ограничений со стороны ShadowDOM и отсутствия удобного фреймворка. Он позволяет изолировать стили и dom, но JS выполняется в основном потоке с доступом к глобальной области видимости.

Single-spa

Признаюсь честно, не использовал, только слышал и немного смотрел документацию. Как я понял это фреймворк и под капотом мало чем отличается от module federation. Потому ничего не буду говорить.

Микрофронты в Леруа Мерлен

Вообще, в ЛМ я встречал 5 подходов с использованием микрофронтов. Предлагаю пробежаться по 4 из них и задержаться на последнем, наиболее распространенном в компании. Архитектура, подходы рассказанные здесь будут применимы ко многим реализациям микрофронтов.

iframe

Да, используем. В основном это связано со скоростью разработки, например в WeCare. Нет необходимости ждать от смежных команд, когда они смогут реализовать микрофронт в своем продукте. Это не целевое решение, но прекрасно работает с некоторыми ограничениями.

Kobi

Это яркий пример Server-side include. Фреймворк из ADEO, в котором используется разметка микрофронт - шаблон с разметкой, а другие микрофронты, через UI добавляются в этот шаблон. Это позволило сделать шаблонизацию, потоковую передачу, управляемую публикацию компонентов. Все очень быстро.

1. <https://medium.com/adeo-tech/behind-leroymerlin-fr-micro-frontends-47fd7c53f99d>
2. <https://adeo.github.io/kobi--io/>

WebComponents

Когда была необходимость, в рамках АЕМ сделать независимую поставку, мной было предложено и внедрено использование данного подхода. Был подготовлен ui-kit, а на его основе реализован набор компонентов с использованием lit-html. А получившийся результат и опыт описал в двух статьях на хабре.

1. [Знакомство с lit-element и веб-компонентами на его основе / Хабр \(habr.com\)](#)
2. [Опыт интеграции веб-компонентов на сайт Леруа Мерлен / Хабр \(habr.com\)](#)

Хочется сказать, что у этого были большие проблемы, например поддержка IE11, которая осуществлялась в те годы. Так же сами разработчики не сильно горели желанием на прямую модифицировать DOM, общаться событиями и прокидывать только строковые атрибуты. Но хотел бы отметить, что вебкомпоненты очень помогли в переходе с АЕМ на текущий подход микрофронтов для внешнего пользователя, а также по сей день являются частью UI и прекрасно справляются со своей работой.

Так же ADEO заинтересовалось в использовании вебкомпонентов. Они их генерировали из [svelte](#). Как одна из реализаций - [Mozaic \(adeo.cloud\)](#).

Считаю, что это самая недооцененная технология. Но рынок порешал.

Module federation

На нем реализован, например, magportal. Так же MF выбран, как основной способ реализации микрофронтности в бекофис продуктах.

Проблема MF, вне зависимости от того, какая выбрана гранулярность - единая точка отказа в виде контейнера-обертки над микрофронтами. Как только основное приложение перестает отвечать, так все остальные перестают работать.

Еще одна проблема это "как прокинуть конфигурацию микрофронта в другой, не хардкодя в исходники". Но это проблема любого решения runtime + client-side.

