Front-end архитектура. Предисловие.

Я enterprise front-end инженер. Как ни странно (меня это реально удивляет!), но в разговорах с некоторыми знакомыми front-end разработчиками (особенно начинающими, но встречаются и достаточно опытные) я часто встречаю странную позицию по отношению к архитектуре клиентской части. А именно, я часто слышу, нечто наподобие “архитектура клиентской части все время изменяется, это живой организм, поэтому её нельзя однозначно описать…”, или что-то вроде “на фронте нет чёткой архитектуры…” или “ну мы используем redux...”. Я хочу сейчас помочь начинающим front-end разработчикам немного проникнуть в этот вопрос и показать всю важность наличия архитектуры для развивающегося проекта. Самое интересно, что например back-end разработчики любого уровня осведомлены о базовых принципах архитектуры построение серверной части, но и так же часто считают, что на клиенте понятие архитектуры отсутствует. Если честно, такие времена были не так давно (буквально 10 лет назад), когда на клиенте архитектуры можно сказать отсутствовала, но об этом чуть позже и об различных подходах - ниже.

Ниже я попробую бегло описать описать основных вещах, о которых стоит подумать перед тем как разрабатывать систему, которые сильно повлияют на архитектуру приложения и тем самым полностью определят и стек и направление развития проекта.

1. **Сайт или приложение?** Это практически полностью определит стек и повлияет почти на все аспекты архитектуры. Ответ не дискретный! Приложение по сравнению с сайтом отличается большей автономностью, отзывчивостью для пользователя, связанностью различных подсистем, что делает приложение более дорогим или долгим в разработке. Крайние части в диапазоне ответов на вопрос «Это сайт или приложение?» это со стороны сайта — набор независимых html страниц, а с другой что-то наподобие PWA приложения. Честно говоря, оба крайних варианта редко встречаются в нише enterprise, обычно более подходящим решение является что-то между ними. На сколько мне известно, доминирующим решением в этом вопросе все ещё похоже часто остаётся приложение с архитектурой на подобие redux, когда мы используем одно глобальное хранилище для всего приложения и обновляем его по когда это необходимо, таким образом добиваясь высокой синхронизации (и этом числе зацеплённостью между компонентами системы! Что не есть хорошо!) в системе. Это приводит к тому, что с разрастанием проекта управлять им становится все сложнее и дороже.

2. На сколько это большое приложение? Сколько команд работают над ним? Это определит архитектуру в вопросах интеграции внутри приложения между командами. Так же нужно будет понять на сколько глубоко связаны команды или могут быть связаны в будущем? Если их связь не велика, то мы можем например смотреть в сторону лёгких интеграций — например интеграцию через гиперссылки, выставления общей библиотеки. Если же интеграции более глубокие или команд слишком много, то приходится думать в сторону интеграции во время сборки, или даже интеграции в рантайме (для уменьшения количества пересборок всей системы, что увеличит скорость доставки изменений, и мы сможем прийти к нормальному CI/CD, если команд уже становится реально много, на мой вкус — больше примерно 10-15 человек).

3. На сколько связаны подсистемы (компоненты) внутри каждой команды (степень статичности страниц)? Это определит архитектурный подход в общении между подсистемами. Ответ сводится опять к поиску баланса между чисто интерактивной архитектурой (когда любое изменение должно быть явно спровоцировано неким субъектом) и чисто реактивной архитектурой с полным графом зависимостей (когда каждый элемент системы может повлиять на каждый другой элемент системы). Промежуточными тут будут варианты как частичной автоматизации интерактивного подхода так и использование разных подходов для разных

частей. Конкретный пример где уместна интерактивная архитектура — какой-нибудь блог на с SSR (server side rendering), или набор относительно статических страничек. А реактивная архитектура будет уместна в сложных страницах содержащими динамические страницы, настраиваемые пользователем блоки, метрики\диаграммы\графики (тут явно нужно продумывать отдельный механизм интеграции между компонентами). Пример — что-нибудь на подобие трейдерского приложение или интерфейса управления промышленной системой. Если связей много, то так же придется приходится делать выбор между адресной коммуникацией между подсистемами, широковещательной, через некоторую шину или хаб (например через dom-события, или веб-воркеры, или BroadcastChannelApi и пр.).

Давайте честно, архитектура (в моей интерпретации) – это высокоуровневое (возможно сказать абстрактное) описание системы или её части, которая призвана помогать делать работу системы более консистентной и прозрачной. Под консистентностью я подразумеваю относительно единообразное использование подходов и практик. Под прозрачностью – возможность быстро разобраться в принципах работы системы, что прямо сказывается на скорости локализации проблем, внедрении новых фич, и как следствие – на стабильности и скорости (цене) развития системы. Если вам более-менее подходит такое определение, то из него сразу вытекает несколько следствий. Например: система у которой нет архитектурного описания (или о нем не знают и не могут относительно чётко сформулировать) становится менее прозрачной и консистентной с точки зрения разработчика. Или ближе к бизнесу: если у вас нет архитектуры в приложении, то при его росте (если проект долгосрочный) приложение станет явно дороже в обслуживании. Часто не замечать пагубного влияния отсутствия явного выделение архитектурных подходов на начальных этапах могут помочь качественный онбординг, высокий опыт и квалификация разработчиков, внедрение тесного общения во время разработки (например парное программирование, или консультационные встречи каждый день по различным мелким техническим вопросам) и пр. Все данные техники хорошо работают и при наличии описанной архитектуры. Но при развитии проекта это перестаёт быть хорошим решением (потому что непрозрачных и неконсистентных вещей накапливается так много, что система превращается в нечто просто непредсказуемое).

Это небольшое вступление, которое может дать пути к размышлению начинающему front-end инженеру. В этом цикле статей я попробую привести более конкретные примеры нескольких популярных архитектур front-end части с описанием и схемами.