Граница по границе контроля

где на проекте проводить абстракции, а где нет

Дмитрий Якубовский

«А это куда положить?»

Написал хук для фильтров — куда его положить?

Маппер ответа API — в shared или в entities?

Компонент нужен на двух страницах. Куда?

Вопрос про границы

```
src/
├─ pages/
├─ features/
├─ entities/
└─ shared/
   └─ useFilters.ts ?
```



Это архитектурное решение

Ошибка стоит дорого

Кто я и почему об этом говорю

5+ лет

в разработке



рабочая группа



архитектура фронтенда

Все эти границы я проводил руками — и ошибался

SCloud — сервис 1С в облаке №1

Браузер → 1С в облаке





scloud.ru

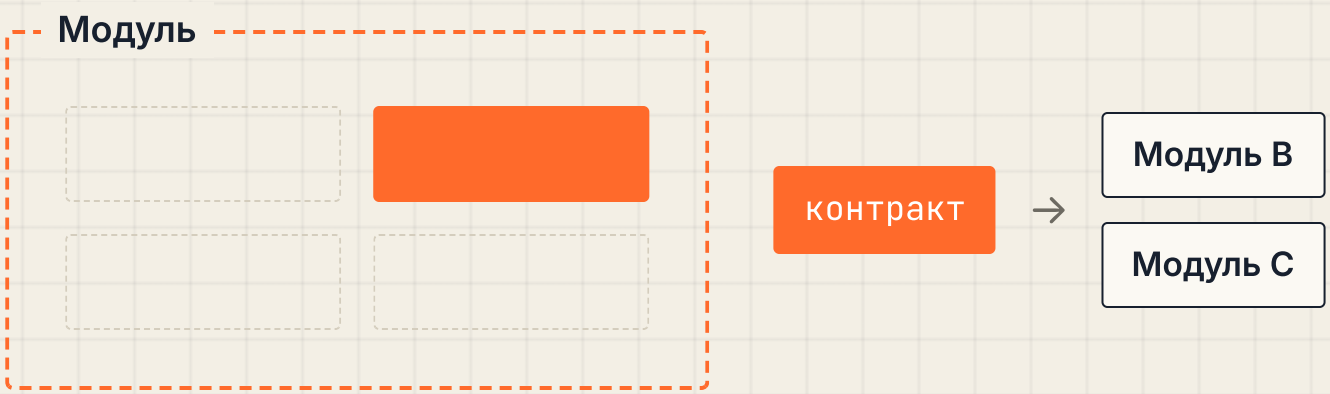
58 000+
пользователей

50+
конфигураций 1С

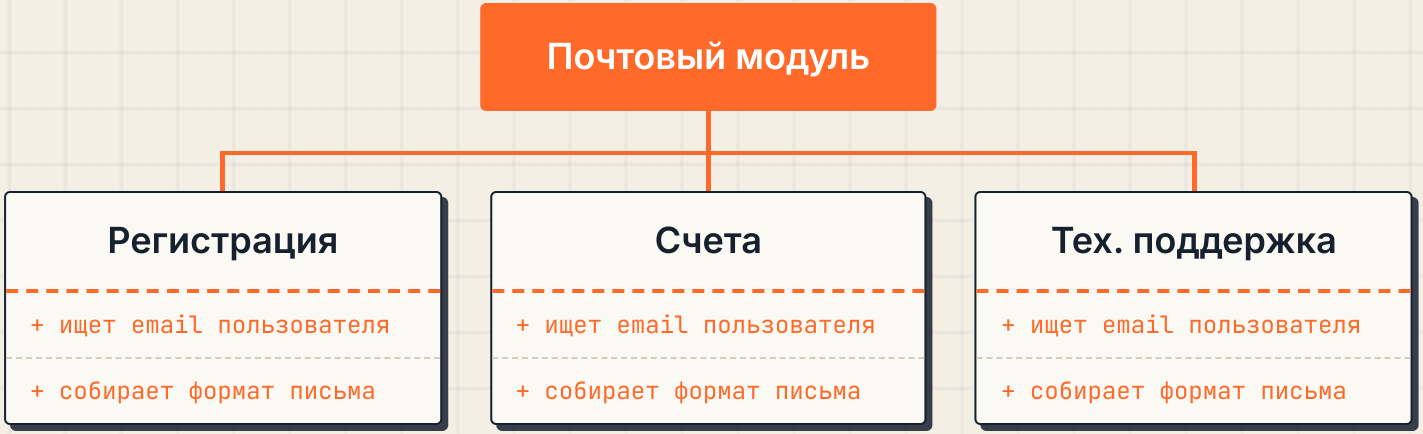
99,9%
SLA

24/7
поддержка

Так что же с границами?



Наш почтовый модуль



Рефакторинг = удаление кода по всему проекту

