

# **Как фронтендеры изобретали чистые функции и TDD**

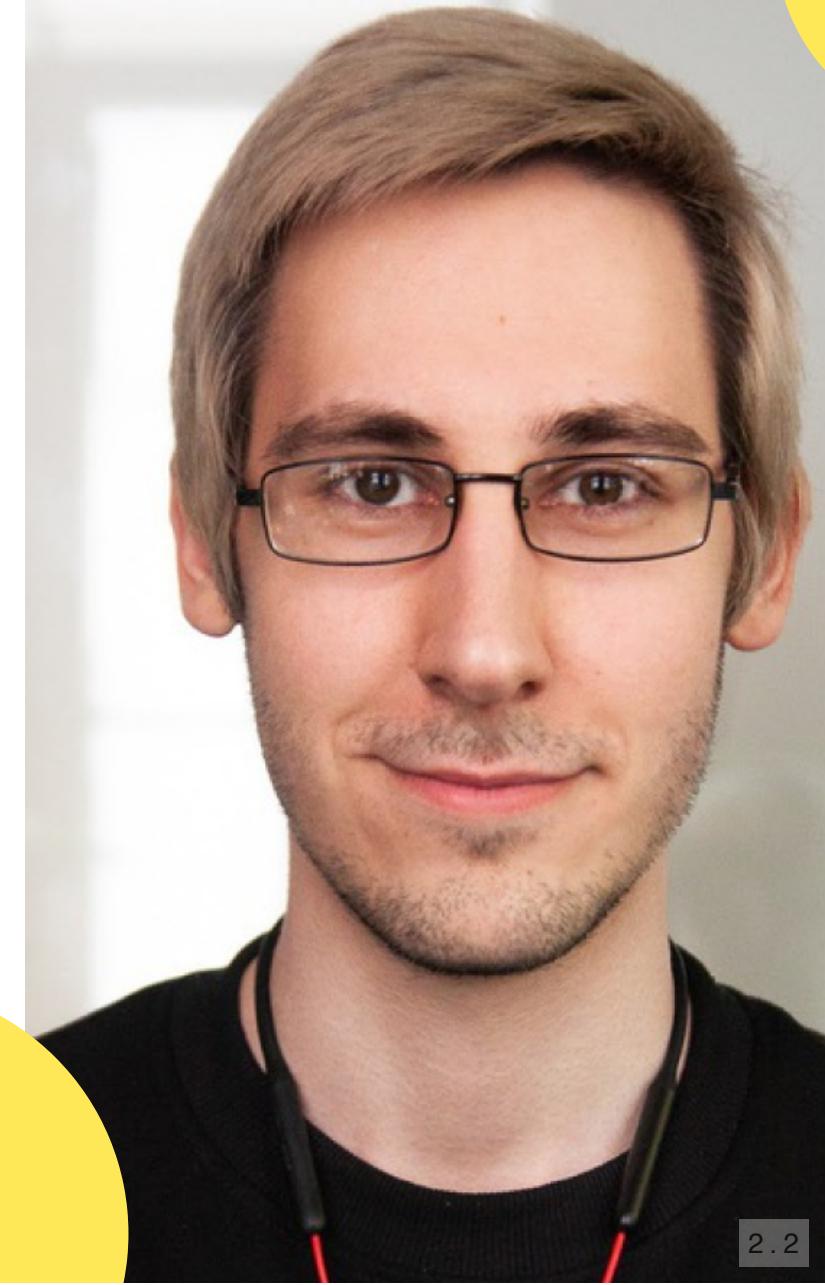
# **История меня**

# Дима Богер

*Python Backend Lead, Cindicator*

- Организатор **PiterPy Meetup**
- Тимлид
- Иногда преподаю

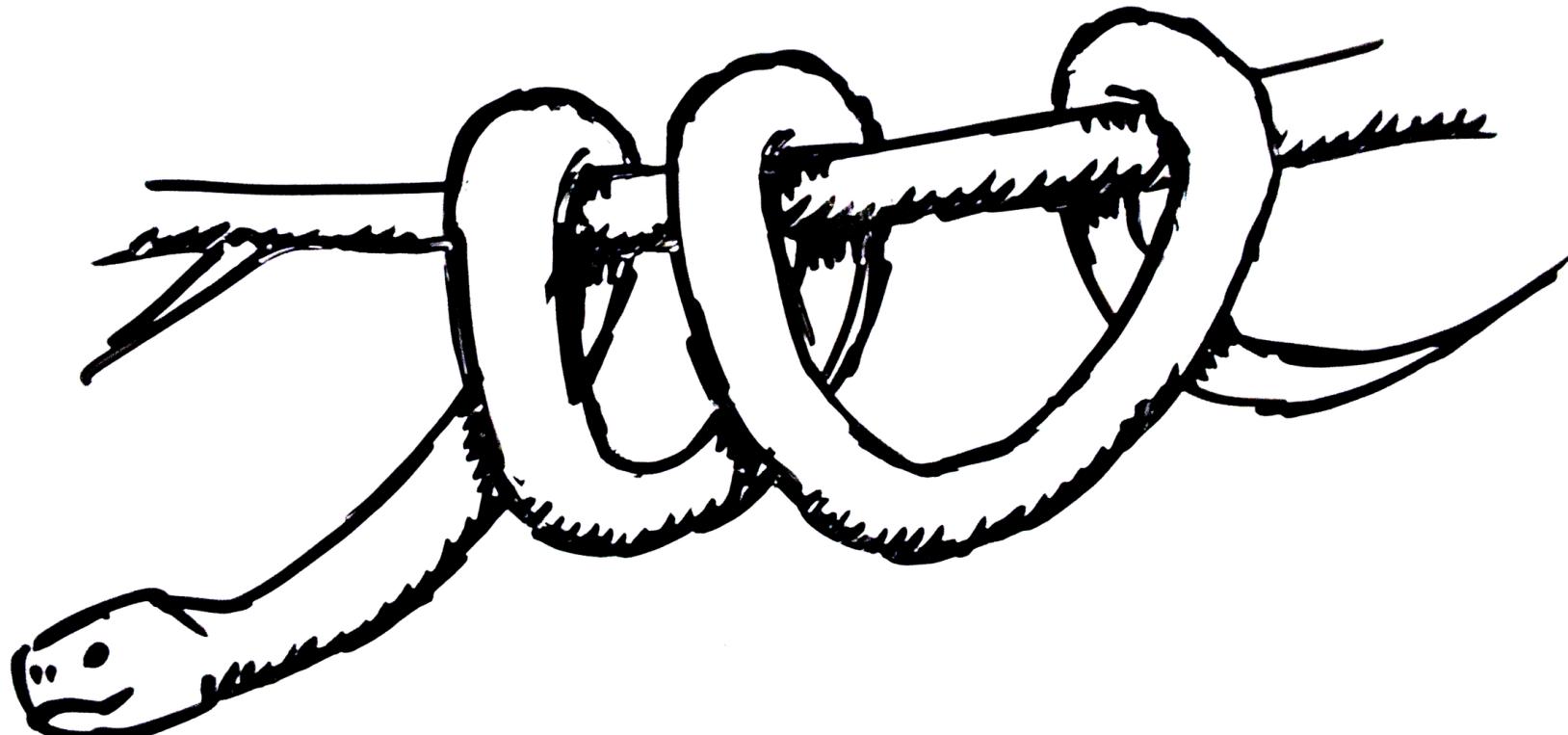
Люблю **Python** и **Vue.js** неосознанной  
любовью



# Я был тимлидом фулл-стек команды

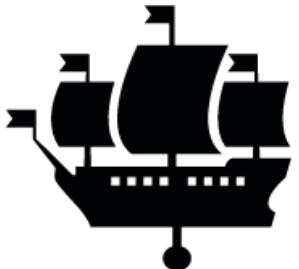


# Я был бекенд-лидом



Hellooo...

А теперь я здесь



L JS T

# Почему мне нельзя верить

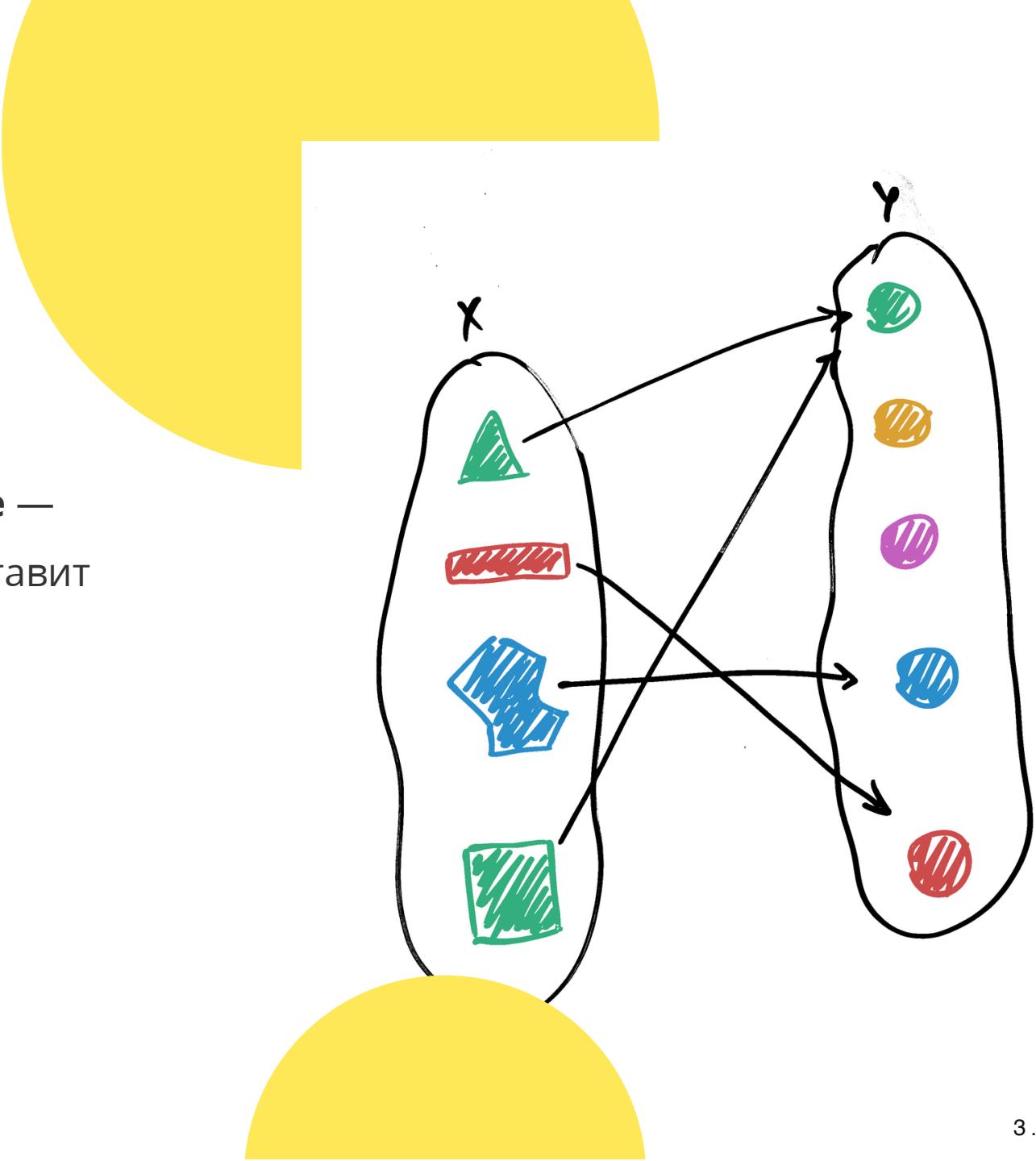


W A R N I N G

# История функций

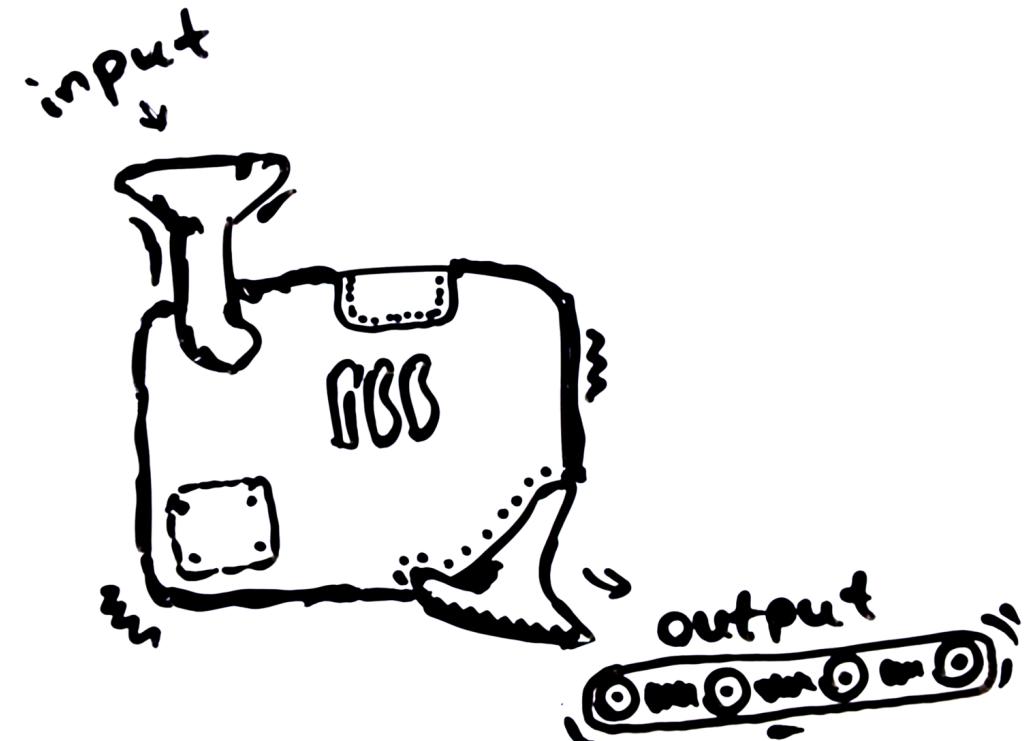
# Математические отображения

Математики придумали **отображение** — абстракцию, которая каждому числу ставит в соответствие другое число



# Математические функции

Математики придумали функции — абстракции, которые что-то получают на вход, и что-то выдают на выход



