



Извлекаем максимум пользы из Unit тестирования





Никита Трофимов | Frontend Developer
2,5 года в Tinkoff Travel

Tinkoff travel

Покупатель

На почуу после оплаты пришлем маршрутную квитанцию, по телефону сообщим об изменениях на рейсе

Электронная почта

Номер телефона

Поле нужно заполнить

Пассажиры

Взрослый 3 425 ₽

Фамилия	Имя	Отчество	?	Дата рождения
Гражданство Россия	Удостоверение личности Паспорт РФ	Серия и номер		
Бонусная карта				

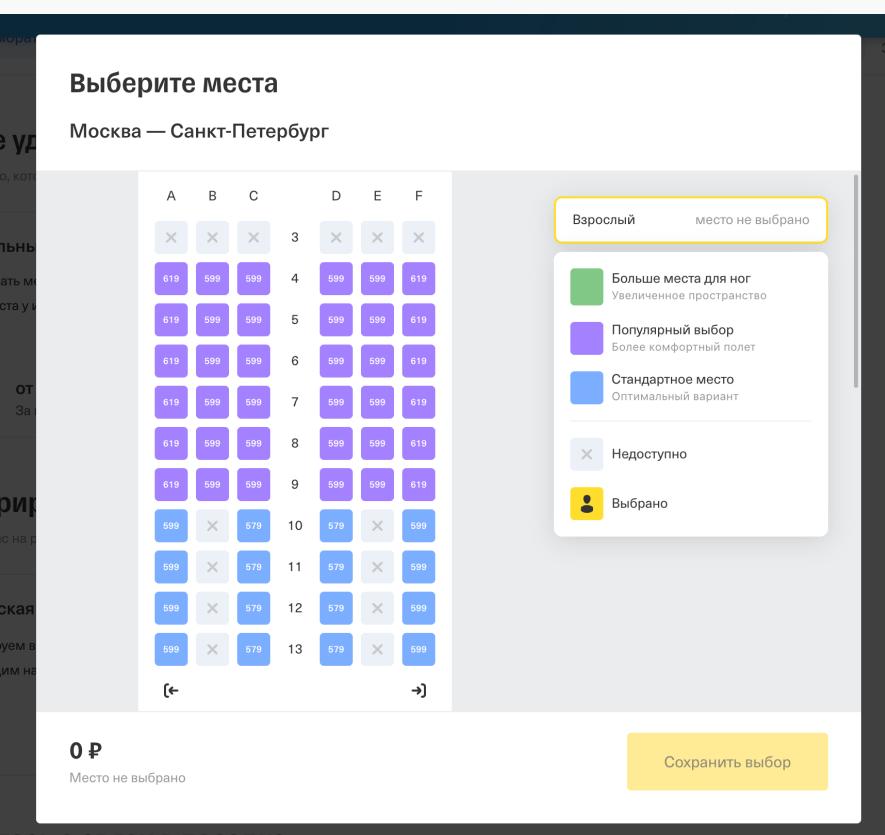
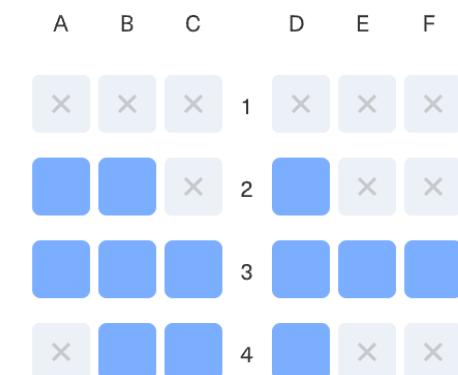
Выберите удобные места

Забронируем место, которое вы выберете. Номер места добавим в маршрутную квитанцию

Самостоятельный выбор места

- ✓ Можно выбрать место с увеличенным пространством для ног
- ✓ Покажем места у иллюминатора или в проходе

Выбрать от 559 ₽
За место



Зарегистрируйтесь на рейс заранее

Зарегистрируем вас на рейс и пришлем посадочный талон за сутки до вылета. Подробные [условия покупки](#)

Автоматическая регистрация

- ✓ Зарегистрируем вас на рейс и пришлем посадочный талон
- ✓ Сами отследим начало регистрации