Что на самом деле нужно счетам?

Модуль счетов

- id пользователя
- сумма платежа
- ~~email пользователя~~
- ~~формат письма~~

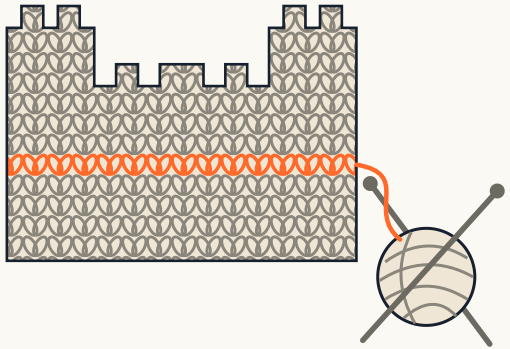
→

Почтовый модуль

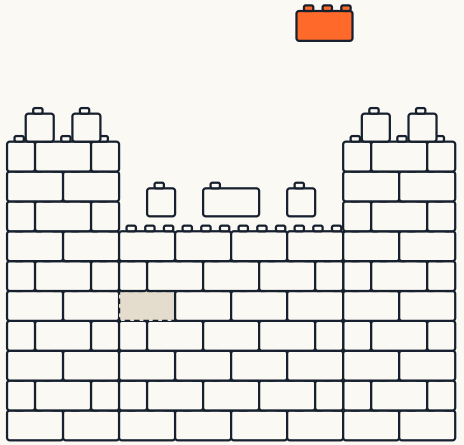
```
notifyPayment(userId, amount)
```

Вязаный замок и замок из Lego

Rich Hickey · Simple Made Easy · 2011



Связан спицами



Собран из Lego

Лифту не нужно знать про ваш диплом

Человек

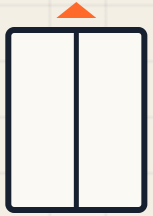
~~высшее образование~~

~~водительские права~~

~~профессия~~

вес

габариты



Пассажир

вес

габариты

Не «человек», а «пассажир»

```
type Passenger = {
  weight: number
  size: Size
}

function ride(passenger: Passenger): void
```

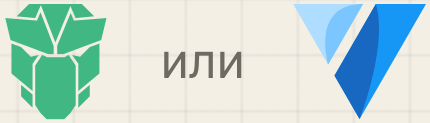
Дисклеймер

1. Доклад не про FSD
2. Нет серебряной пули
3. Примеры на Vue

Границу стоит проводить там, где кончается контроль.