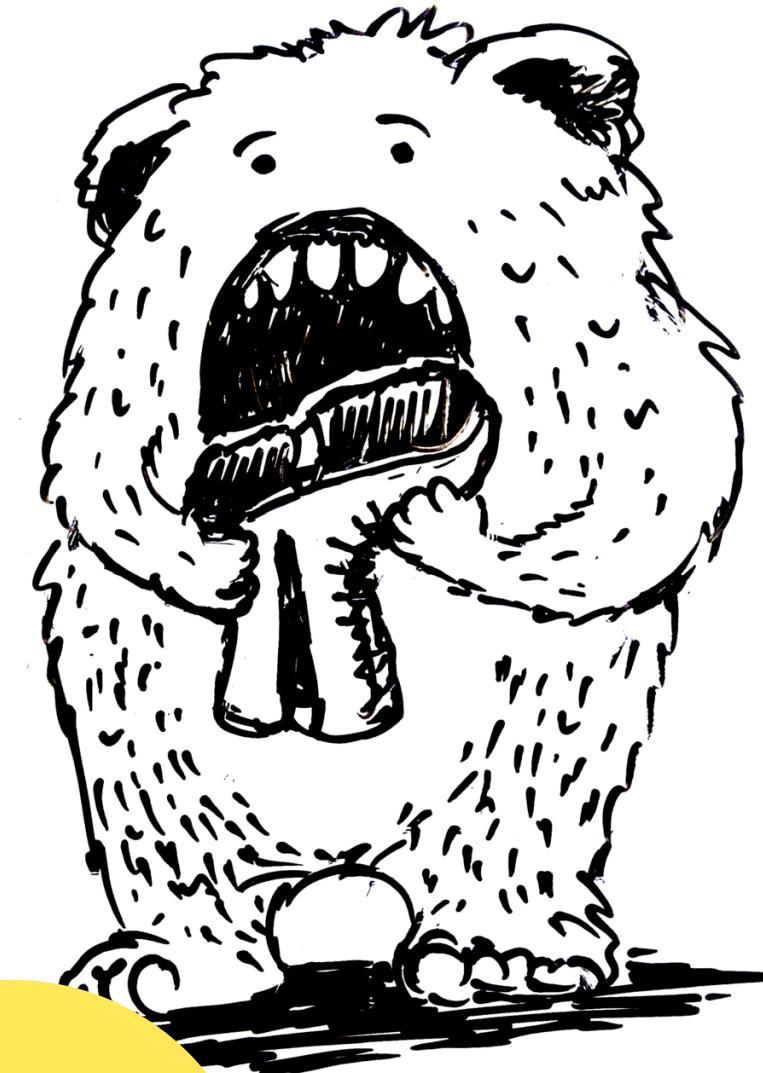
# Свойства функции



# Полная определенность

**Что не скорми — всё прожует**

На самом деле секрет в том, чтобы не кормить лишним



# Детерминированность

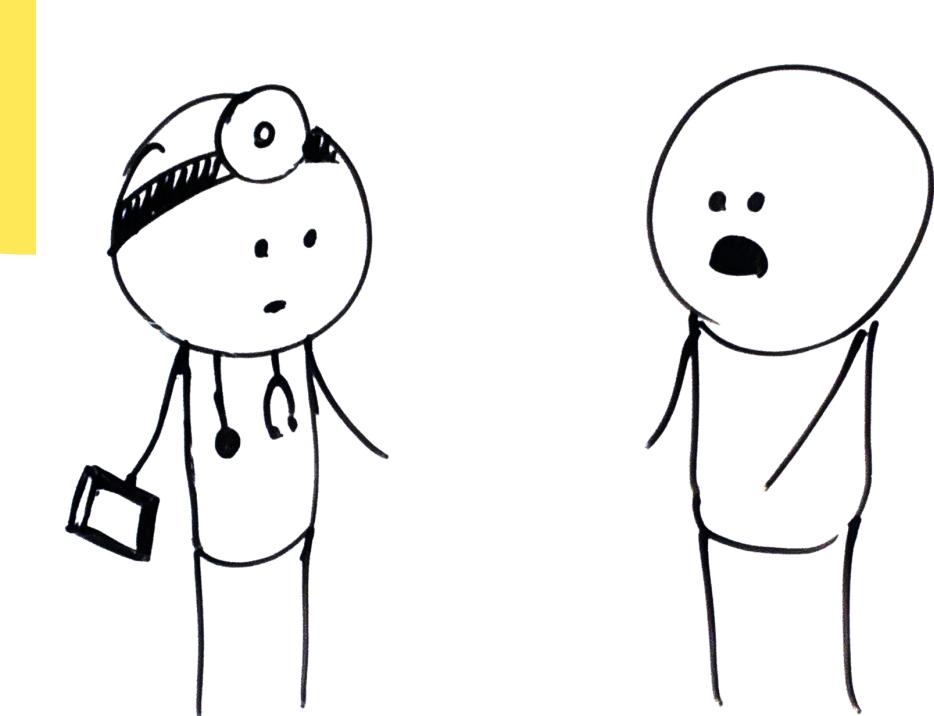
**Всегда** возвращает одинаковый результат  
на одинаковые значения аргументов

Например, пусть и неявно, функция  
random в Python детерминирована

# Отсутствие побочных эффектов

Не лезет куда не надо

Математики такого явно не говорили, но мы такое придумали: взять что-то из аргументов, сходить в API, мутировать аргумент

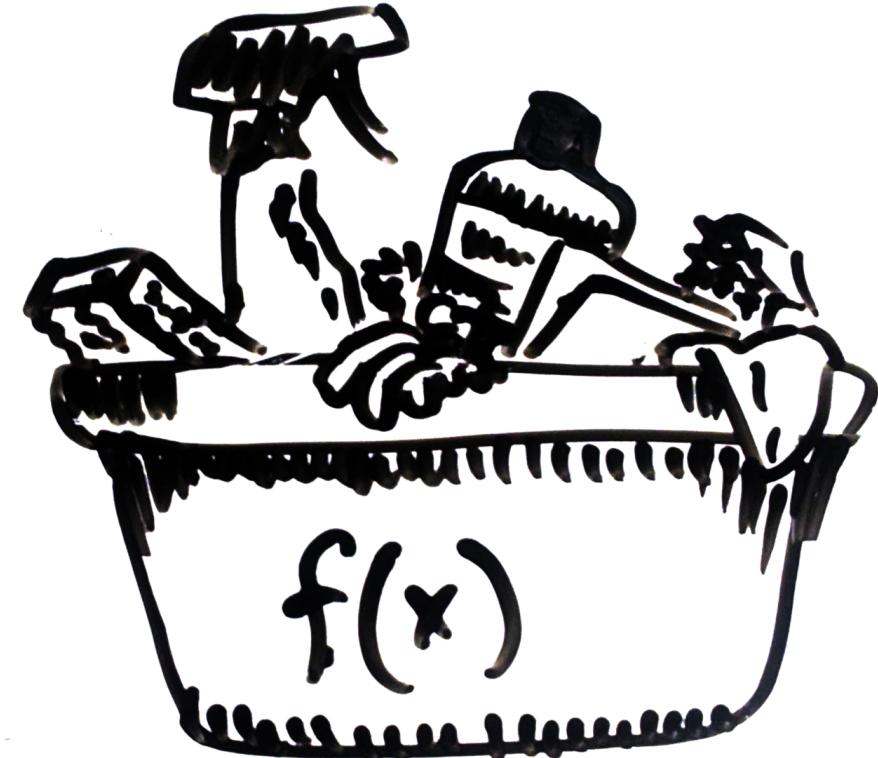


# История чистых функций

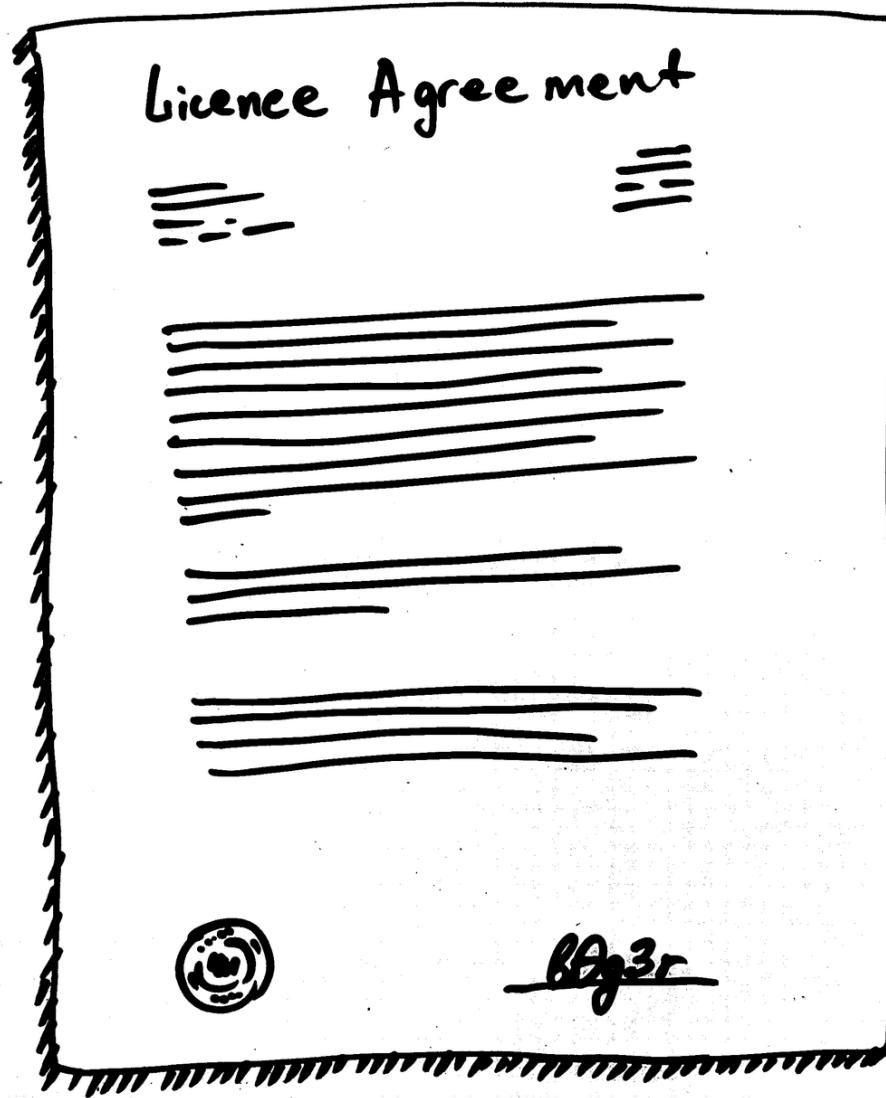
(у программистов)

# Идеальная чистая функция

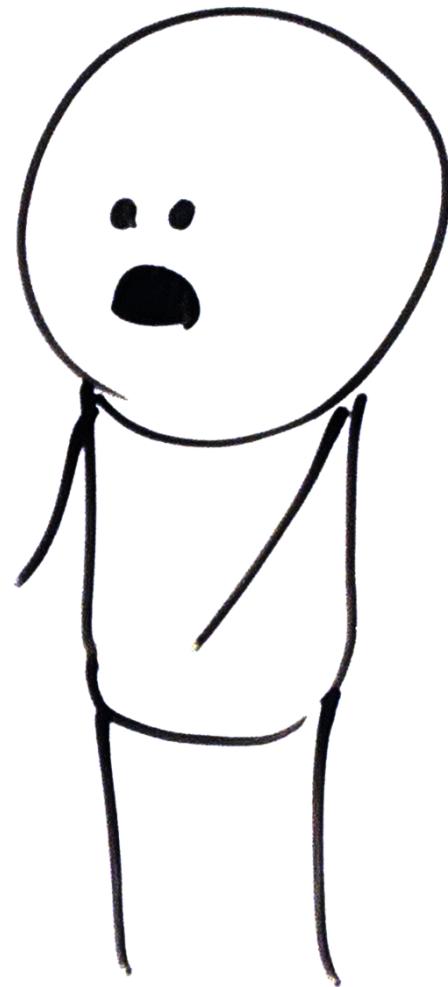
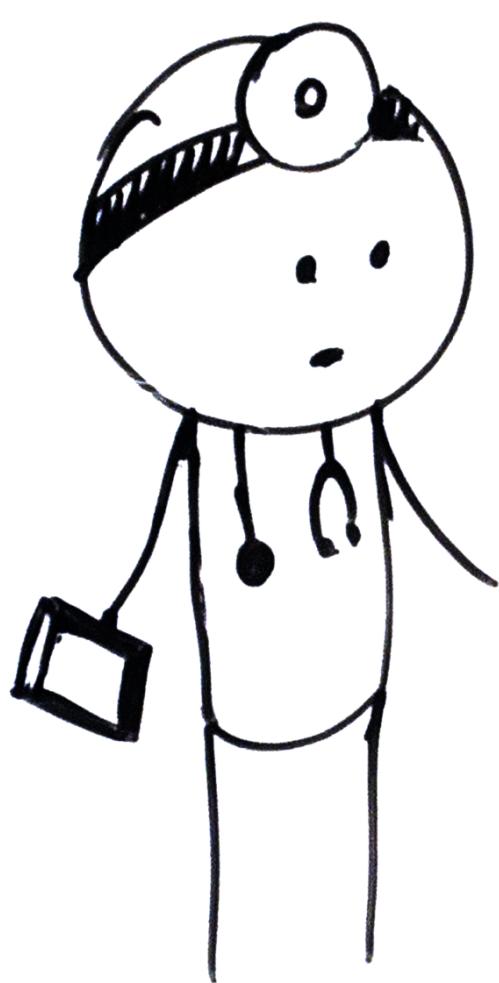
- работает для любых входных аргументов
- возвращает одинаковый результат на одинаковые вызовы
- не лезет наружу и не мутит (отсутствие побочных эффектов)



# Что такое сайд-эффекты?



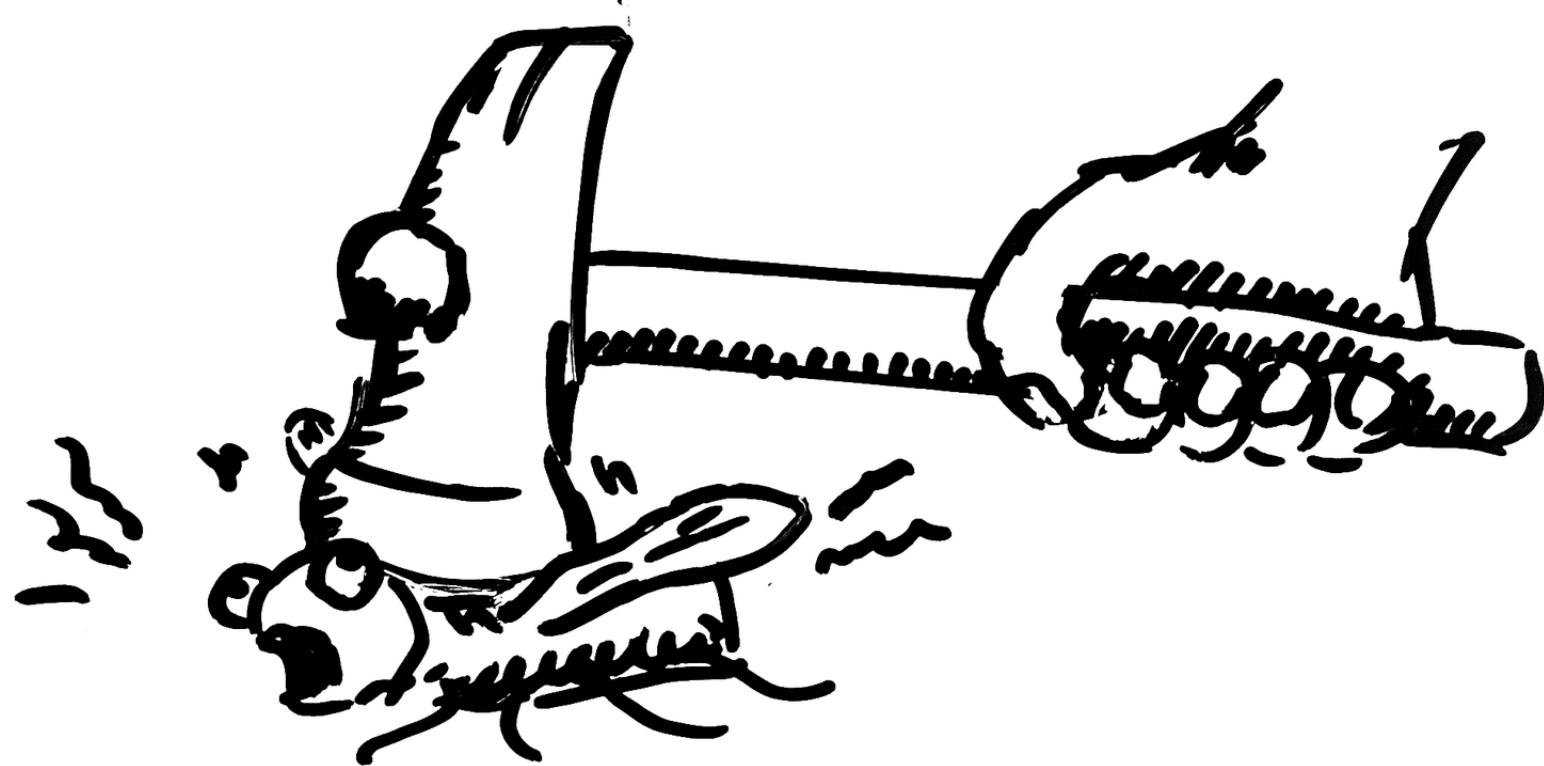
# Так зачем нужны чистые функции?