Добавить



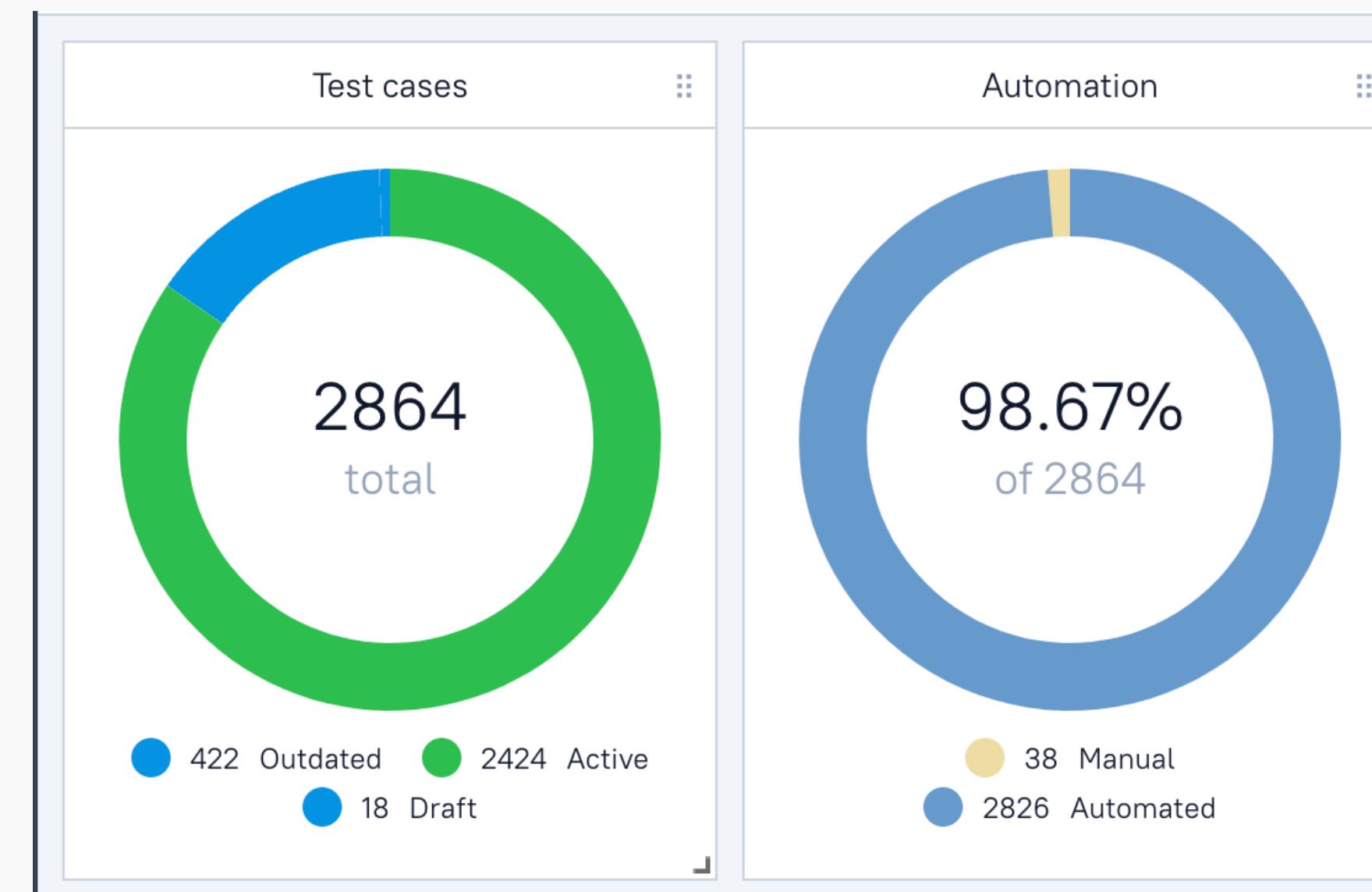
150 ₽

За пассажира

Позаботьтесь о связи в поездке

Доставим сим-карту по адресу, который укажете после оплаты авиабилета. Подробные [условия акции](#)

