Что-то полезное

Автор: автор чего-то полезного

By @JUSSIAR

Построение и архитектурное разделение сущностей в приложении React

Здесь речь пойдет о том, как мы можем думать в терминах react, как можем строить структуру, на что стоит посмотреть и о чем подумать, а главное зачем нужна каждая деталь и за какую зону ответственности она отвечает.

Компонент

Компонент отвечает за целостность какого-то куска логики и управление каким-то куском данных.

Страница

Страница это композиция компонентов и внедрение нужных сервисов + единица, с которой можно связать маршрутизацию.

Helper или Util

Кусок чистой логики, которая явно отсоединена от UI и ради атомарности и удобства проверки которой почти всегда выносится в отдельную инкапсулирующую сущность.

Тест

Файлик или кейс в разном понимании. Здесь ответственность за документирование и качество. Код с хорошими тестами можно документировать тестами. Идем в тесты и собираем предложения из тестовых кейсов – entity should have this behaviour for this input. По полноте тестов понимаем что как и когда делает сущность.

Директория common

Место в проекте для общего всего – компонентов, хелперов, типо, констант, доков (adr и прочее).

Директория dev

Место в проекте для генерации и хранения чего то для режима разработки. На практике можно использовать для чего угодно. Есть удобный пример – генерация моков из json файлов. Куда бы вы при необходимости засунули подобное в проекте?

UI-kit или какой-нибудь еще kit

Kit – набор инструментов для сборки чего угодно из атомарных и объединенных набором свойств частей. Kit чаще всего представляется нам как хранилище компонентов задизайненного UI, но есть и другие применения, например, feature-kit. Храним там всякие касотмные поведения и пере-используем – вполне себе.

Компонент кита

Button, Input, Container, Holder, Wrapper, Plane, Icon, Picture … .

Что угодно, что может быть атомарным, без логики и при необходимости параметризованным.

Hook

https://habr.com/ru/company/infopulse/blog/140456/

Ну и на практике реакта тоже самое.

Entry point

Точка входа или то, откуда все начинается, где происходит гидрация, рендеринг и все что угодно вплоть до открытия порта на прослушивания в системе – localhost be like.

Use-cases

Можно просто перевести, но посути это всякие места, которые содержат описания разных флоу. Например, есть сервис аутентификации путем разных флоу – партнеры, SSO, соцсети, номер телефона, почта. Где то на подобную штуку находится switch – чаще всего внутри app/service, а описание поведений – уже зона ответсnвенности use-cases.

Types

В типизированном подходе добавляется необходимость куда то выносить пере-исползываемые типы и интерфейсы – все это попадает сюда.

Apps или Services

Куски приложения, реализующие какой то функционал. Например аутентификация. Можно вынести сервис комментариев, можно вынести сервис профиля и внутри обрабатывать все нужные куски логики.

Конфиги

Конфигурации тестов, линтеров, сборки, плагинов для сборки и всего чего угодно – нужно везде и мы не забываем про это.

API

Часть проекта, где описываем контракты взаимодействия со сторонних интерфейсами и клиентами. Backend, аналитика, что угодно еще – все запихиваем сюда и если надо разбиваем по сервисам.

Entries или Entities

Сущности или модели данных, используемые фронтом.

Константы

Имена, хранимые литералы или что угодно неизменяемое и относящееся к проекту напрямую.

Окружение

.env файлик или параметры при запуске скриптов в package.json

Другое

Redux составляющие, например. Ну или любые вводные сторонних либ.

Server-side rendering (SSR)

Эта часть затрагивает приоритеты в разработке, затрагивает проблемы SPA, проблемы CEO и всякое еще, что заставляет не спать неравнодушных и писать фреймворки. Мы что-нибудь здесь скажем, про что-нибудь подумаем и посмотрим насколько оно все полезно.

Подход

В целом подход довольно понятный со стороны без погружения – надо отрендерить что то на сервере и прислать готовое.