# ИНТЕРАКТИВНОСТЬ



# ОТЛАДКА



# ТЕСТИРОВАНИЕ



# easy.py

```
● ● ●  
1 def divide(a: float, b: float) -> float:  
2     return a/b  
3  
4 assert divide(6.0, 2.0) == 3.0  
5 assert divide(1.0, 0.0) exception raised
```

# easy.py

```
● ● ●  
1 def divide(a: float, b: float) -> float:  
2     return a/b  
3  
4 assert divide(6.0, 2.0) == 3.0  
5 assert divide(1.0, 0.0) exception raised
```

# easy.py

```
● ● ●  
1 def divide(a: float, b: float) -> float:  
2     return a/b  
3  
4 assert divide(6.0, 2.0) == 3.0  
5 assert divide(1.0, 0.0) exception raised
```

# easy.py

```
● ● ●  
1 def divide(a: float, b: float) -> float:  
2     return a/b  
3  
4 assert divide(6.0, 2.0) == 3.0  
5 assert divide(1.0, 0.0) exception raised
```

# hard.py

```
● ● ●  
1 import coeff  
2 import external_api  
3  
4 def divide(a: float, b: float) -> float:  
5     return external_api.call(a, b, coeff)  
6  
7 assert divide(6.0, 2.0) == ???  
8 assert divide(1.0, 0.0) ???
```

# hard.py

```
● ● ●  
1 import coeff  
2 import external_api  
3  
4 def divide(a: float, b: float) -> float:  
5     return external_api.call(a, b, coeff)  
6  
7 assert divide(6.0, 2.0) == ???  
8 assert divide(1.0, 0.0) ???
```

# hard.py

```
● ● ●  
1 import coeff  
2 import external_api  
3  
4 def divide(a: float, b: float) -> float:  
5     return external_api.call(a, b, coeff)  
6  
7 assert divide(6.0, 2.0) == ???  
8 assert divide(1.0, 0.0) ???
```

# hard.py

```
● ● ●  
1 import coeff  
2 import external_api  
3  
4 def divide(a: float, b: float) -> float:  
5     return external_api.call(a, b, coeff)  
6  
7 assert divide(6.0, 2.0) == ???  
8 assert divide(1.0, 0.0) ???
```

# Почему нельзя писать только чистые функции?

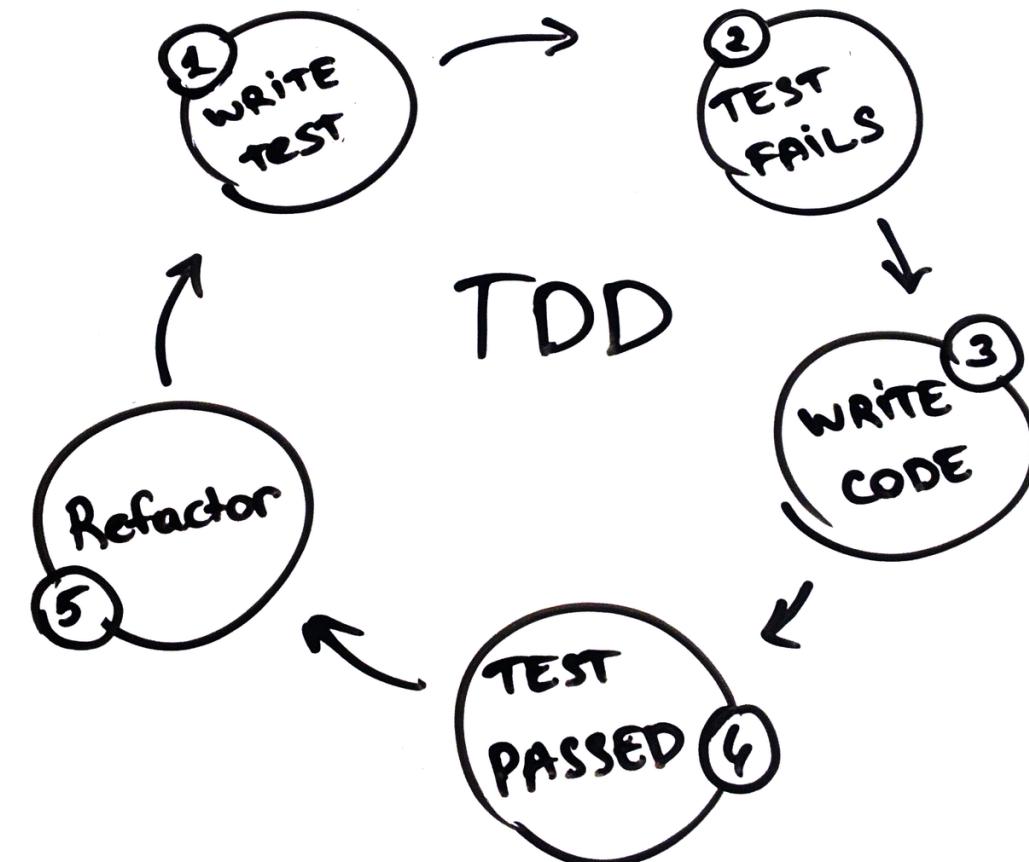
Постоянно нужно делать грязные вещи



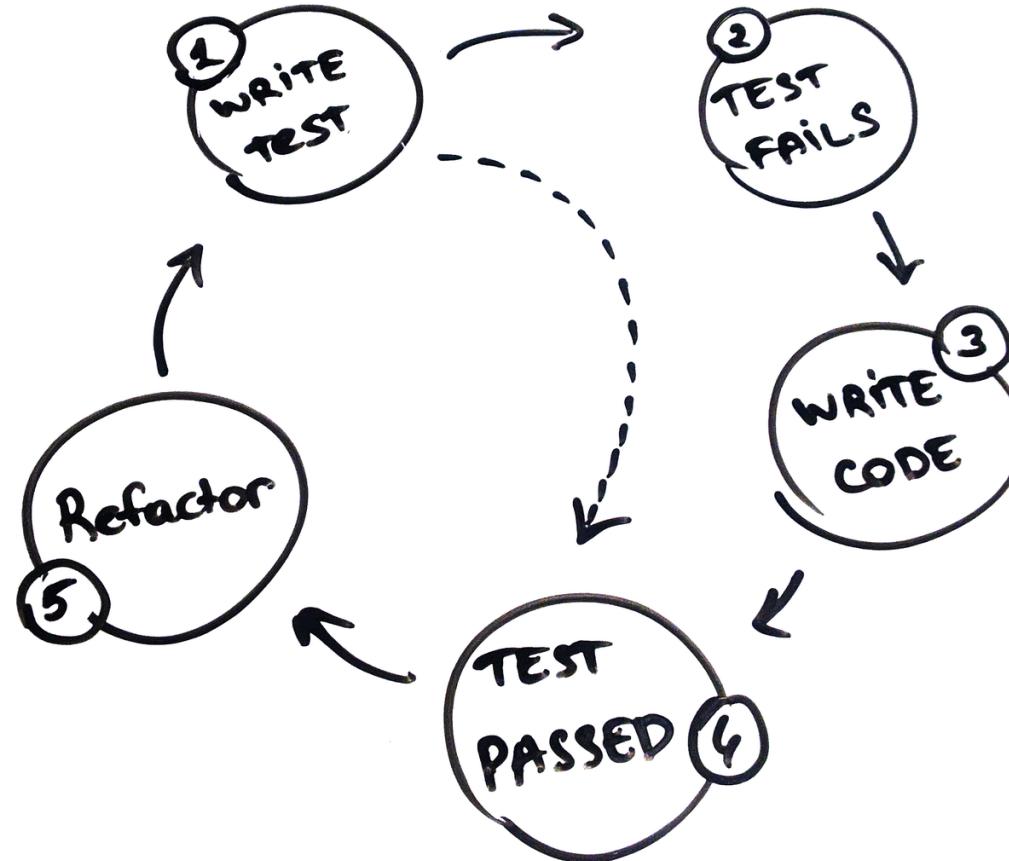
# **Test Driven Development**

(кто-то вообще этим пользуется?)

# TDD



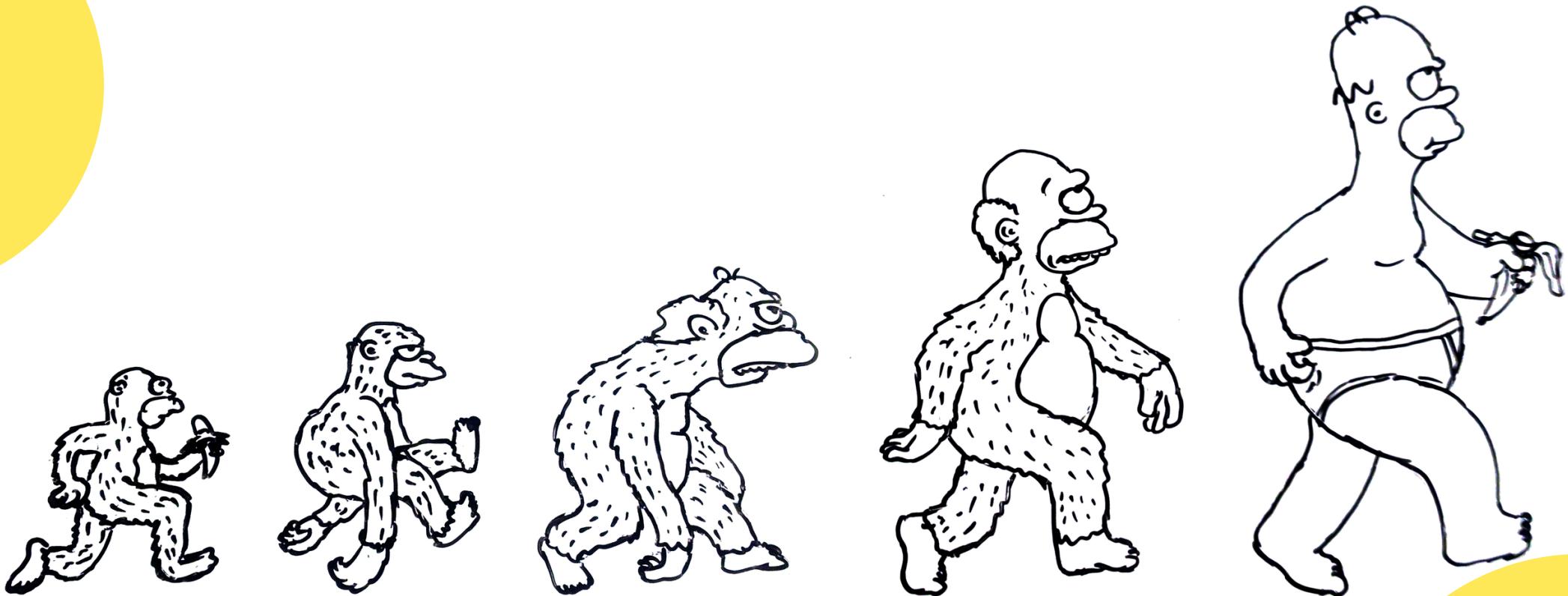
## А иногда и так



# **История компонентного подхода**

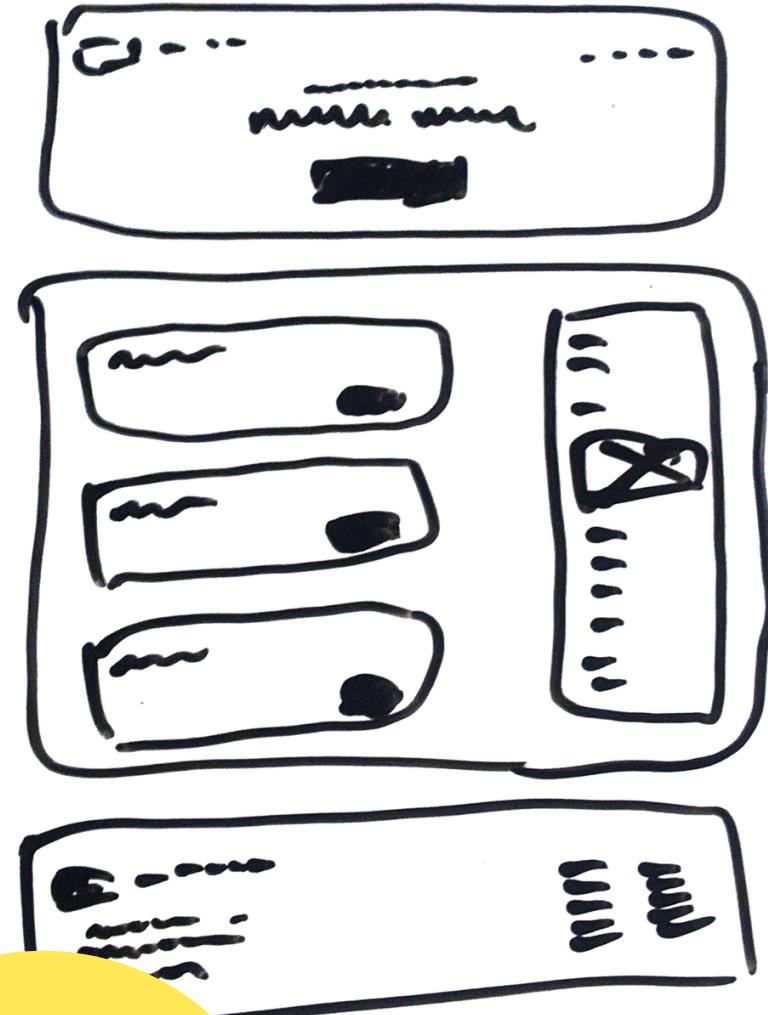
**(у строителей пользовательских интерфейсов)**

# Эволюция



## Блоки вёрстки

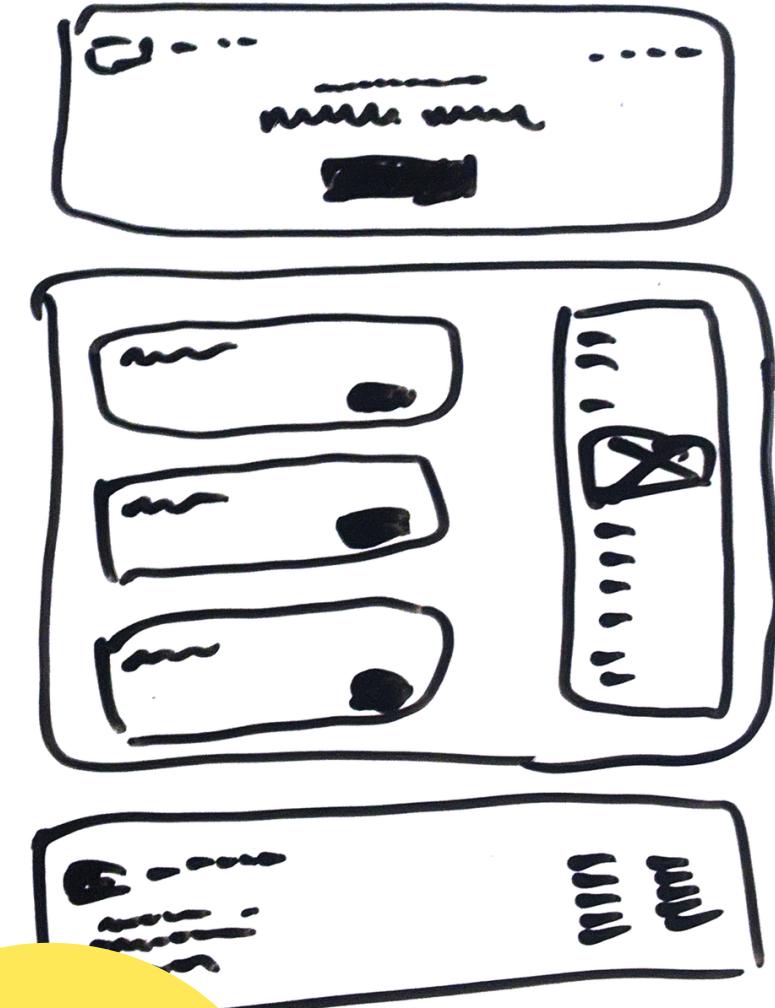
Сначала мы писали страницу модулями — блоками вёрстки, которую кусочно генерировали на сервере