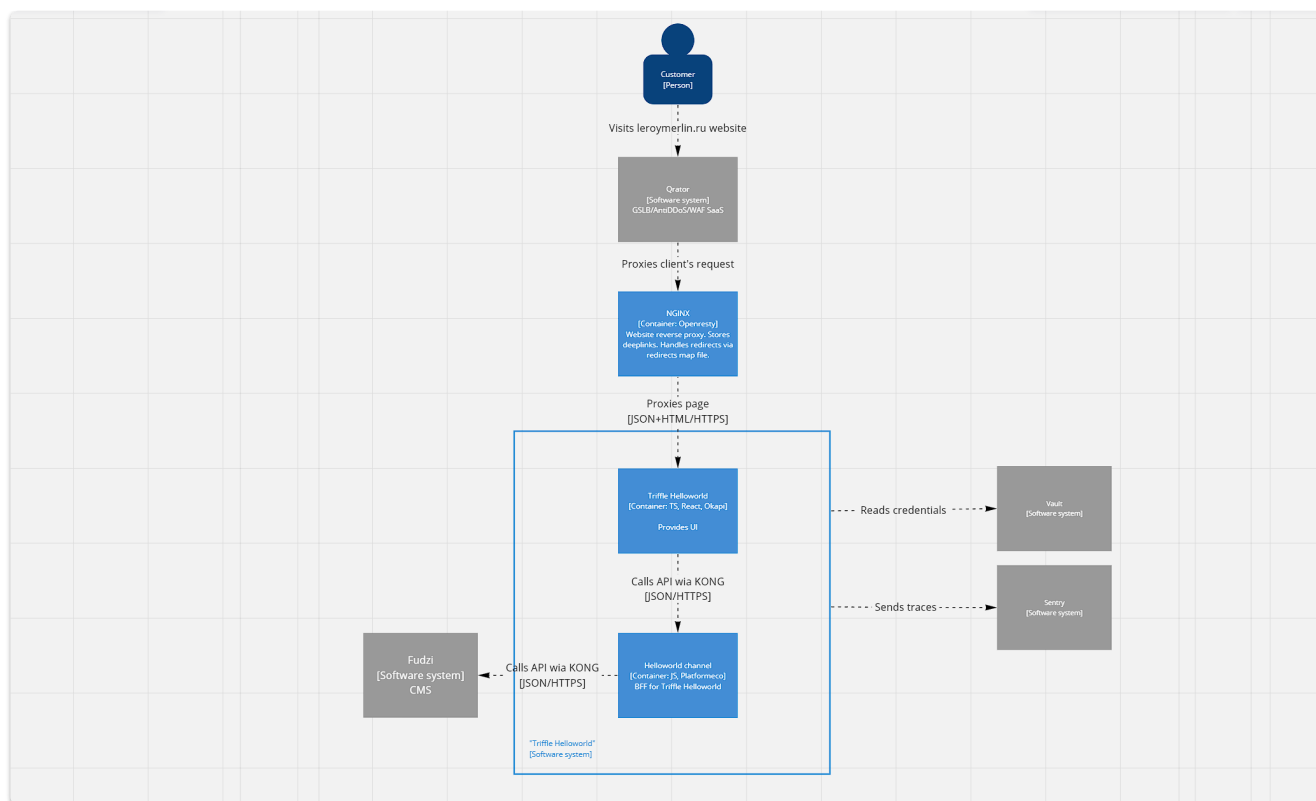
Платформа микрофронтон

Тут я бы хотел подробнее остановиться.

Немного контекста. Пример я приведу на основе сайта leroym Merlin.ru

Сайт, по сути, это набор микрофронтон с постраничной гранулярностью с некоторыми вкраплениями виджетов.

Типовая архитектура



[Example Helloworld, Visual Workspace for Innovation \(miro.com\)](#)

Архитектурный квант, как описано в книге "Building Evolutionary Architectures", представляет собой независимо развёртываемый компонент с высокой функциональной связностью, который включает в себя все структурные элементы, необходимые для правильной работы системы.

Микрофронты представляют архитектурный квант и состоят из 3 частей:

- техническая реализация гранулярности (в данном случае okapi и доменное имя)
- мидлвара между бекком и фронтом (graphql или rest через bff)
- инфраструктура и процессы (в данном случае платформа triffle)

Okapi

Okapi это сервер для запуска react приложений. Он реализует:

- server side rendering (страничная гранулярность)
- remote с server side renderin (виджетная гранулярность)

На начальных этапах закладывался еще и межмикрофронтонный роутинг, но в процессе эволюции фреймворка он оказался ненужным т.к. угрожал стабильности и производительности.

Раньше, в архитектуре присутствовал еще и service discovery, чтобы меньше конфигурировать сервер и оперативно изменять место нахождения ремоутов.