Двойное мышление

Этот подход заставляет разработчика думать о рантайме исполнения кода дважды – как код отработает на сервере и на клиенте. Например рантайм реакта рендирит на стороне сервера только каркас, а все запросы из методов жизненного цикла не отправятся просто потому что жизненного цикла нет и скоуп того же коллбэка в useEffect может содержать что угодно, но на сервер оно не просочится, н сработает там. Надо полумать еще про DI и окружение. Так например АПИ браузера и ноды могут содержать один интерфейс, но разную реализацию (даже не координально, но тем не менее).

Чем хорошо

Первая загрузка быстрая, поисковики находят проброшенные SEO-данные. Ну и что-нибудь еще хорошо

Чем плохо

Сложнее писать и поддерживать, на сервак нагрузка больше. Ну и что-нибудь еще плохо.

Инструменты и подходы

Вообще говоря речь про эмуляцию специальным фреймворком – например NextJS для реакта. Либо другой подход – пишем свою либу и как то более тупо просачиваем гидрированные данные в движок приложения.

Примеры из практики

Пример с тремя сущностями – html as string + разворачиваем хэшированную статику на CDN и отдаем урлы на их адреса – удобно и очень быстро. И вроде даже не дорого с uglify.

Кастомизация

Кастомизация какая угодно – решайте только свои задачи и не тащите жуть из фреймворков. Трудитесь, так сказать.

Цитата из смешариков:
«Труд – это время, проведенное с пользой. Было бы время»

Проектирование

Ну соотвественно тут речь про микросервисы скорее всего, иначе зачем столько мучений для рендеринга какого-то лендоса. Или мы наконец то делаем что то больше, чем сайт с рецептами пирогов?

Опыт собеседований

Про опыт собеседований можно говорить очень много и у каждого собеседующего свой стек вопросов и свой взгляд на знания кандидата. Но в любом случае есть какой то свод проверяемых компетенций, который должен позволять понимать реальный уровень, ширину и глубину бэкграунда, а также грейд и количество денег, которые команда будет готова дать вам при найме. Здесь успеем поговорить о вопросах и о том как строятся системы максимально объективной оценки людей.

Язык программирования

Вопросы в ЯП, его работу с памятью, базовые типы, всякое такое, в чем нельзя плыть. Если решаете таску на ЯП и видите очевидное применение чего-то, что помните смутно – НЕ НАДО. Лучше потом скажите если спросят про альтернативу и выкручивайтесь, а сами не подсказывайтесь на ровном месте.

Core

Вопросы про контексты, замыкания, работу, компиляторы, интерпретаторы – надо шарить.

База

Многим важно, чтобы знали базово про многие вещи – например базовый порт сервера, работа DNS, уровни доменов, типы запросов, FPS. Надо быть готовым не проглотить язык.

Фреймворки

Ну тут довольно очевидно – надо уметь на вопрос почему этот фреймворк или в какой ситуации именно он. И кстати можно добавить, что:

«потому что я его знаю»

Шутка.

Не надо.

React

Хуки, работа со стейтом, ре-рендер, мемоизация, ЖЦ, декомпозиция, дебаг.

Браузер

Очень прикольный вопрос: вот вы ввели URL в браузере – событие один, и вот вы видите, что все прогружено и готово к работе – событие два. Расскажите все по порядку, что было между ними? И тут приколы про все веселье с сетью, после чего надо подробно рассказать что делает браузер, в т.ч. про деревья DOM и CSS, про загрузки статики, про всякое еще там, про кэш что-нибудь, про CORS и его запрос OPTIONS.

Зарплата

Про это надо помнить когда вы подаете резюме. При скрининге можно сильно завысить или продешевить и потом на вас будет взгляд под углом грейда и надо быть аккуратнее.

Вы можете получить первый вопрос вида: расскажите что такое Gateway.

Или вопрос: а как бы вы проектировали WatsAPP?

Алгоритмы

Надо уметь хотя бы немного в алгоритмизацию для секций программирования и не теряться на подбном – на это смотрят и эти секции роляют очень много где. На фронте несильно, но плюс в карму при хорошем результате обеспечен. А плюс в карму на интервью увеличивает плюс в кошельке.

Как рассуждать

Не надо бояться рассуждать вслух – молчание пугает гораздо больше, чем излишняя говорливость. Инициатива наказуема, но не всегда – здесь могут отметить, что держите микрофон и транслируете поток мысли, что важно при дискуссии или просто обсуждении.