# Переиспользование

Потом мы поняли, что блоки одни и те же,  
и вынесли их в компоненты

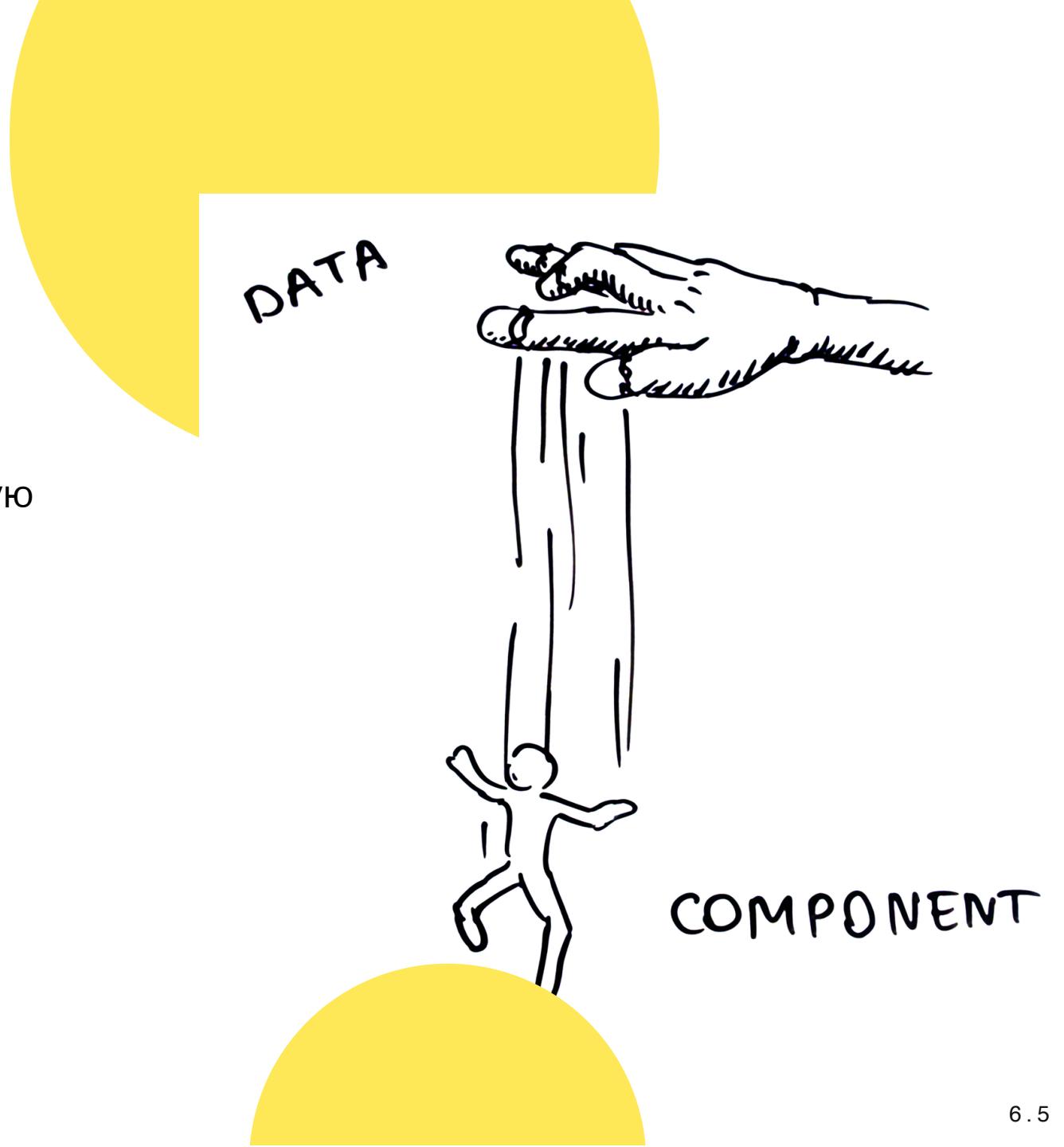
Которые зависели **только** от данных



## Данные для блоков

Если такие блоки зависят только от данных, то можно придумывать разную магию:

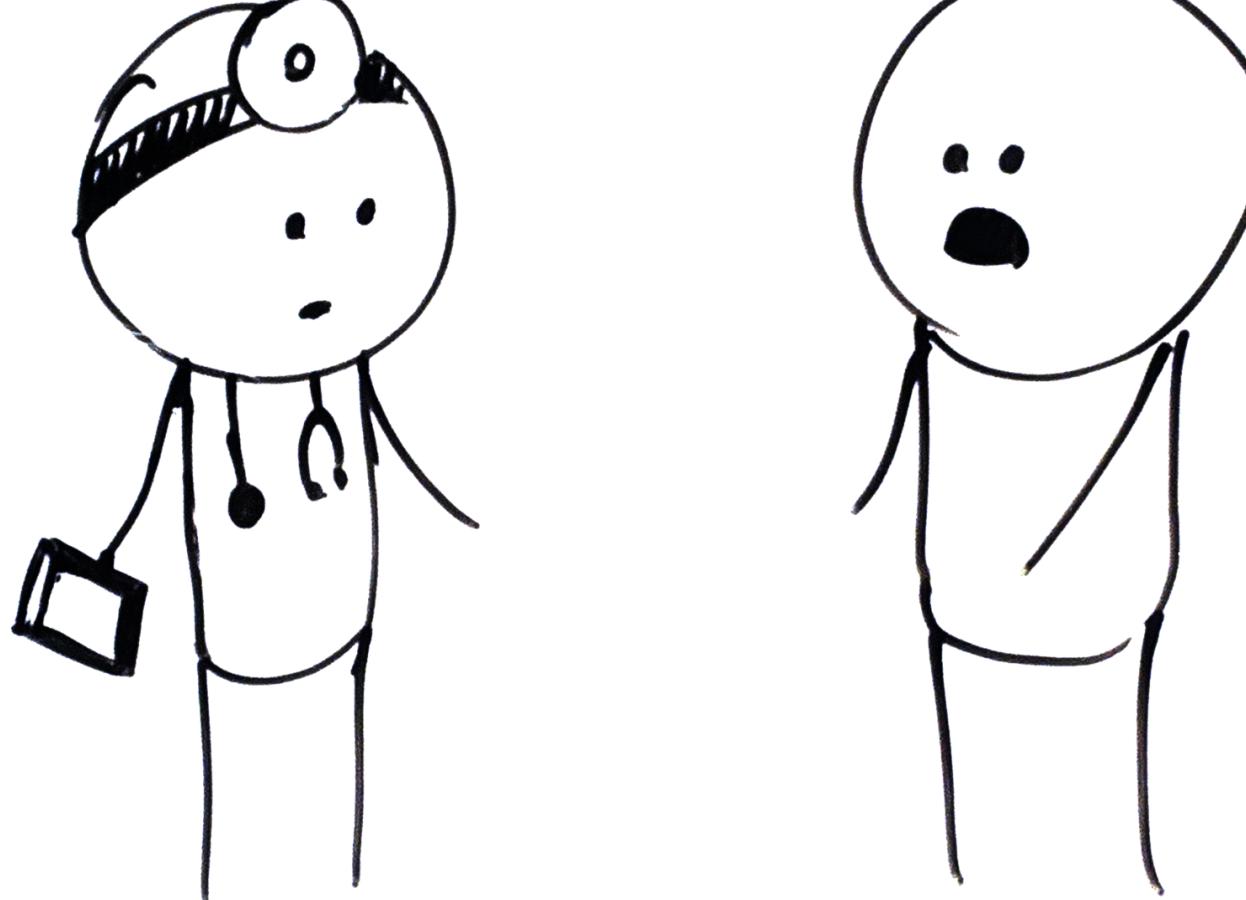
- способы передачи этих данных;
- хранилища;
- способы синхронизации;



# **А в чём проблема?**

**(данные есть компоненты есть)**

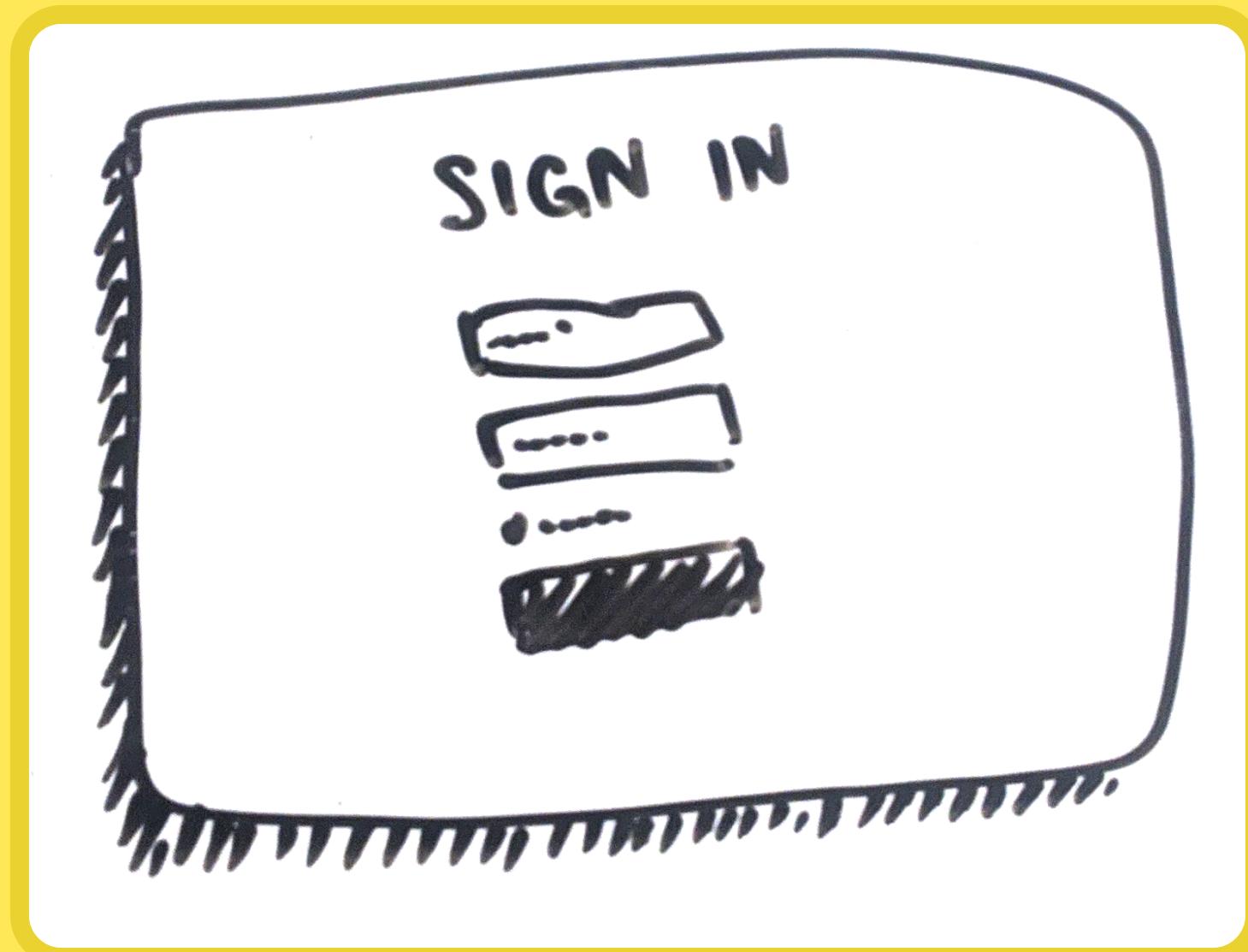
# Побочные эффекты!