Обёртка над чужим

Кнопка из библиотеки — везде



Button

Header.vue | Cart.vue | Login.vue | Orders.vue | Filters.vue | Settings.vue

Одно из двух

Мажорная версия

Своя дизайн-система

Правим все места использования



А так — работает

shared/ui/VButton.vue

```vue
<script setup lang="ts">
import { computed } from 'vue'
import Button from 'primevue/button'

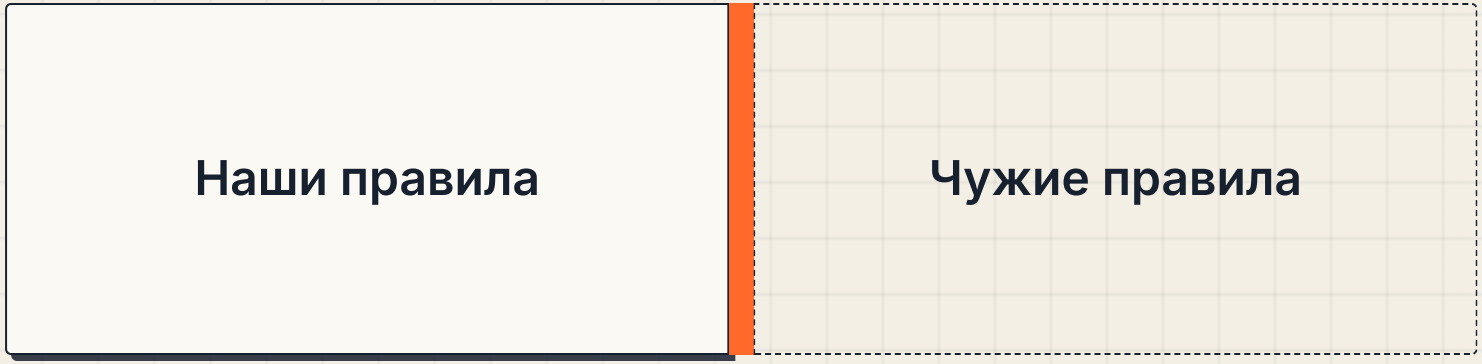
type Variant = 'primary' | 'ghost' | 'danger'

const { variant = 'primary', wide = false } = defineProps<{
  variant?: Variant
  wide?: boolean
}>()

// всё знание о библиотеке живёт здесь и только здесь
const severity = computed(() => ({
  primary: 'primary',
  ghost: 'secondary',
  danger: 'danger',
}[variant]))
</script>

<template>
  <Button :severity="severity" :class="{ 'w-full': wide }">
    <slot />
  </Button>
</template>
```

Таможня



Барьер там, где начинается чужой код

Колокация

Сложили рядом



```
src/shared/ui/toast/
├── VToast.vue
├── useToast.ts
├── formatToast.ts
└── index.ts
```

То, что меняется вместе, должно лежать вместе

По технической природе

```
components/ VToast
composables/ useToast
utils/ formatToast
```

По смыслу

```
toast/
  VToast · useToast · format
```

Сегменты — та же идея, уже названная

Feature-Sliced Design

Правила организации кода фронтенда

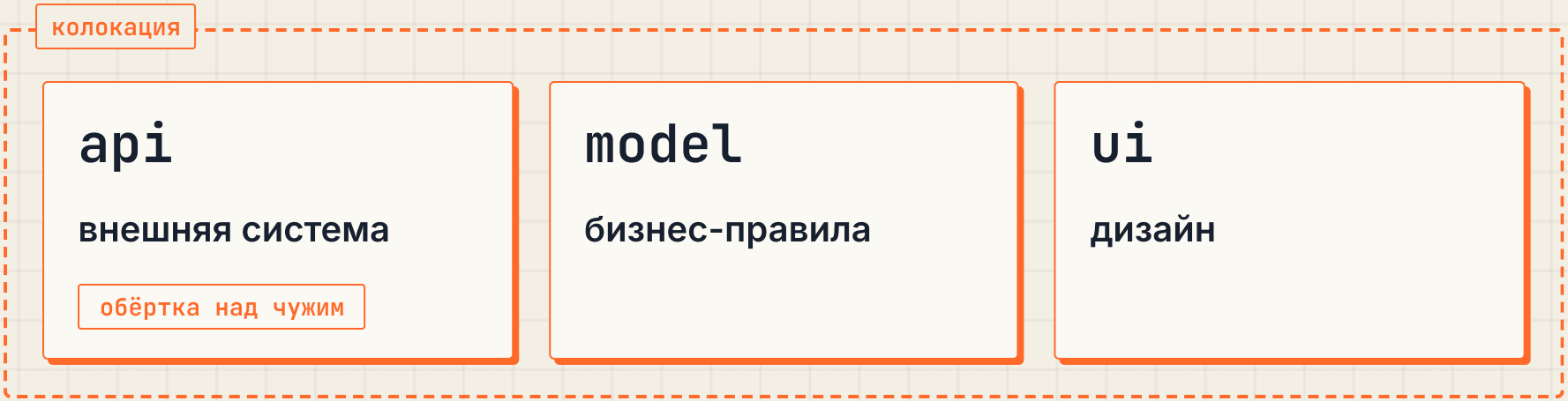
- Единообразие
- Изоляция изменений
- Язык бизнеса