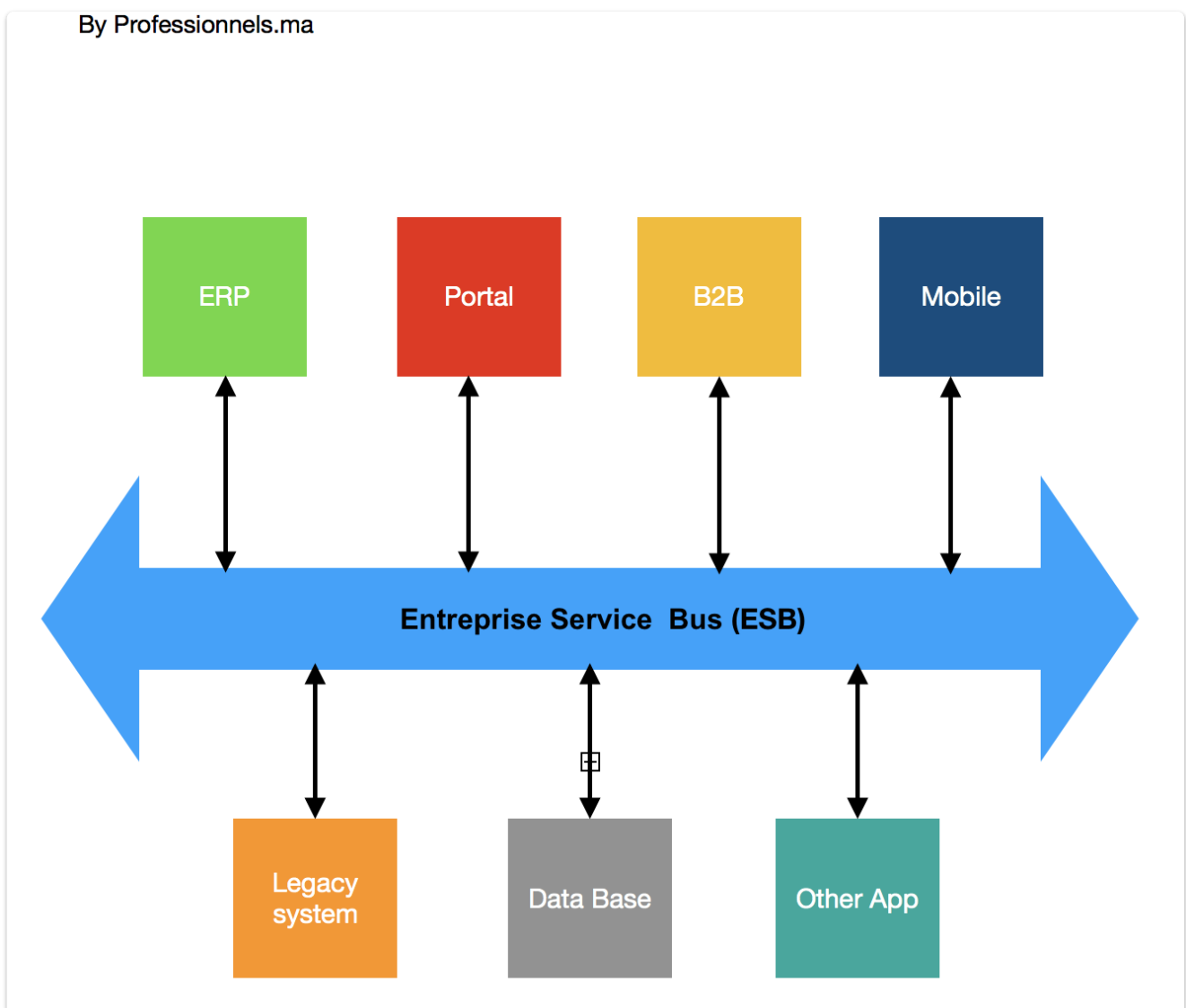
Но в процессе роста системы, SD оказался единой точкой отказа и в целом не нужной системой т.к. используется платформа triffle с k8s его неймспейсами и ингрессами.