# **Пример**

**(входим и выходим)**

# Как быть?



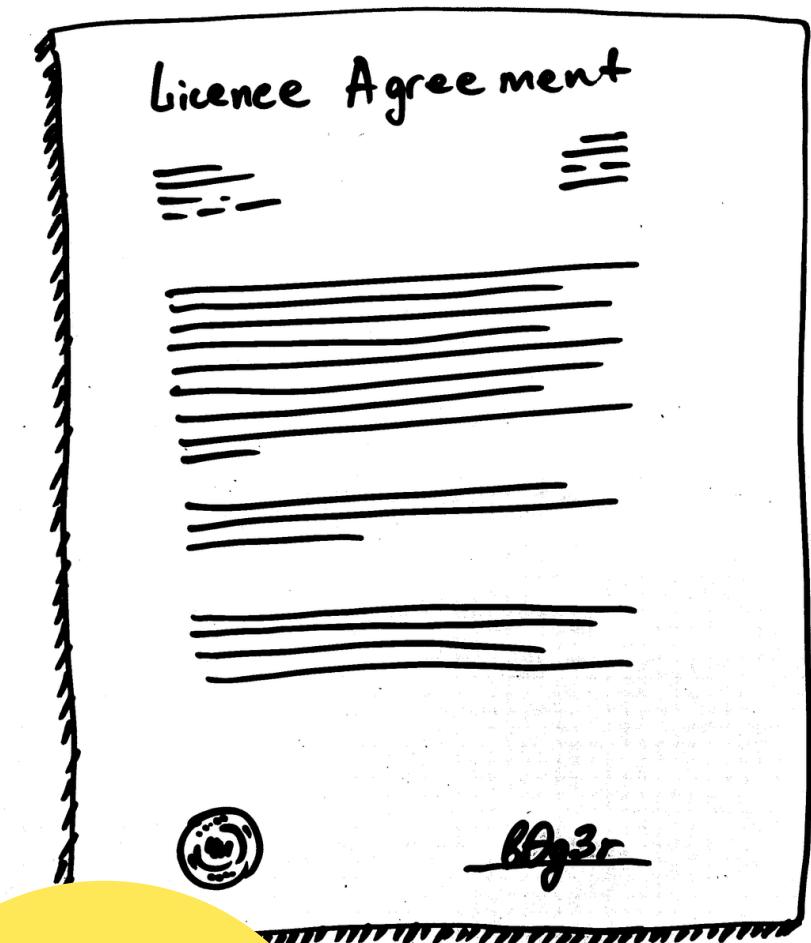
# **Глупые и умные компоненты**

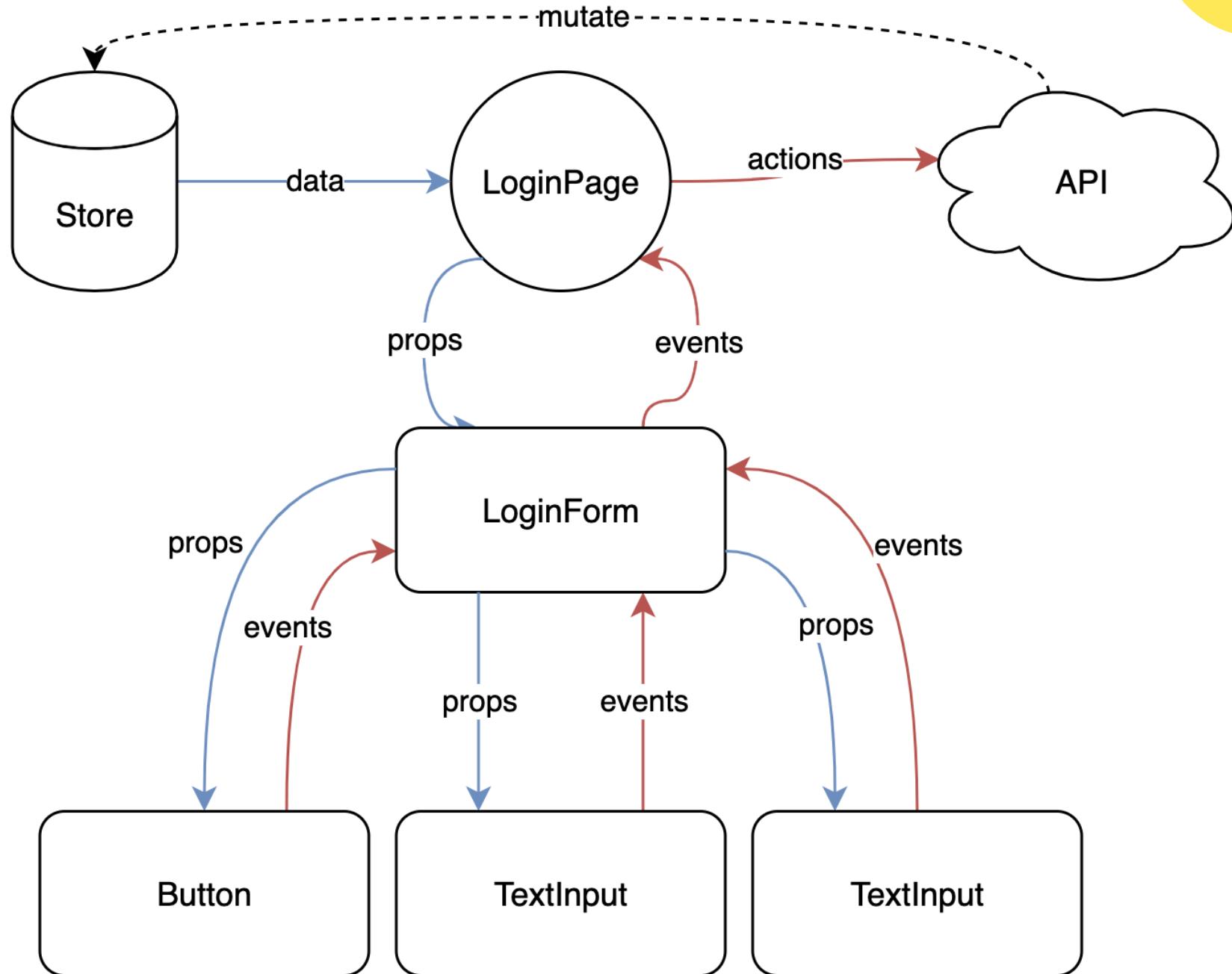
**(представительные и контейнеры)**

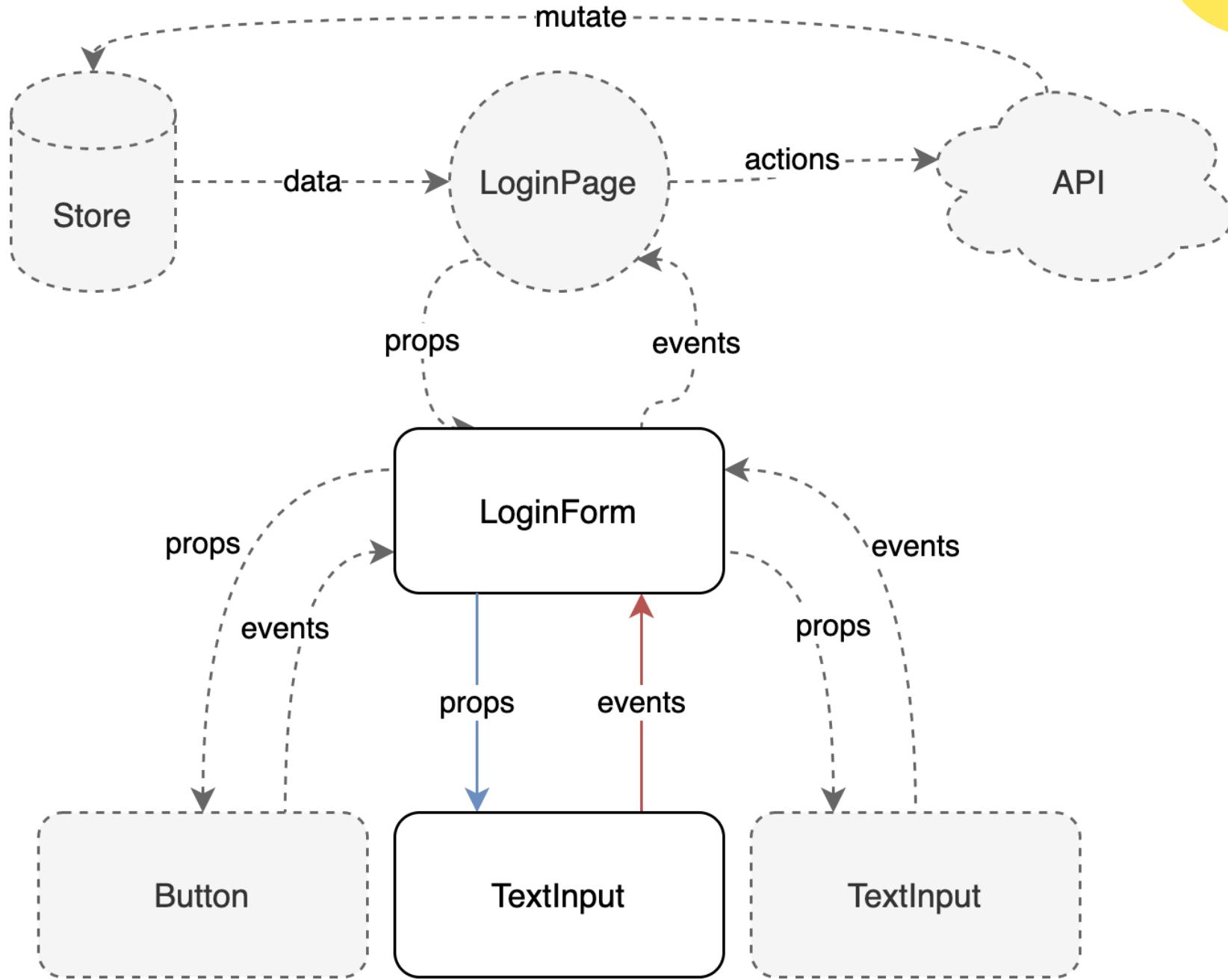
# Разделяй и властвуй

Глупые — не делают сайд-эффектов

Умные — делают сайд-эффекты (API,  
хранилище, муттирование)