fsd.how/ru

Сегменты

```
entities/order/
├── api/
│   └── orders.ts
├── model/
│   └── useOrders.ts
├── ui/
│   └── OrderCard.vue
└── index.ts
```

Не три папки — три причины меняться



api — обращение наружу и нормализация

entities/order/api/orders.ts

```ts
interface OrderDTO {
  order_id: number
  created_at: string
  client_name: string
  total_sum: string          // бэк отдаёт сумму строкой
  status: 'new' | 'in_progress' | 'done'
}

const mapOrder = (dto: OrderDTO): Order => ({
  id: dto.order_id,
  createdAt: new Date(dto.created_at),
  clientName: dto.client_name,
  total: Number(dto.total_sum),
  status: dto.status
})

export const getOrders = async (): Promise<Order[]> => {
  const response = await fetch('/api/v1/orders')
  const data: OrderDTO[] = await response.json()
  return data.map(mapOrder)
}
```

`snake_case` → `camelCase`

`string` → `number`

`string` → `Date`

model — логика поверх нашего контракта

entities/order/model/useOrders.ts

```ts
export const useOrders = () => {
  const orders = ref<Order[]>([])
  const isLoading = ref(false)

  const activeOrders = computed(() =>
    orders.value.filter((order) => order.status !== 'done')
  )

  // бизнес-правило, а не форма данных
  const canBeCancelled = (order: Order) =>
    order.status === 'new' && order.total > 0

  const load = async () => {
    isLoading.value = true
    try { orders.value = await getOrders() }
    finally { isLoading.value = false }
  }

  return { orders, activeOrders, isLoading, canBeCancelled, load }
}
```

ui — говорит на языке домена

entities/order/ui/OrderList.vue

```vue
<script setup lang="ts">
import { useOrders } from '../model/useOrders'
const { activeOrders, isLoading, canBeCancelled, load } = useOrders()
onMounted(load)
</script>

<template>
  <VSpinner v-if="isLoading" />
  <ul v-else>
    <li v-for="order in activeOrders" :key="order.id">
      {{ order.clientName }} — {{ order.total }} ₽
      <VButton v-if="canBeCancelled(order)" variant="danger">
        Отменить
      </VButton>
    </li>
  </ul>
</template>
```

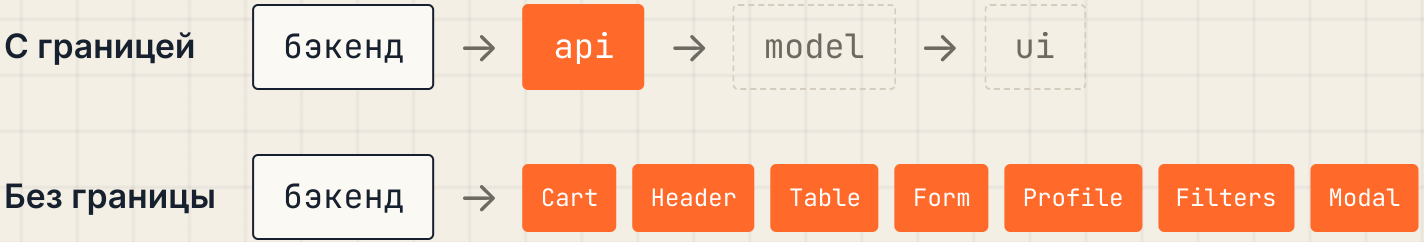
Чего здесь нет

~~`order_id`~~

~~`Number(total_sum)`~~

~~`парсинг даты`~~ в шаблоне

Бэкенд поменялся. Что дальше?