Мидлвара

Вообще, рассматривают 3 подхода к ее реализации:

- корпоративная сервисная шина (ESB)
- graphql
- backend for frontend (BFF)

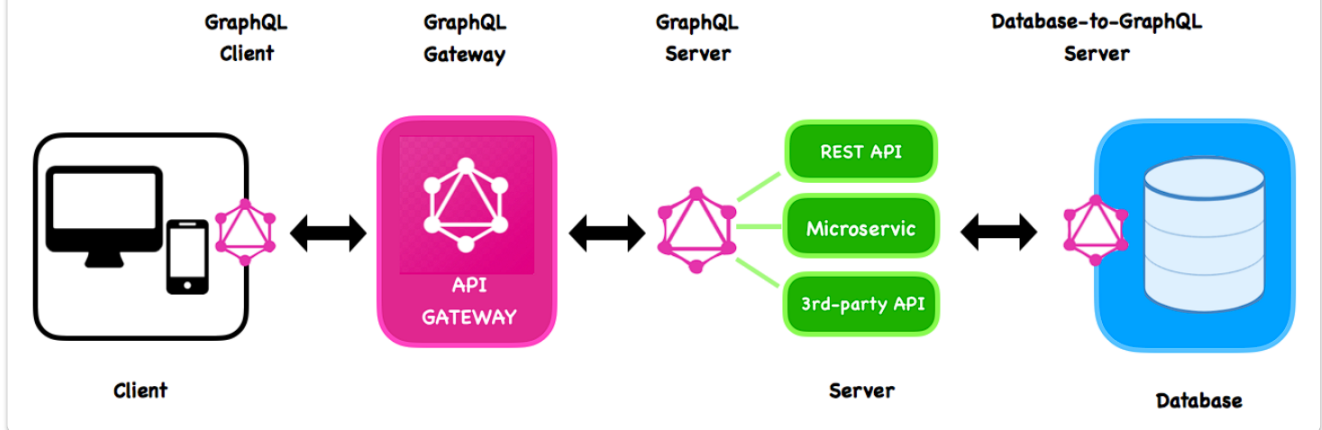
Корпоративная сервисная шина



Самый негативный сценарий. Он мешает команде быть потоковой т.к. команда ей не управляет, является единой точкой отказа с низкой производительностью.