Вопросы

Многие вопросы или задачи подразумевают сразу задать вопрос. Уточнение ограничений или контекста или ресурсов – все это важно и если вам оно надо – надо спрашивать.

Глубина знаний

Могут начать копать куда-то, где кажется, что плаваете. Ну и ладно. Желательно до этого не доводить, но в целом есть то и другие компании…

Чекпоинты и касания

Можете коснуться чего-то стандартного, но не из области интервью. Например какой то вопрос вида: а на какой порт по умолчанию шарится сервер? Это вопросы в ширину и тут не надо знать все. Просто хотят понять – сталкивались ли с подобным или хотя бы гуглили/слышали. Кругозор – показатель правдивости опыта. Если человек за 3 года работы много где не может расшифровать АПИ, ссылаясь на то, что он компонентики рисовал – это беда.

Приукрасить !== ******

На собеседовании отношение зависит от резюме и добавить «статусности» и ярлыков бывает полезно, но не говорите про что-то, о чем вообще не имеете понятия. Хотя бы чихнуть в ответ что-нибудь надо уметь.

Идеи проектирования фронта и проблемы в распиле монолитов

Идей на самом деле много и суть в том, что все очень зависит от выбора технологического стека, от наличия финансирования проекта, целей проекта в бизнесе и еще много чего. Но мы затронем ту часть проблематики, которая очень важна людям, готовящим платформы к старту, принимающим решения по работе с легаси, стеком и вообще всем тем, чем живет разработка.

API и schemas

Api часто содержит схемы, которые композируют разные сущности.
Тот же микросевис фронта может имет контракты на входе,
которые заставляют что-то объединить – про это надо думать.

Директория common

Надо помнить, что при распиле – это ваша боль, библия и камасутра в одном флаконе.

DI

Тоже боль – отломить нужную реализацию очень просто, а при нехватке экспертизы (что очень часто на практике) можно что-то разломать. Сюда же не относится конечно, но вспомним про пресеты – разломать стилизацию или что-то зависимое очень легко – надо про все думать.

Тесты

Интеграция и e2e – дикая боль распила.

Apps

Аппы часто шарят свой «апи» на весь сервис и надо думать куда его выносить – скорее всего кор.

Kit

Кит сразу надо выносить – даже можно не думать. Распил, переиспользование, версии – все тут.

Пакеты

Кор. Кор. И еще раз кор. Тянуть лямку копипасты кастомных либ – самая тупая и нераспределнная затея, за которую невозможно нести ответсвенность.

Версионирование

Нужны поддержки версий. Для этого нужен свой registry по-хорошему и паблишить туда версии своего кастома. Тогда жизнь спокойнее и развивать разные части можно без боли в разном темпе. Сюда же версионирование кита – куда же без итераций редизайна и дизайн-ревью. На думайте про дубликацию и кэш – оно ведь все прогрузится и еще там что нибудь мало ли))

Сваггер

Надо аккуратно разрезать. Всегда все любят линейность – она гибкая. Распилил и ничего не сломал.

Распил

Распил есть распил – тут про микросервисы и про зоны ответсвенности.

Команда

Распил == разделение команды и разделение зон ответсвенности.

Ответственности

Они самые. Как на уровне людей, так и приложений. Нельзя утерять экспертизу. Нужна единая дока с разводящими, нужны эксперты, нужно проводить встречи, нужно синхронизировать, тестировать, поддерживать. Нужно думать про альтернативы на случай форс-мажоров. Как мы видим по пример 24 февраля – они случаются и что-то может быть нужно заменить или сделать селф-хостед и об этом кто-то должен думать.

Дебаг и тесты

В погоне за качеством продукта придумали методологии разработки, заставили гуманитариев думать о железках – наняли продуктовых менеджеров, отпочковали профессию инженера QA и сделали кучу других непростительных ошибок(шутка). Мы же поговорим про часть, где проблемы не выходят за рамки рукоделий разработчиков и немного сформируем ту сторону, в которую всем нам хочется двигаться в погоне за идеалами.

Тестирование

Ну тут в целом ясно. Coverage все дела.

