Микрофронтенды — достаточно хайповая тема во фронтенд-разработке. Если коротко, то это деление монолитного приложения на семантически изолированные части и их независимая разработка. Каждая страница сайта или приложения структурно состоит из блоков, это может быть блок рекламы, блок обратной связи и так далее. Они взаимодействуют между собой по задуманной разработчиком логике, но при этом их можно разрабатывать, тестировать и деплоить как отдельные приложения.

Сама идея не нова и пришла из мира бэкенда — речь о микросервисной архитектуре. Суть та же: у нас есть большое приложение, которое нужно продолжать поддерживать и разрабатывать. Над этим трудятся несколько команд, каждая из которых занимается своей частью. В логике монолитного бэкенда разрабатывать свою часть без риска затронуть функциональность другой части приложения непросто. Поэтому бэкенд разделяет приложение на микросервисы и разрабатывает их изолированно, но при этом так, чтобы они продолжали взаимодействовать между собой.

Микрофронтенды, как и микросервисная архитектура, — результат естественного эволюционного процесса. Старый подход в виде монолита был удобен во многих смыслах, но с учетом усложнения технологий и роста приложений приобрел свои минусы. Монолит плохо масштабируется, он труден в поддержке, разработке и тестировании.

Классический способ борьбы со сложностями — это разделение на составляющие. Если вести разработку всего приложения целиком сложно, нужно «распилить» его на части и разрабатывать отдельно.

Где подвох

Логика микросервисов наложилась на фронтенд-разработку. Программистам, работающим на «фронт», тоже хотелось приобрести гибкость и независимость команд, ту масштабируемость, которая характерна для микросервисной архитектуры.

Одно но: фронтенд всегда работает с единой средой, которая трудно отделима. Он выполняется на стороне браузера (клиента), поэтому в рамках одного приложения всегда есть только одна адресная строка, один глобальный объект localStorage, один DOM. Как бы мы ни делили приложение, некая общая часть всегда остается. Именно вокруг этой проблемы строятся основные ограничения микрофронтендов.

И главное разочарование: в рамках микрофронтенда не получится организовать такую же изолированность частей приложения, как при микросервисной разработке в бэкенде. Во всяком случае пока.

Так как полной изолированности не достичь, программистам фронтенда все равно приходится очень взвешенно принимать любые изменения в коде. Поэтому при разработке приложения в логике микрофронтендов очень важна экспертиза разработчиков. Они должны понимать, как потенциально может повлиять на приложение в комплексе то или иное изменение в коде.

Если вы хотите перейти на микрофронтенд

В первую очередь, поймите, зачем это вам. Микрофронтенд нужен не всем. Да, это тренд, который сейчас активно обсуждается в комьюнити. Да, многие большие компании так или иначе придут к этой архитектуре. Но все это актуально в случае объемного приложения. Если сайт состоит из нескольких страниц, микрофронтенд теряет смысл. Это уже оверхэд. Смысл в делении монолита есть, если у вас крупный проект, несколько команд разработки, которые, возможно, используют разные фреймворки. Это, к слову, еще один плюс микрофронтенда: разные команды могут писать код на разных стеках. Кто-то пишет на React, кто-то — на Angular, но все объединяется в работе одного приложения.

Затем нужно понять, как именно можно поделить ваш монолит на относительно независимые для разработки блоки. Чаще всего отталкиваются от команд разработки. Обычно деление зон ответственности уже есть, просто оно не воплощено в архитектуре. Например, в [Selectel](https://selectel.ru/) основной продукт разработки фронтендеров — это Панель управления, где клиенты могут приобретать и использовать наши услуги. При этом в самой панели также есть деление на продукты компании и, как правило, у каждого продукта, будь то выделенные серверы, «Облачная платформа Selectel», Managed Kubernetes и т.д., есть свои фронтенд-специалисты. То есть они уже развивают только определенную часть панели.