GraphQL

GraphQL Ecosystem Breakdown



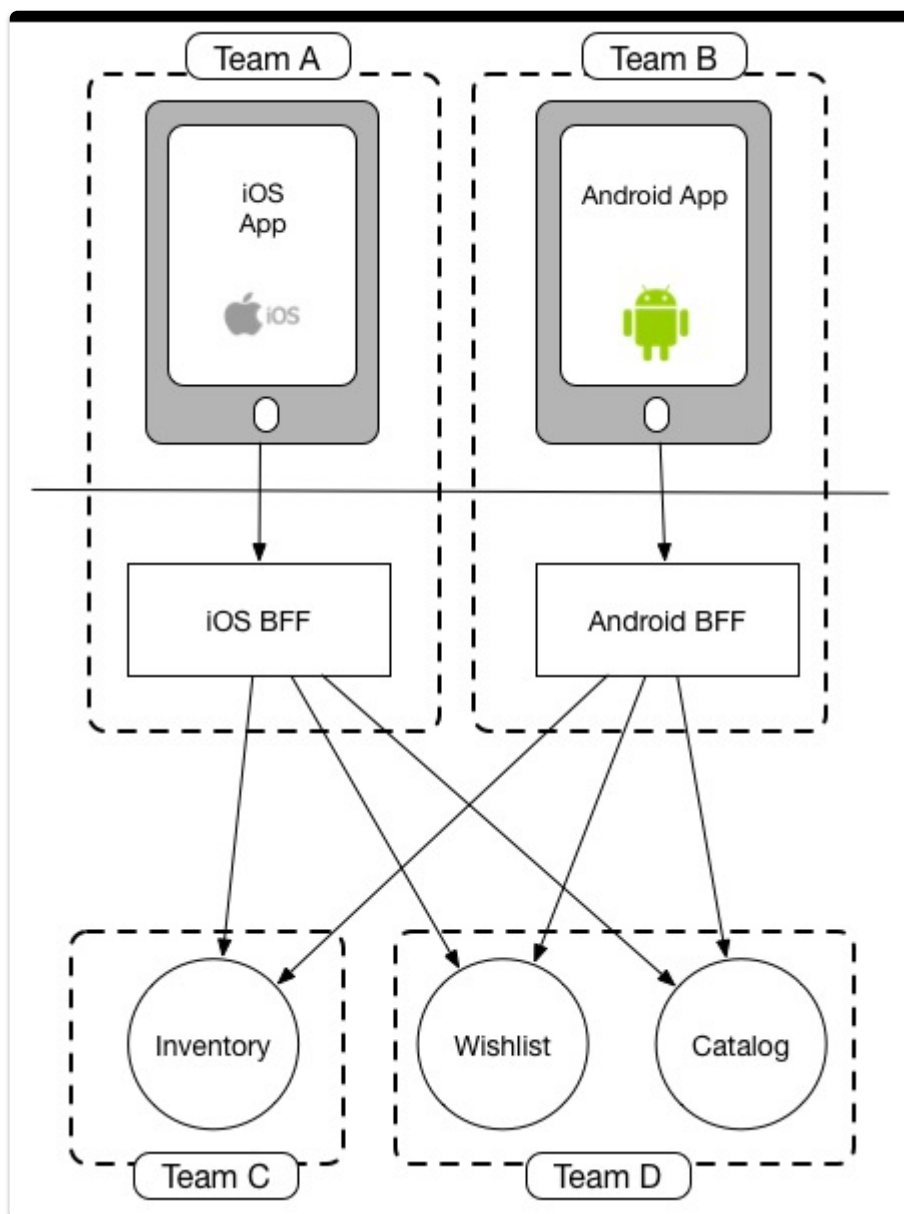
Он создан для агрегации и фильтрации данных по запросу. На мой взгляд, его абсолютное преимущество перед REST - это наличие явной схемы данных.

Но у graphql есть 3, на мой взгляд, недостатка:

- Это не "бесплатная" технология. Для того что бы ее использовать, необходимо иметь клиенты и на фронте, и на бекенде для работы с запросами.
- Некоторая сложность с изменениями данных через graphql. Я видел реализации, когда данные получали через graphql, а изменяли уже по REST.
- Некоторая сложность с кешированием данных.

Мне сложно сказать, почему он у нас не прижился в стеке в границах продукта.