Второй эффект — для меня главный

1. Макет
2. Свой `Order`
3. `model` + `ui` на моках
4. `api` — последним

Фронт и бэк не блокируют друг друга

«А если маппинг один в один?»

Критерий — кто по ту сторону границы

Чужой сервис

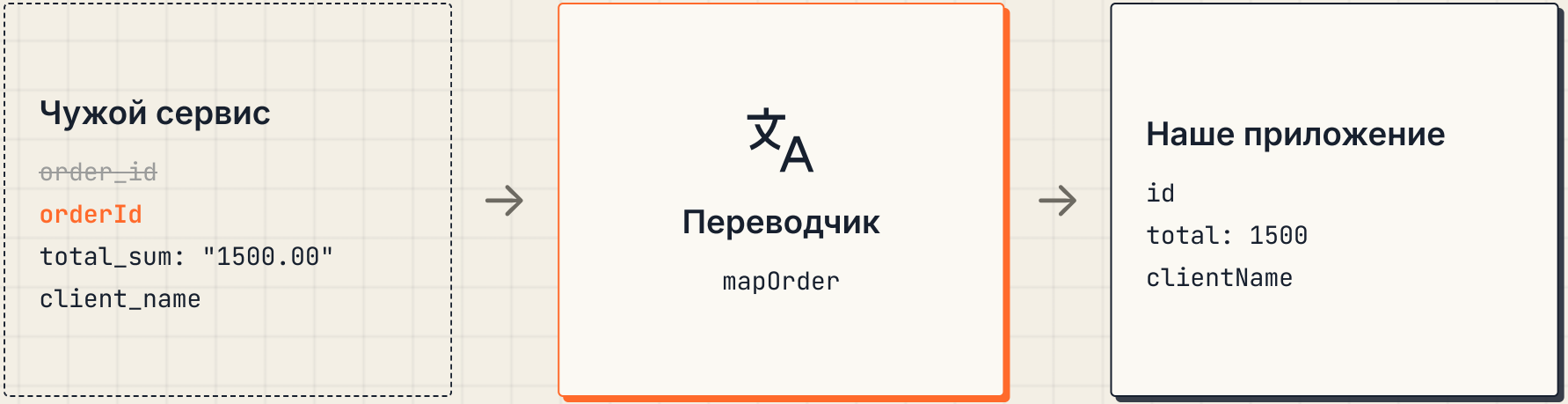
Ставим всегда

`антикоррупционный слой`

Свой бэкенд, CRUD

Можно не ставить

Антикоррупционный слой



Чужое изменилось — правим одно место

Граница постфактум

Вставили границу — один файл

entities/order/api/orders.ts

```ts
interface OrderDTO {
  id: number
  client: { first_name: string; last_name: string }
  amount: { value: string }
}

const mapOrder = (dto: OrderDTO): Order => ({
  id: dto.id,
  clientName: `${dto.client.first_name} ${dto.client.last_name}`,
  total: Number(dto.amount.value),
})

export const getOrders = async (): Promise<Order[]> => {
  const response = await fetch('/api/v1/orders')
  const data: OrderDTO[] = await response.json()
  return data.map(mapOrder)
}
```