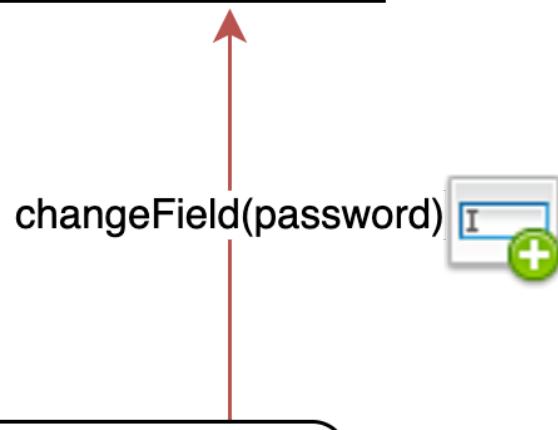




LoginForm
username
password

LoginForm
username
password = new_password

LoginForm
username
password

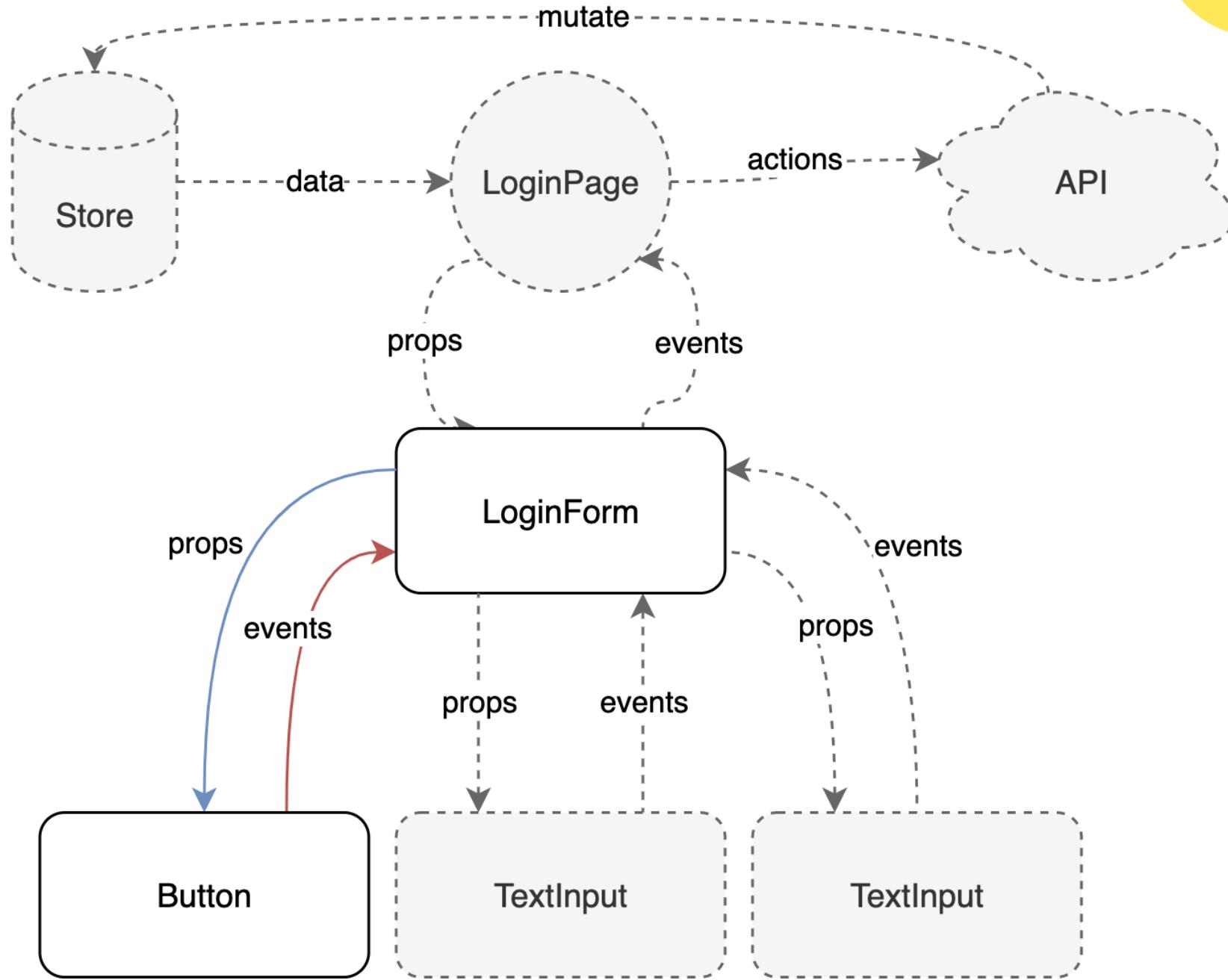


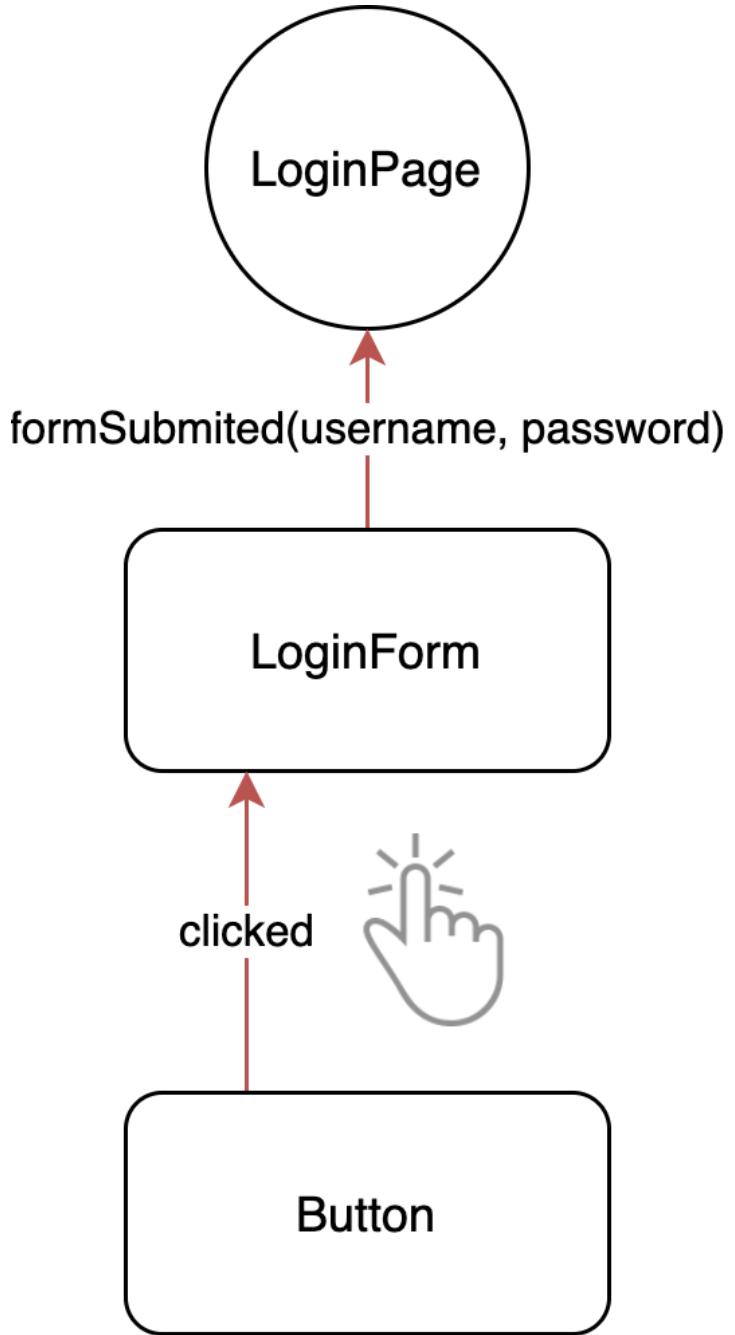
TextInput

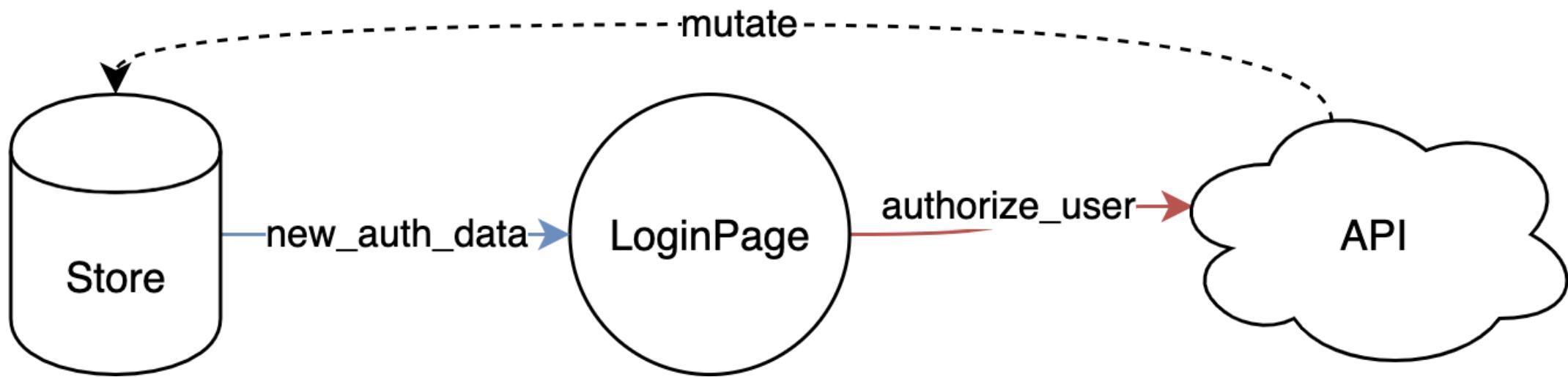
TextInput

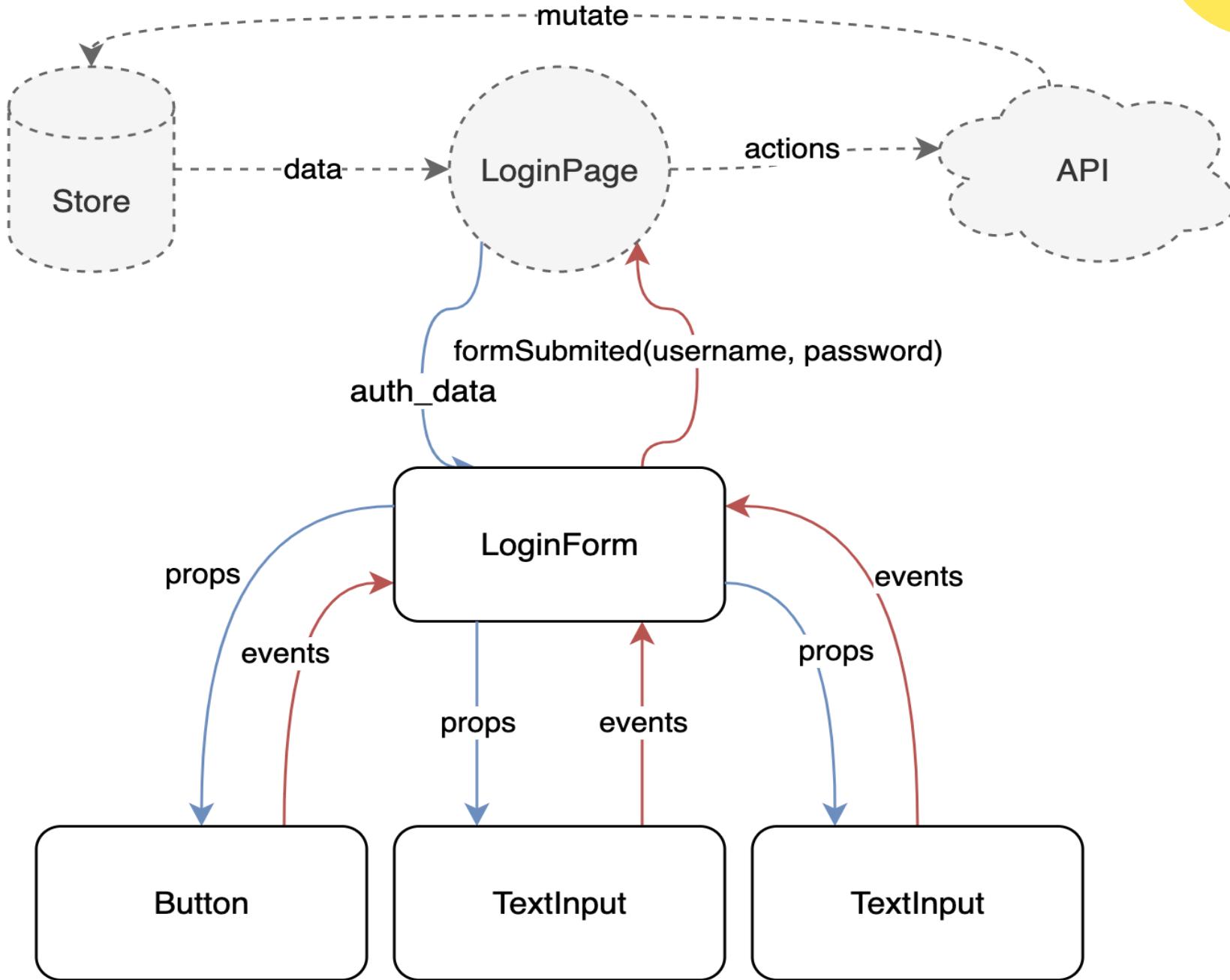
TextInput

new\_props





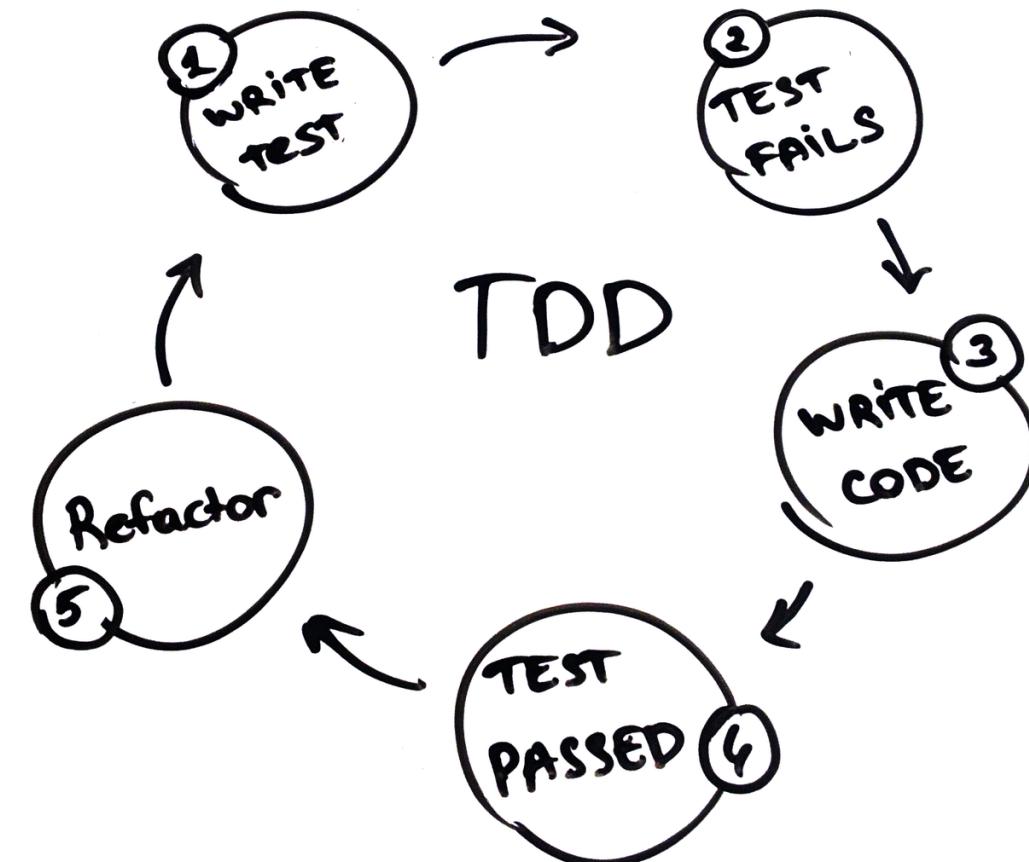




# **Visual TDD и CDD**

**визуальное тестирование и разработка через компоненты**

# TDD



# Visual TDD и связь с заказчиком

## CommentList

## HasData

Avatar



Luke Lorem ipsum dolor sit amet,  
consectetur adipiscing elit.

Author



Leah Donec erat ante, auctor ut lacus  
vel, ullamcorper viverra urna.

Body



Han Class aptent taciti sociosqu ad  
litora torquent per conubia nostra

# Задача — источник состояний

CommentList



**Luke** Lorem ipsum dolor sit amet, consectetur adipiscing elit.



**Leah** Donec erat ante, auctor ut lacus vel, ullamcorper viverra urna.



**Han** Class aptent taciti sociosqu ad litora torquent per conubia nostra

HasData

CommentList



**Luke** Lorem ipsum dolor sit amet, consectetur adipiscing elit.



**Leah** Donec erat ante, auctor ut lacus vel, ullamcorper viverra urna.



**Han** Class aptent taciti sociosqu ad litora torquent per conubia nostra



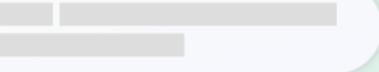
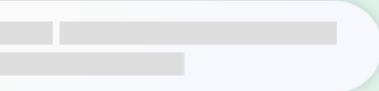
**Poe** Donec erat ante, auctor ut lacus vel, ullamcorper viverra urna.



**Finn** Class aptent taciti sociosqu ad litora torquent per conubia nostra

Load more

CommentList



Loading

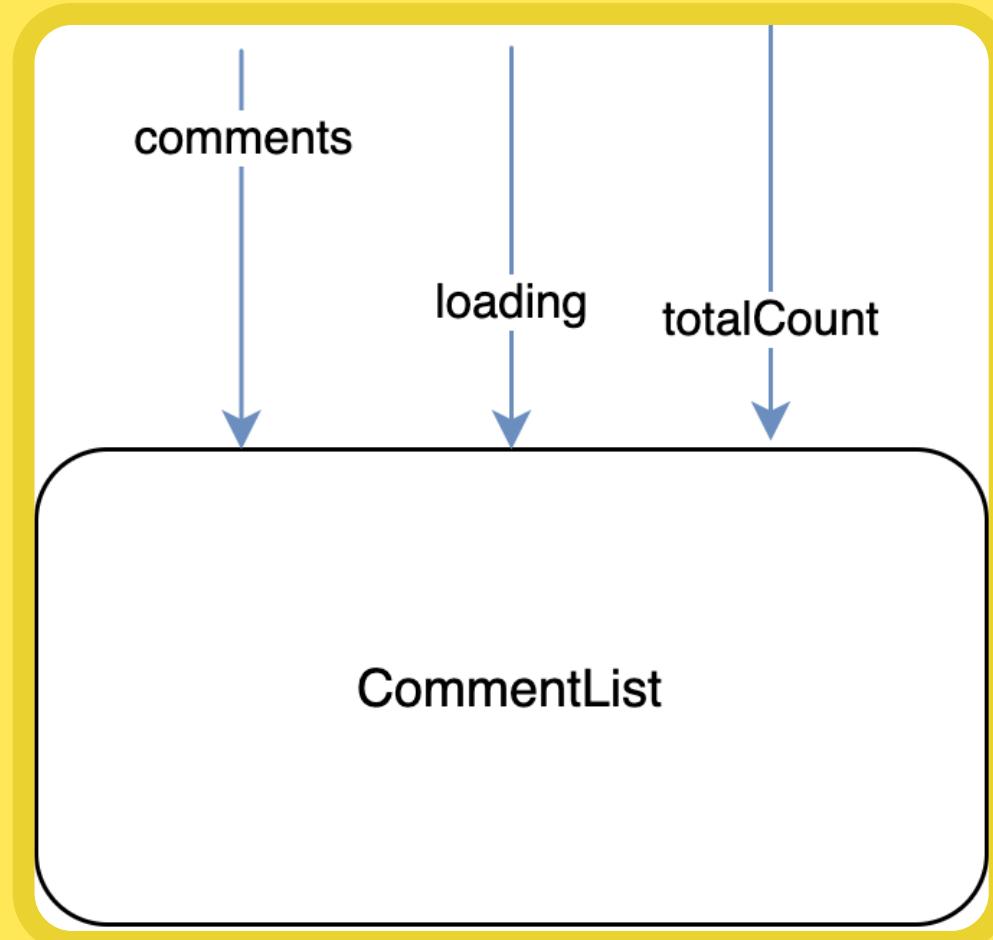
CommentList

No comments yet  
Have something to say? Say it!