Backend for frontend



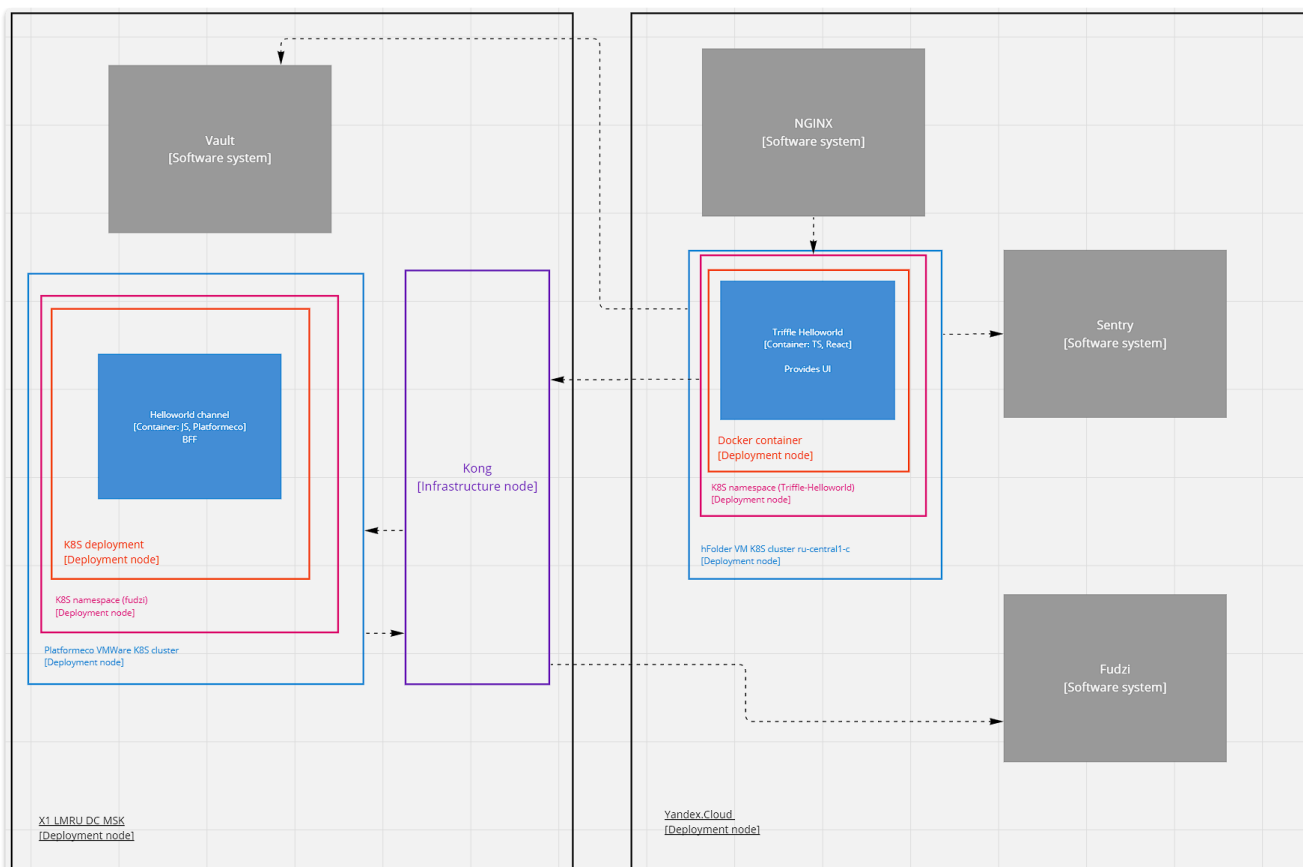
Этот паттерн, который полностью замещает graphql т.к. тоже создан, для агрегации и фильтрации данных, но построен на REST. В нашей архитектуре эта почетная роль выпала на Platformeco.

Сразу скажу, что "один микрофронт - один BFF". Это позволяет снизить связанность между продуктами, обеспечить независимое развертывание. Не бойтесь дублирования кода, это допустимо в рамках достижения слабой связанности. Плохо будет, если ваш продукт перестанет работать, потому что другая команда обновила версию какой нить зависимости. Т.е. это гарантия некоторой стабильности продукта.

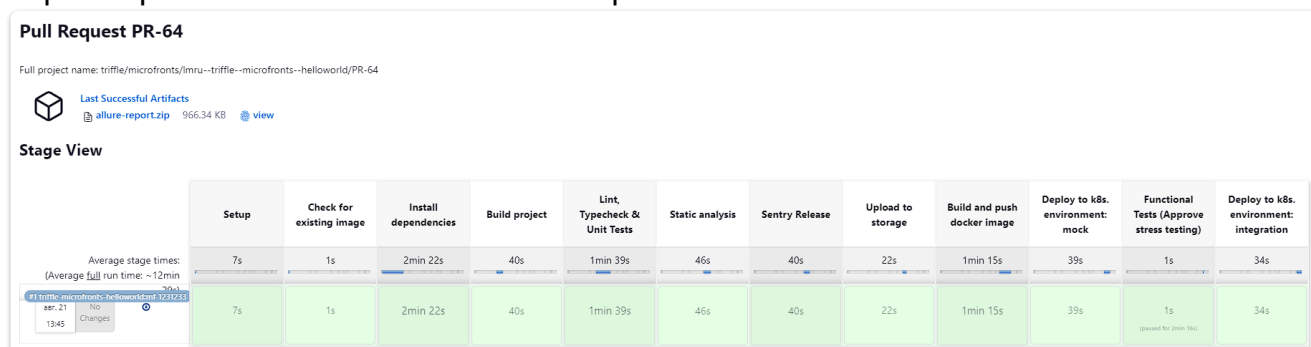
Инфраструктура и процессы

В нашем случае это платформа Triffle.