api — правка

model — не тронут

ui — не тронут

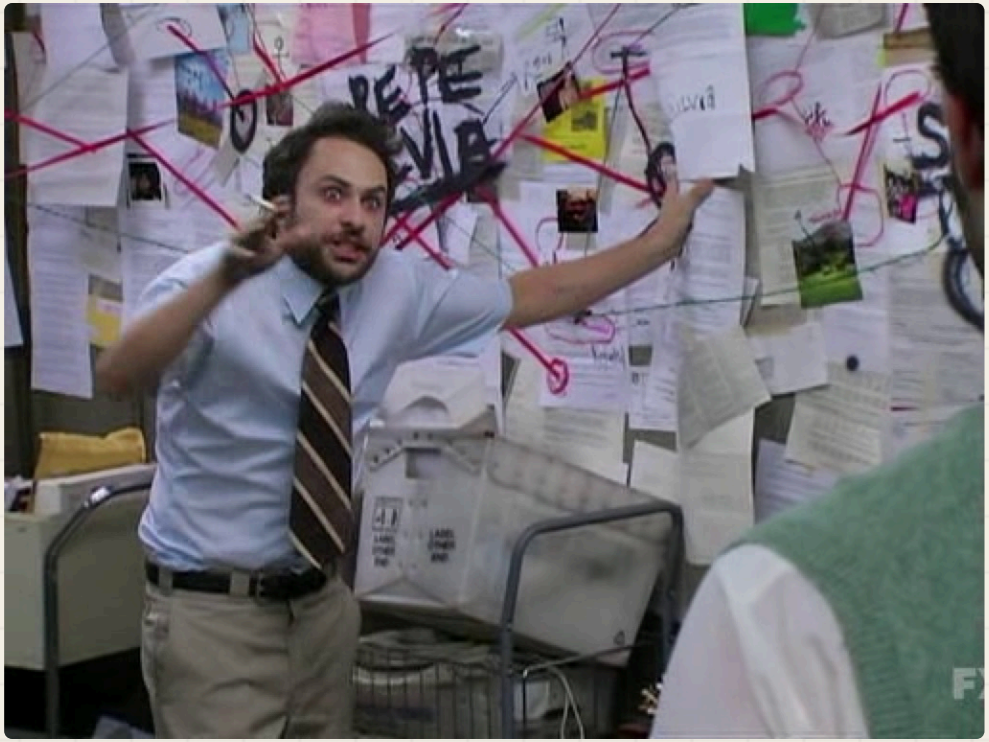
Границу не обязательно ставить сразу. Важнее уметь вставить её вовремя.

Провал: ранняя абстракция

Сделал строго по документации

```
src/
├── entities/organization/
│   ├── api/          # запросы по организации
│   ├── model/        # типы, стор
│   └── ui/           # карточка организации
├── features/create-organization/
│   ├── api/          # запрос создания
│   ├── model/        # валидация формы
│   └── ui/           # форма
└── pages/organization-create/
    └── ui/           # страница, которая всё это собирает
```

И что получилось?



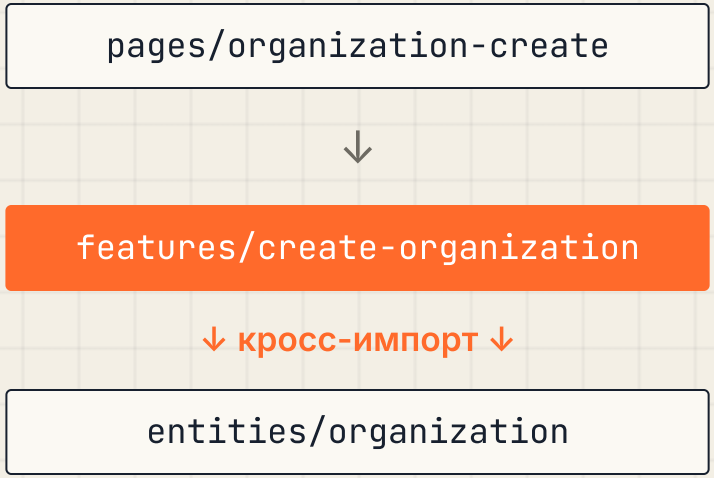
Что пошло не так

Всё на **одной странице**

Обёртка ради обёртки

Кросс-импорты

Один экран — **три места**



По правилам — и хуже, чем кучей

Как чинил — укрупнил обратно

```
src/pages/organization-create/
├── api/
├── model/
└── ui/
```

Внутри — те же `api` / `model` / `ui`

Выношу при втором потребителе

Здесь ранняя абстракция обошлась дороже поздней

Критерий: когда выносить

Общепринятый ответ

«Когда выносить общий код в отдельный модуль?»

1. Правило трёх
2. Второй потребитель
3. DRY: правишь в двух местах

Правила верные. Но у каждого есть место, где оно ломается.

1. Правило трёх: «похожие» — не «одинаковые»

Три похожих куска → вынесли

Условие → флаг

Ещё условие → ещё флаг

И ещё...