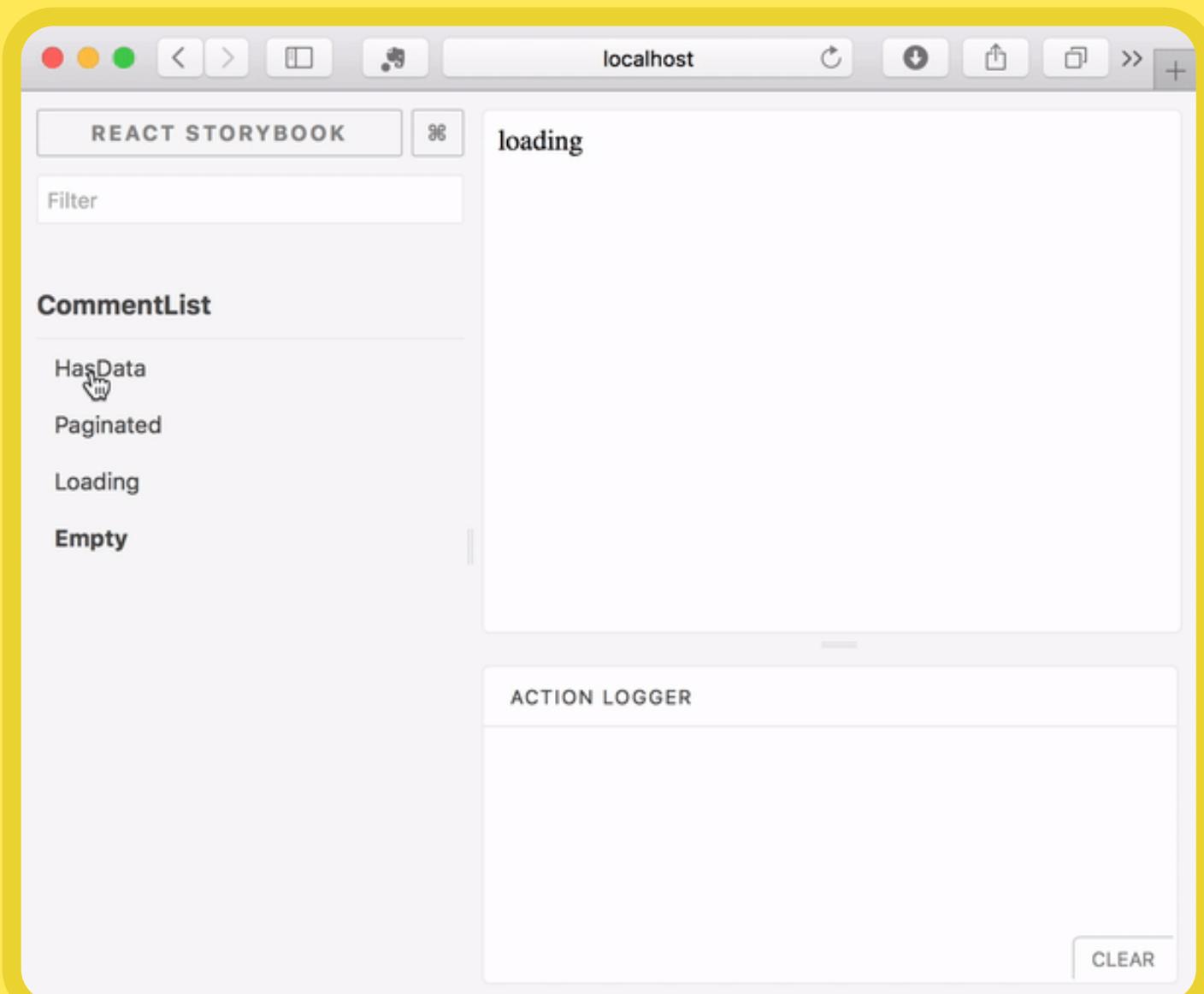
Empty



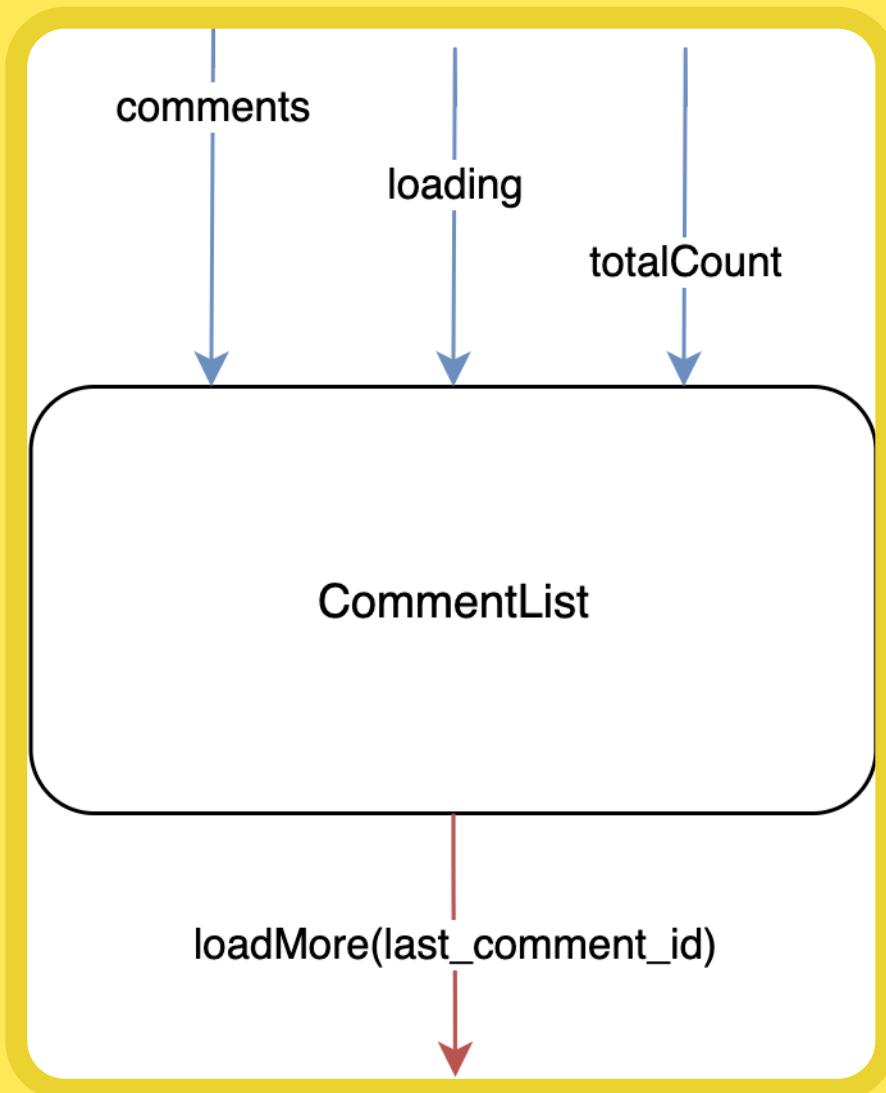
# Пишем заглушку компонента



# Пишем "визуальный тест"



## Вторая итерация



# Пишем реализацию

REACT STORYBOOK

Filter

CommentList

- HasData
- Paginated
- Loading
- Empty



**Luke** Lorem ipsum dolor sit amet, consectetur adipisicing elit.



**Leah** Ut enim ad minim veniam, quis nostrud exercitation ullamco.

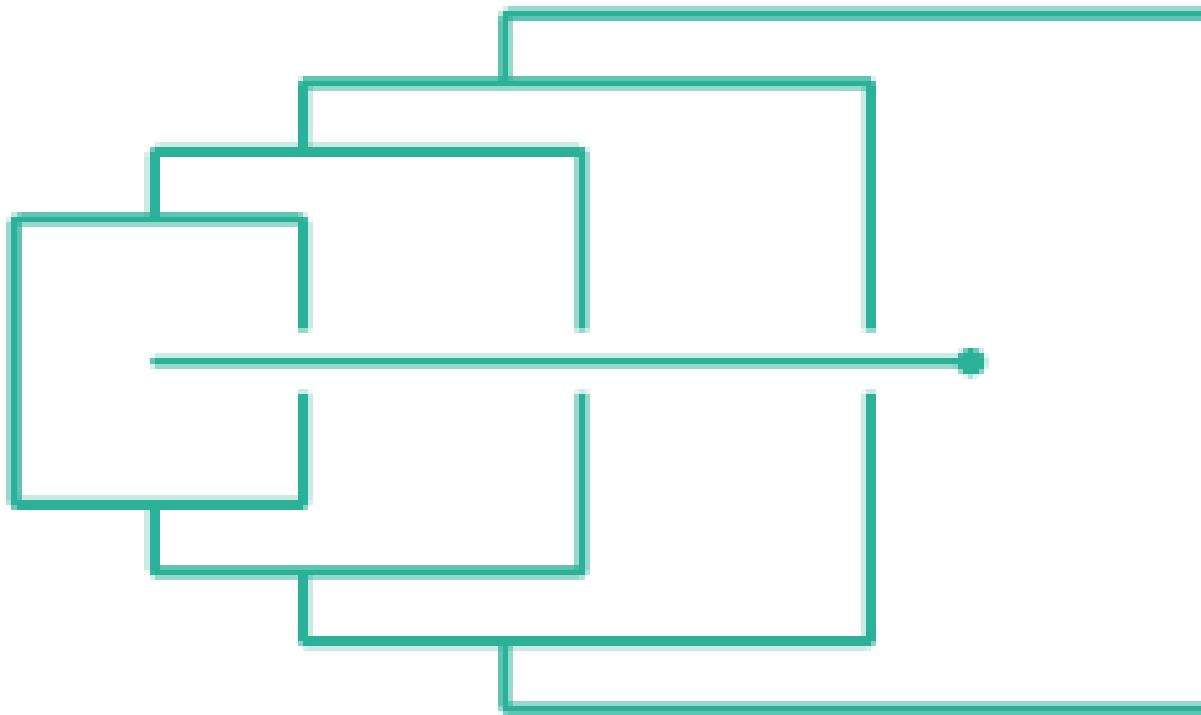


**Han** Duis aute irure dolor in reprehenderit in voluptate.

ACTION LOGGER

CLEAR

# Повторяем



# **Storybook**

**и другие component workshop**

# Storybook

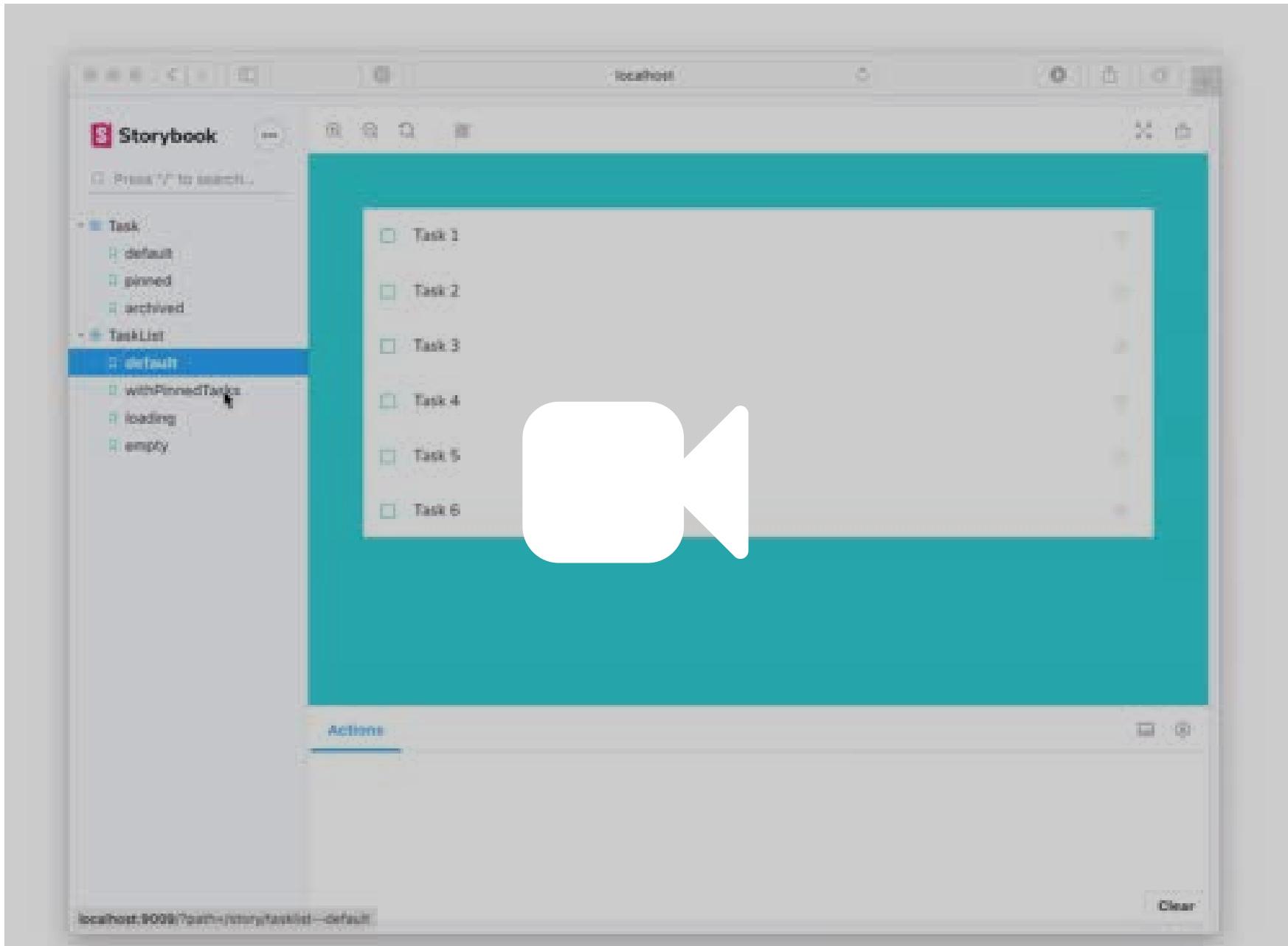
- У каждого компонента есть состояния
- На каждое состояние — историю
- Истории — по папкам
- Плагин-система для логирования
- Плагин-система для песочницы



# Как это пишется

```
1 export const Default = () =>
2     <TaskList tasks={defaultTasksData} {...actionsData} />;
3
4 export const WithPinnedTasks = () =>
5     <TaskList tasks={withPinnedTasksData} {...actionsData} />;
6
7 export const Loading = () =>
8     <TaskList loading tasks={[ ]} {...actionsData} />;
9
10 export const Empty = () =>
11     <TaskList tasks={[ ]} {...actionsData} />;
```

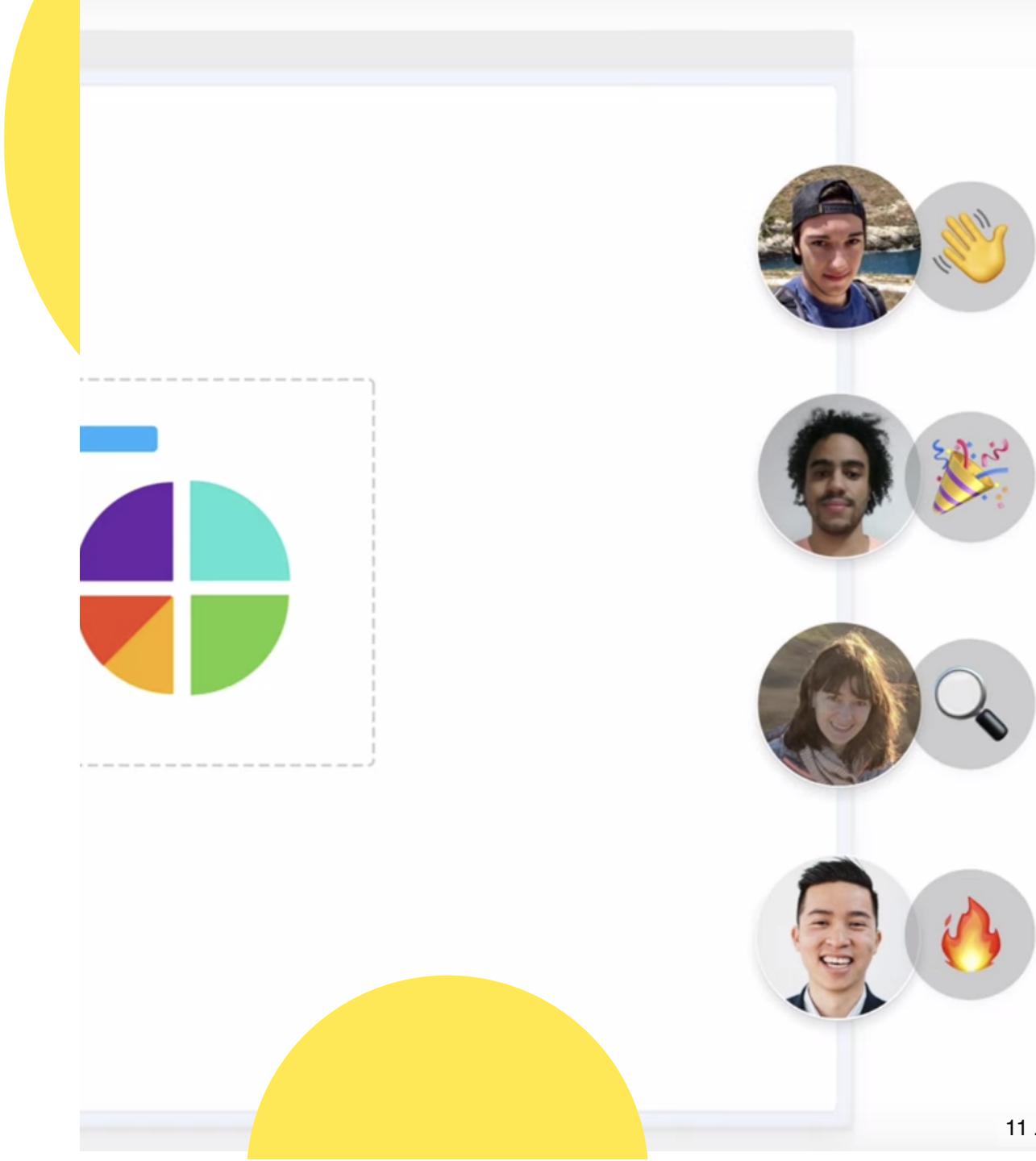
# Как это выглядит



# Плюшки

# Теперь у вас есть дизайн-система!

У всех разработчиков, менеджеров и дизайнеров есть реальный интерактивный набор компонентов



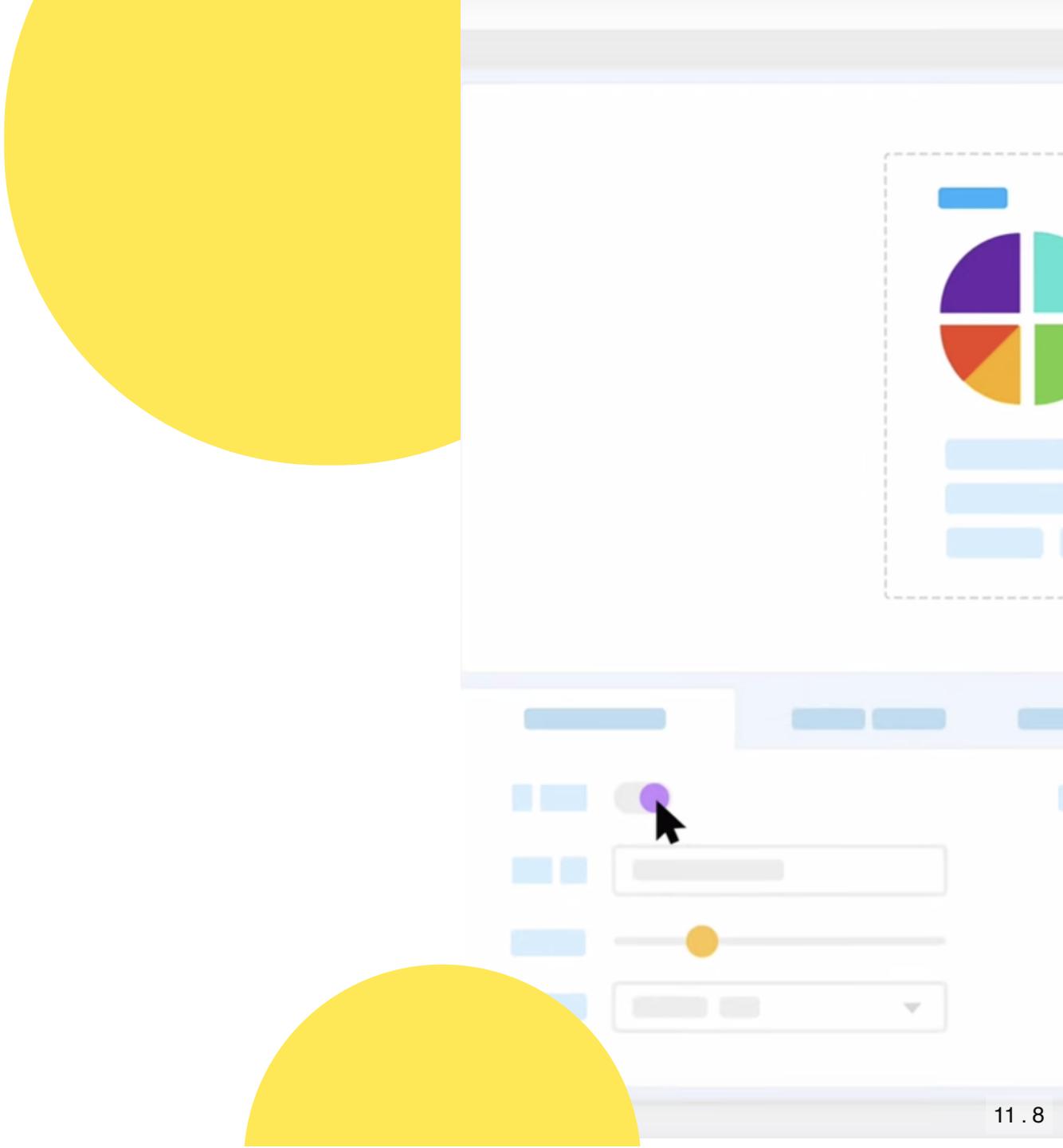
## Границные случаи

Историями легко показать как компонент ведёт себя в граничных случаях, как выводит ошибки и как ведёт себя на тёмном фоне



# Песочница

Можно дёргать ручки и смотреть что поменяется.