Саму платформу я разделяю на пайплайн и инфраструктуру в k8s. Вот типичная схема развертывания.



А пайплайны обеспечивают надежную поставку инкремента с гарантированным качеством и быстро.



В пайплайнах тоже бывают затыки, но где их нет. Ткущий пайплайн предлагаю рассмотреть на этапе вопросов-ответов т.к. тут много что есть.

Результат

По сути, сайт, из-за страничной гранулярности, стал набором шардов с репликацией каждого шарда. Независимая поставка, высокая отказоустойчивость, доступность и масштабируемость. Слабая связанность между границами микрофронтонтов и высокая связанность внутри продукта дают стабильность, а также возможность увеличивать количество микрофронтонтов в рамках общего продукта.

Проблемы

Реализация только через SSR. Ремоут компоненты, полноценно, работают только при SSR. Для использования ремоутов допустимо использование только окарі.

А еще у меня нет ответа на то, что такое "ошибка 500" в рамках микрофронта, т.к. в большинстве случаев сами микрофронт доступен. Не доступны некоторые данные, без которых он не может быть полноценно отрисован и полноценно работать. Но т.к. микрофронт содержит ремоуты, а они могут быть отрисованы (хедер и футер) то мне бы хотелось максимально сильно сократить количество "500 ошибок" через более разумный подход к пользовательскому опыту.

Заключение

- Микрофронты - это про организационную структуру, договоренности и в меньшей степени про технологии, которые их реализуют.
- Нет единого, правильного решения по микрофронтному стеку, везде есть компромиссы.
- Нужно четко понимать, что именно будет "микрофронтом". Нельзя сделать сразу все или быстро перейти на этот стек. Так же достаточно дорого будет выделить из текущей реализации некий шаренный элемент т.к. он несет за собой множество зависимостей.
- Если есть возможность не делать микрофронты, то лучше их не делать.
- Если есть необходимость распилить монолит, то я бы рекомендовал использовать архитектурный паттерн Душитель и грануляцию на основе страниц.

Референсы

1. [Micro Frontends \(martinfowler.com\)](https://martinfowler.com/microfrontends.html)
2. [Micro Frontends - extending the microservice idea to frontend development \(micro-frontends.org\)](https://micro-frontends.org/)
3. <https://www.manning.com/books/micro-frontends-in-action>
4. [Micro Frontend Architecture | Nx](#)
5. [Welcome to Micro frontend dev!](#)
6. [Микрофронтенды: зачем нужны и как к ним прийти \(tproger.ru\)](#)
7. [Создание микросервисов. 2-е издание \(piter.com\)](#)
8. [Что такое микрофронтенды? Объясняем за 5 минут | Вокруг IT | Дзен \(dzen.ru\)](#)

9. https://www.youtube.com/watch?v=yWzKJPw_VzM
10. [Google Chrome — Википедия \(wikipedia.org\)](#)
11. [Team Topologies: Рациональный Подход к Организации Команд / Хабр \(habr.com\)](#)
12. [Потоковые команды — Aleksei Solonkov на vc.ru](#)
13. [Разделяй и не страдай: что выбрать для микрофронтенда? \(vk.com\)](#)
14. [Apache httpd Tutorial: Introduction to Server Side Includes - Apache HTTP Server Version 2.4](#)
15. [CSI, SSI, ESI в композиции микрофронтендов - IT dev journal \(it-dev-journal.ru\)](#)
16. [iframe: The Inline Frame element - HTML: HyperText Markup Language | MDN \(mozilla.org\)](#)
17. [Window.postMessage\(\) - Интерфейсы веб API | MDN \(mozilla.org\)](#)
18. [Tutorial - A Guide to Module Federation for Enterprise - DEV Community](#)
19. [Sam Newman - Backends For Frontends](#)