Через полгода →

```
function renderCard(
  item: Item,
  compact: boolean,
  withIcon: boolean,
  isAdmin: boolean,
  skipCache: boolean,
) {
  if (compact) { /* ... */ }
  if (withIcon) { /* ... */ }
  if (isAdmin) { /* ... */ }
  if (skipCache) { /* ... */ }
  // ...
}

renderCard(item, true, false, true, false)
```

Дырявая абстракция хуже дублирования

DRY vs YAGNI

DRY
не повторяйся

vs

YAGNI
не понадобится

«Дублирование гораздо дешевле неправильной абстракции» — Сэнди Мец

```
// косметика
renderCard(item, { compact: true, isAdmin: true })

// лечение
renderAdminCard(item)
renderCompactCard(item)
```

Сгруппировали по «это всё про пользователя»

```
// визуальная задача: добавляем цвет. И заодно меняем ключ
localStorage.setItem('user_data', JSON.stringify({
  id, name, role, avatar,
  color        // ради этого всё и затевалось
}))
```

Визуальная задача уронила права доступа

Если завтра одно из этих мест изменится — должны ли остальные измениться вместе с ним?

Да — это общность, объединяй

Нет — это совпадение, связывать нельзя

Где ещё правила ломаются

2. Потребители не равнозначны

- Из теста — не сигнал
- Из соседней фичи — слабый
- С другого экрана — сильный

3. Цена ошибки несимметрична

- Рано и зря — дёшево
- Поздно, на дубль налипло — дорого
- Рано, и наросло — очень дорого

Как я решаю сейчас — три сигнала

1. Третий раз — и хотя бы раз вне исходной страницы
2. Потребитель, которого не было в сценарии
3. **Уже правлю в двух местах синхронно**

Код меняется под новые реалии — это нормально.

Единый принцип

Три места — один ход

Пример	По ту сторону границы	Что сделали
`VButton`	чужая библиотека	свой контракт пропсов
`mapOrder`	бэкенд, внешний сервис	свой тип вместо DTO
данные пользователя	хранилище + чужие изменения	разделили по причине изменения

Во всех случаях по ту сторону границы — то, что я не контролирую

Контроль — не бинарный. Это шкала



Свой код
в своём модуле

Бэкенд
своей команды

Чужое
публичное API

Обернуть всё — паранойя

Когда граница окупается

цена изменения →

Смотрю по ситуации

Ставлю

 UI-библиотека · чужое API

Не ставлю

свой CRUD

Смотрю по ситуации

как часто меняется без меня →

Где я границу НЕ ставлю

CRUD один в один

Нет бизнес-логики

Свой бэкенд — соседний чат

→ завязываюсь прямо на DTO

Осознанный размен

Маппер появится, когда за него заплатит реальность

Моя позиция: 70 / 30

70

Строгие границы по умолчанию

- обращения к внешним системам
- UI от всего остального
- чистые функции от грязных

30

Исключения

- нечего прятать
- всё под вашим контролем

Нормально или самодур?

Тимлид пересматривает архитектуру каждые две недели. Это нормально?

Егор Бугаенко: держитесь за такого человека

Пересматривать — не страшно

Живой проект

перестраивается под изменения

с опытом видно, что сделать лучше

Мёртвый проект

трогать страшно

двигаем как получится

Пересмотр архитектуры — не провал, а её работа

Всё это работает без FSD

Обёртка над библиотекой	работает
Колокация	работает
Свой контракт вместо чужого DTO	работает



дала имена. Имя — не источник.

Что забрать с собой

Начинаете

Приёмы как шаблоны

Пишете сами

«Что дешевле откатить?»

Решаете за команду

Язык цены границы

Три строчки

Границу ставь по границе контроля

Абстракцию заслужи

Цена границы — не код, а её неправильность

Дмитрий Якубовский

yakubovskuy.ru

@yakubovskuy

dyakubovskiy

Скилл для Claude по докладу

`github.com/dyakubovskiy/boundary-by-control`