# Снепшот-тесты

Можно фиксировать поведение ваших компонентов между релизами (см chromaticqa)

chromaticqa

Build 9

Initiated 4 hours ago by Tom Coleman • 81e4b4c to master

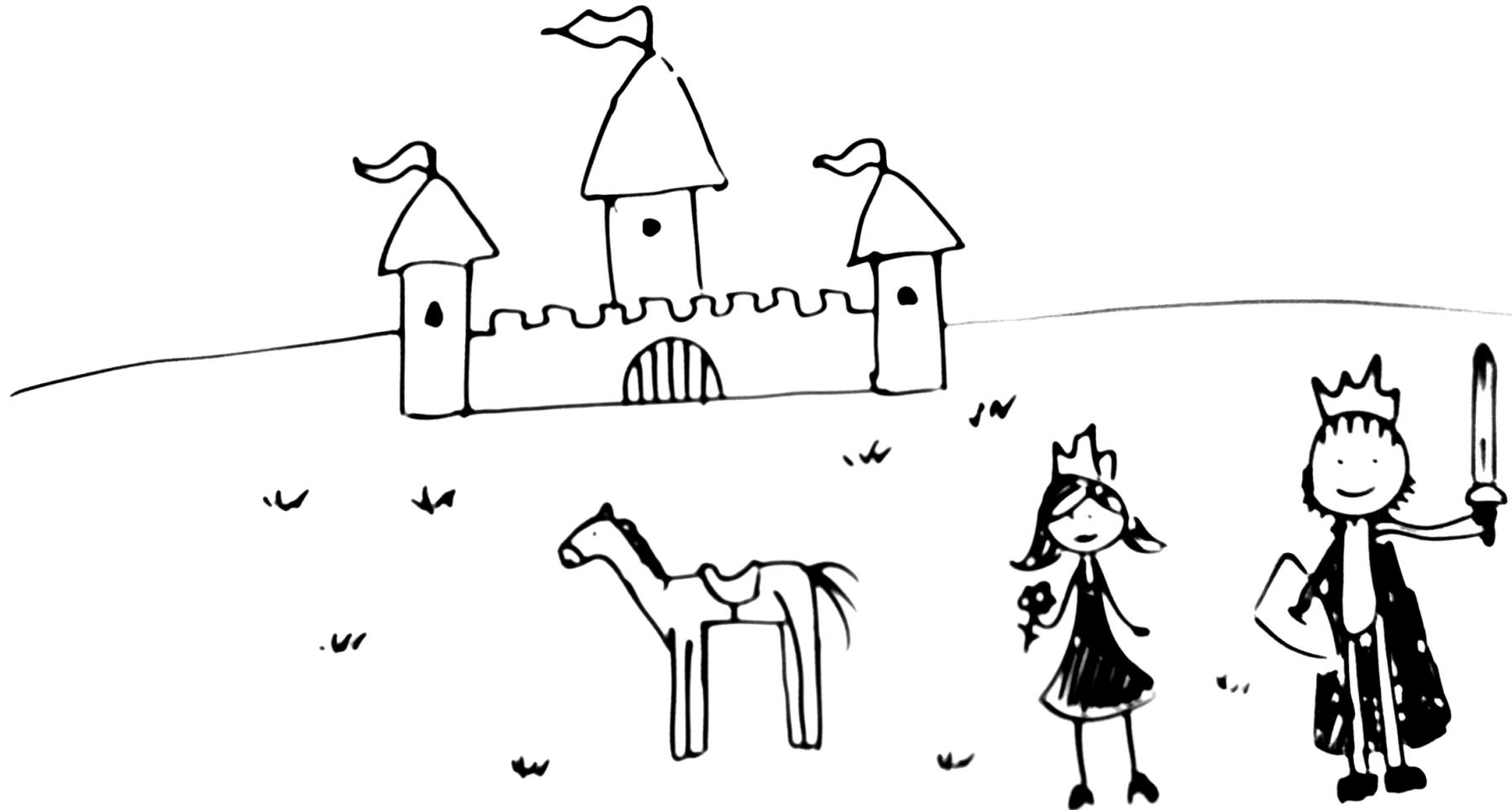
SUMMARY

33	46s	4	0	0
Unchanged	Build time	Changes	Accepted	Denied
33	46s	4	0	0
Unchanged	Build time	Changes	Accepted	Denied

Changes	Status
Default	Unreviewed
Low data	Unreviewed
DatePicker	Unreviewed
	Unreviewed

**ypa!**

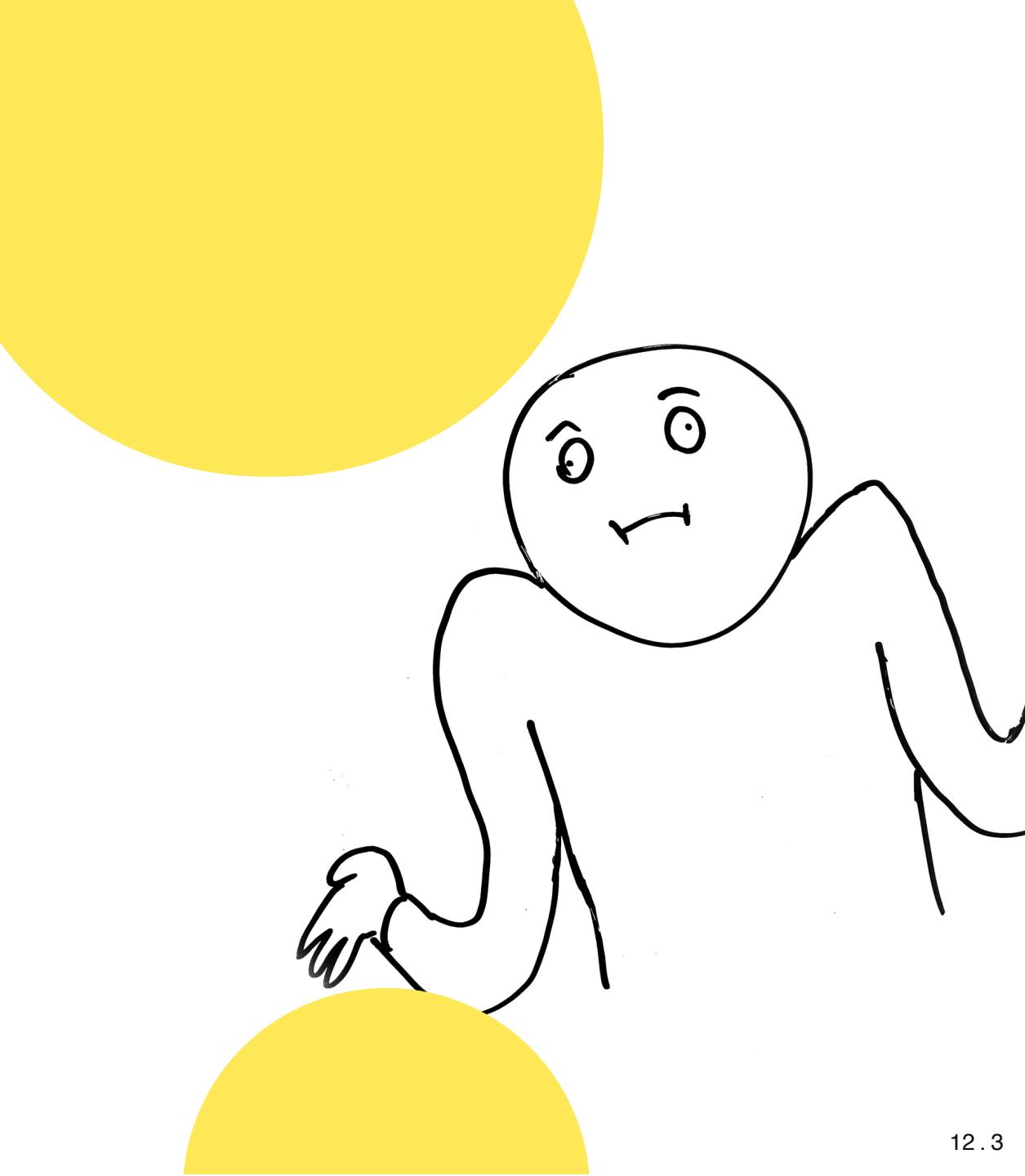
# Красивая сказка



## Поддержка языка

Ни язык, ни фреймворк не поддерживает  
деление на "чистые" и на "грязные"  
функции/компоненты

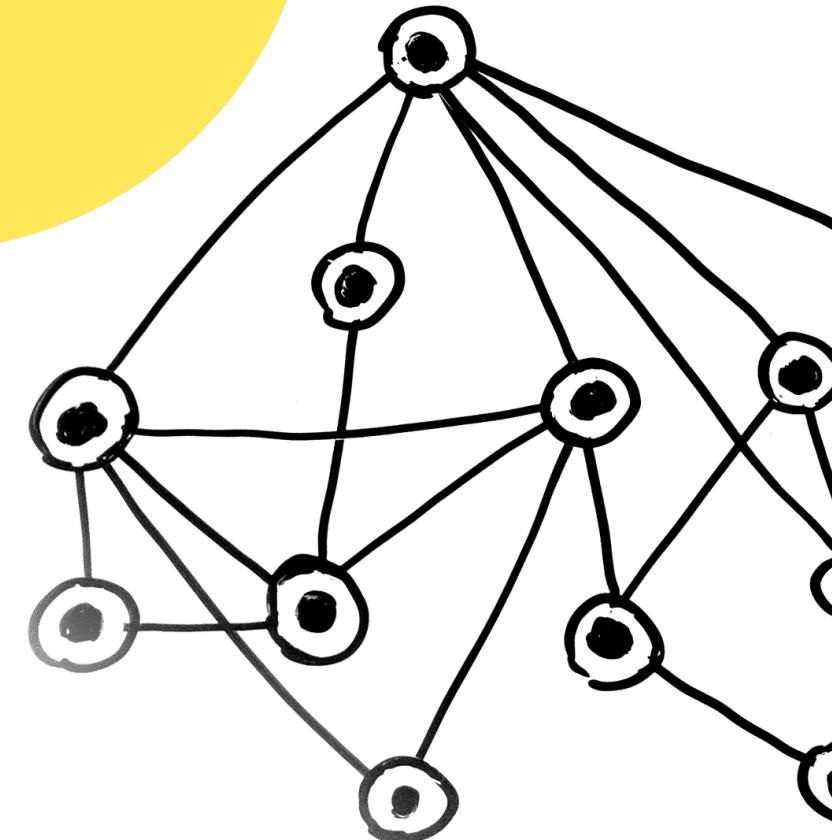
Нет никакой возможности не отстрелить  
себе случайно ногу



# **Реальная жизнь сложнее**

Не всегда получится написать чистый компонент, особенно при большом зацеплении компонентов.

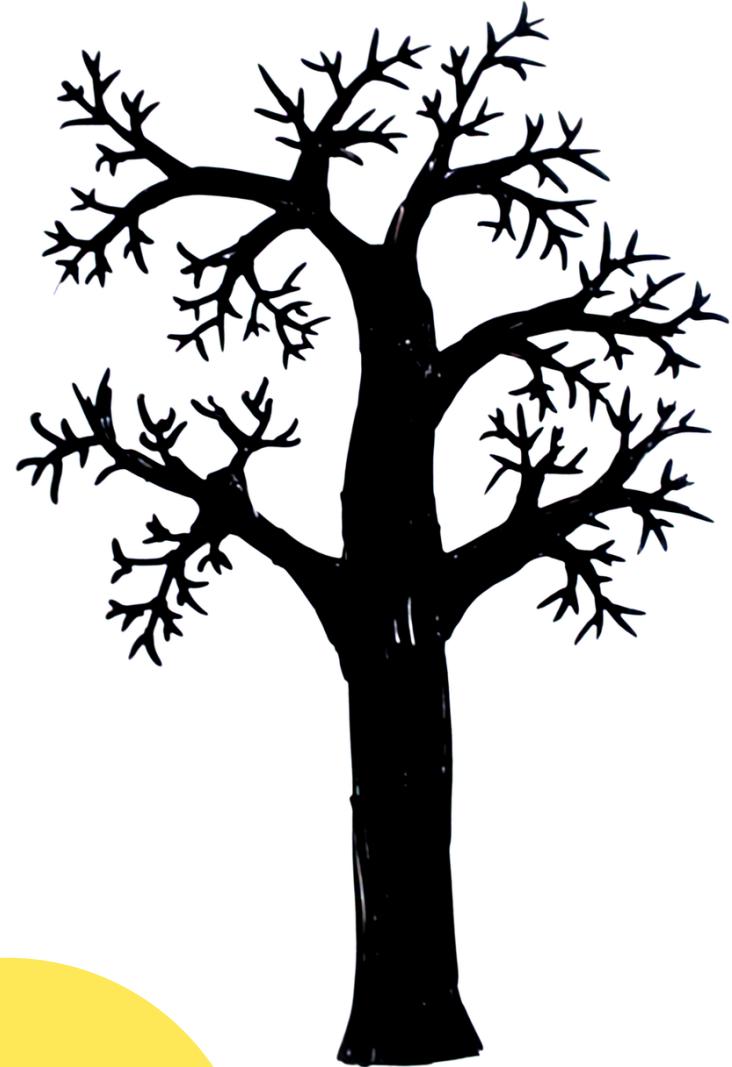
Иногда нужно императивно взаимодействовать с браузером или получать одну маленькую переменную из общего стора.



## **Большие деревья и прослойки**

Иногда событие нажатия на кнопку  
придётся прокидывать через много  
уровней.

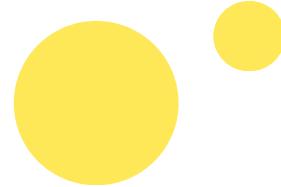
Не всегда понятно где делать прослойки  
из умных компонентов.



# **Изучайте новое**

**не будьте занудами**

# Вопросы?



- Меня зовут Дима
- Я рассказывал про
  - свойства математических функций
  - свойства чистых функций
  - TDD
  - глупые и умные компоненты
  - Visual TDD
  - Storybook
  - Проблемы подхода



# **Использованные материалы**

# Использованные материалы

1. [ruhaskell.org] Эффекты и чистые функции
2. [draw.io] Копия диаграм со страницы с компонентами
3. [hexlet.io] Урок по чистым функциям
4. [wikipedia.org] Чистота функции
5. [wikipedia.org] Ссылочная прозрачность
6. [wikipedia.org] Разработка через тестирование
7. [ippon.tech] TDD Adoption

# Дополнительные материалы

- [\[learnstorybook.org\]](https://learnstorybook.org) How to Storybook!
- [\[chromaticqa.com\]](https://chromaticqa.com) Chromatic QA
- [\[medium.com\]](https://medium.com/storybookjs) Visual TDD
- [\[medium.com\]](https://medium.com/storybookjs) Visual Testing
- [\[youtube.com\]](https://www.youtube.com/watch?v=KJLjyfXWzqU) Реклама Storybook
- [\[youtube.com\]](https://www.youtube.com/watch?v=KJLjyfXWzqU) Реклама Chromatic
- [\[medium.com\]](https://medium.com/storybookjs) Component-Driven Development