Виды тестирования

Юнит, интеграция, e2e. Можно еще нагрузку, но тут как пойдет)

Моки

Нельзя стучаться в реальные базы, нельзя стучаться в реальные апи или ломать реальные данные ради проверки. Нужны всякие штуки, повторяющие поведение без боли для окружающих.

Контуры

Круто если есть контуры кроме прода. Staging, testing – это все оно. Там можно без страха и боли все ломать и смотреть че будет.

Прогоны

Неплохо при написании тестов их самому специально сломать и посмотреть что они сломались. Ато может тесты настолько «хороши», что вообще всегда работают.

CI-CD

Ну интеграция и деплоймент это ясно – не забваем проверить и на серваке, ато вдруг че из окружение или конфигов подъехало?

Покрытие

Тестов много не бывает. И помним про документацию тестами.

Окружение

Мокать можно и environment. И нужно. Надо прорабатывать сценарии чего угодно, потому что именно оно и произойдет, но уже точно не идеальные условия.

Браузер

Браузер очень полезный инструмент - посмотреть что все хоть как то работает уже нормас. Проверить на разных экранах – мобилки там и тп. Пример проблем – тот же ховер и медиа на ширину устройства – вечная боль. Там есть какой то дебаггер – вроде полезный.

Dev-Tools

Мощный интструмент. Маст-хэв. Посмотрите че нибудь на ютубе про это.

Sources

Сорсы – важная штука чтобы смотреть на то, что вообще лежит на клиенте, ато может микросервисы из движка прилетели неразбитым паком и пользователь грузит 5 реактов себе на страницу. Оптимизация однако.

Network

Полезно смотреть что и как долго приходит, при необходимости блочить урлы и смотреть что будет дальше.

Cross-браузерность

Проверяйте везде. Особенно сафари. Самая гадкая штука со своими политиками. Пример расскажу про синхронность пользовательских действий.

Caniuse

Классный сайт – рекомендую.

CORS и cross-domain

Это речь про проверки запросов на доступы и разрешения и проблемы с работой фрейма, например. Бывают траблы, с которыми приходится идти к другой компанде и просить ввести новое событие для синхронизации.

Логи

Пишите логи если надо, катите в пробные контуры, добавляйте мидлвари, о чем речь дальше.

Мидлвари

Омега полезная вещь – в редаксе советую.

Есть еще тул полезный под это дело:
https://github.com/reduxjs/redux-devtools

Режимы

Mode === ‘development’ – истории отсюда.

Инструменты и архитектура для фуллстек разработки

Фуллстек как роль в команде - очень неопределенная сущность сама по себе. Знать наверняка о зоне ответственности такого человека сложно. Но понятие фуллстек в отвлеченности от ярлыков людей в разработке обязано покрывать весь процесс и быть весьма широким. Здесь мы скажем пару слов о проблематике и о том, что используют сейчас, чем пользовались раньше и как оно вообще там крутится (если б я об этом хоть что-нибудь знал…).

Инструменты

Да их много в целом – все зависит от задач. Подробнее далее.

Языки

Важно какие ЯП юзаете и почему. Какая команда, какие даньги. Какие цели, можно ли что-то упросить. Есть ли математика/ан-дан.

Направленность

Бизнес цели. Что мы делаем, зачем, для кого? Проектирование АПИ, все дела.

Архитектура

Ну тут паттерны и всякое разное.

Подходы

Микросервисы. Пожалуйста, микросервисы. Но опять же цели.
История стартапов – вообще своя история.

Согласованность

Транзакции в несколько баз, финансы – все это сложно и объемно, надо думать.

Отложенная согласованность – очень важная штука. Шины, мессадж брокеры.

Распределенность

Рапределенные транзакции и зоны ответсвенности. Да, опять они, но че поделать)

Данные

Это опять проектирование АПИ.

Скорость

REST или шина?

Языковой полиморфизм

Писать несколько кусков на одном ЯП – удобно! Но не во вред важным вещам.

Команды

Время разработчиков самое дорогое – надо думать об этом.

Задачи

Цели, бизнес и всякие другие слова, приходящие во сне к архитекторам.

Ответственность

Да. Опять. Важно. Нужно.