Выберите инструменты, которые помогут вам построить микрофронтенд-архитектуру. Сейчас появляется все больше библиотек, инструментов, которые помогают разработчикам на пути к микрофронтендам. Например, такая библиотека, как single-spa, которая объединяет несколько javascript-приложений в одно. Или спецификация Realms, которая позволит на уровне нативного javascript упростить разработку микроcервисов на фронтенде.

Современные фреймворки типа Vue и Angular также имеют некоторые возможности, позволяющие писать свое приложение так, чтобы в результате его проще было интегрировать в микрофронтенды.

Опыт компаний

В целом, большинство крупных компаний так или иначе перейдут на микрофронтенд-архитектуру для своих приложений. Известно, что в России так работают в «Додо-пицце», Avito, «Тинькофф банке».

[Как использовать микросервисы в веб-разработке — возможные проблемы и их решения](https://tproger.ru/articles/web-microservices/" \t "_blank)

[tproger.ru](https://tproger.ru/articles/web-microservices/" \t "_blank)

Selectel тоже стремится к этому. К микрофронтендам мы подходим как к неизбежной необходимости. В компании большой продуктовый портфель, включающий как базовые услуги дата-центров, так и современные PaaS. Как я уже писал, развитием продуктов занимаются, как правило, отдельные команды. И уже сейчас мы отмечаем сложности в их синхронизации при разработке и релизах.

Наш путь к микрофронтендам начался во многом из-за переезда с AngularJS на Angular нашей Панели управления. Сейчас мы в процессе миграции и поддерживаем гибридную структуру фронтенда.

Как известно, есть несколько способов миграции из одного фреймворка в другой. Первый — более дорогой и долгий — подразумевает поддержание приложения на старом коде и параллельное его переписывание.

Второй способ, который мы в итоге выбрали, — более мягкий. Когда мы внутри нашего текущего приложения подключаем другой фреймворк. И потихоньку, кусок за куском, переписываем код приложения. И старая, и новая версии работают одновременно. В процессе переписывания мы рефакторим код. А в течение рефакторинга выделяем части приложения, делаем их более изолированными, избавляясь от лишней связанности. Таким образом, мы создаем подобие микрофронтенд-архитектуры.

Как решать проблему общих ресурсов

Хорошим шагом к организации микрофронтендов может стать создание монорепозитория. Это git-репозиторий, внутри которого содержатся ваши npm-пакеты, зависящие друг от друга. Монорепозиторий поможет решить проблему с конфликтом версий библиотек и ведением разработки разными командами — все в одном месте, не нужно бегать по разным репозиториям и линковать пакеты.

При разработке Панели управления Selectel мы как раз используем монорепозиторий. Раньше у нас каждая услуга была отдельным git-репозиторием. В результате возникали сложности не только при локальной разработке, когда нужно было линковать пакеты, но и при обновлении библиотек и сборке в CI. Монорепозиторий решил практически все эти проблемы, причем на его создание ушло не так много времени. С точки зрения рабочих процессов монорепозитории облегчают организацию микрофронтендов.

Упомяну и очевидные вещи: не стоит исключать лучшие практики разработки (в том числе DRY, KISS, YAGNI) и простую коммуникацию — обмен опытом между фронтенд-разработчиками компании.

Выделим ключевое

Микрофронтенды — это деление монолитного приложения на семантически изолированные части и их независимая разработка.

Микрофронтенды имеют смысл, если:

1. У вас большое приложение.
2. Несколько команд разработки.
3. Вы все чаще сталкиваетесь с проблемами при выкате обновлений приложения.
4. Преимущества монолита обернулись в минусы.

Для остальных проектов микрофронтенды будут избыточны.

Ограничения микрофронтендов:

1. Остается проблема общих ресурсов. Полноценная изоляция, как в случае с микросервисной архитектурой в бэкенде, пока невозможна.
2. Требуется прослойка для оркестрации приложения, которая может вносить свой оверхэд.

Что поможет справиться с ограничениями:

1. Создание монорепозитория.
2. Использование специальных библиотек и инструментов.
3. Хорошая команда специалистов.
4. Использование лучших практик разработки (например, DRY, KISS, YAGNI).