Tinkoff travel



О чём поговорим

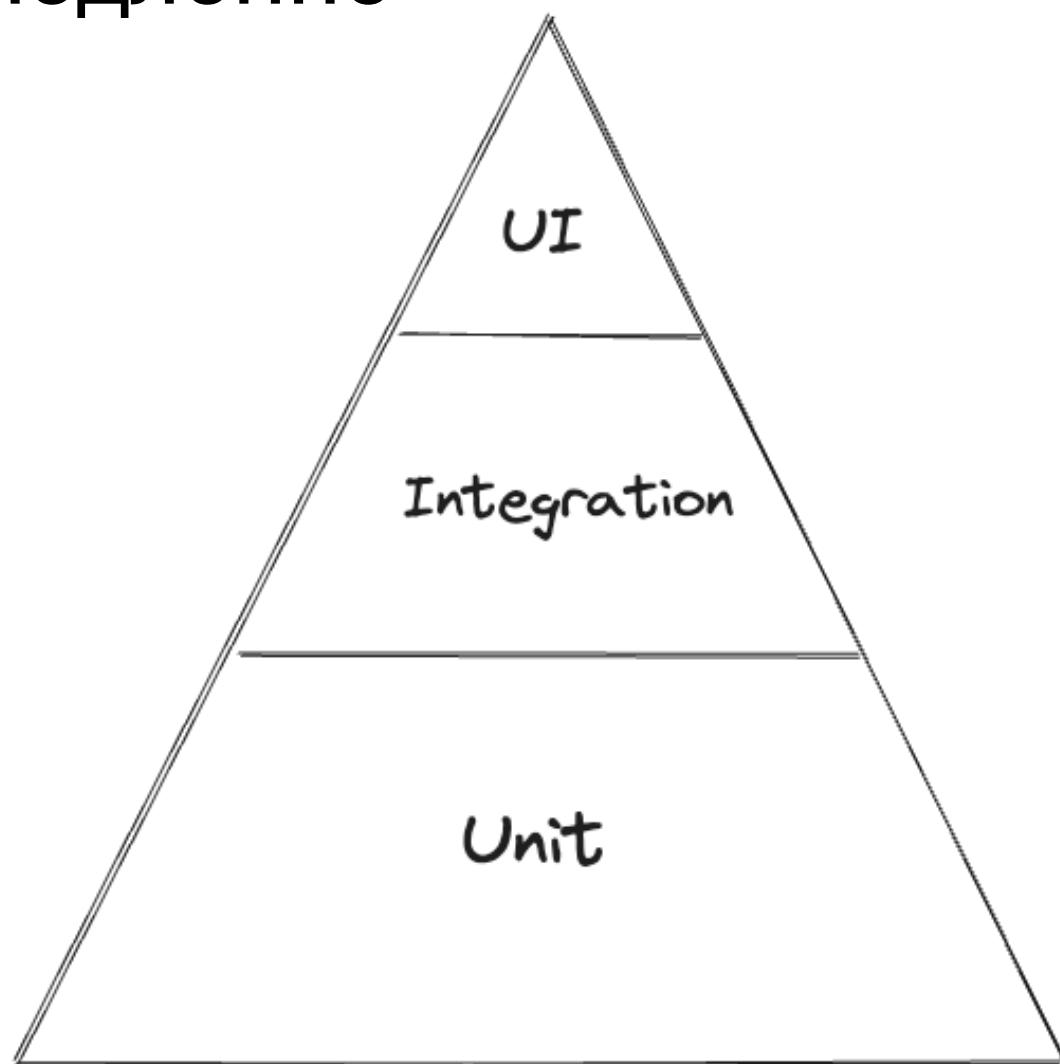
- О тестировании фронта
- О проблемах, вспомним пирамиду
- Почему UI это боль
- О том как писать полезные Unit тесты

Зачем мы пишем тесты?

Надежность

Пирамида тестирования

Ценно, сложно, медленно



Быстро, легко, менееично

Unit тесты

Проверяем все граничные значения работы модуля

Интеграционные тесты

Модуль А работает так, как ожидает модуль В

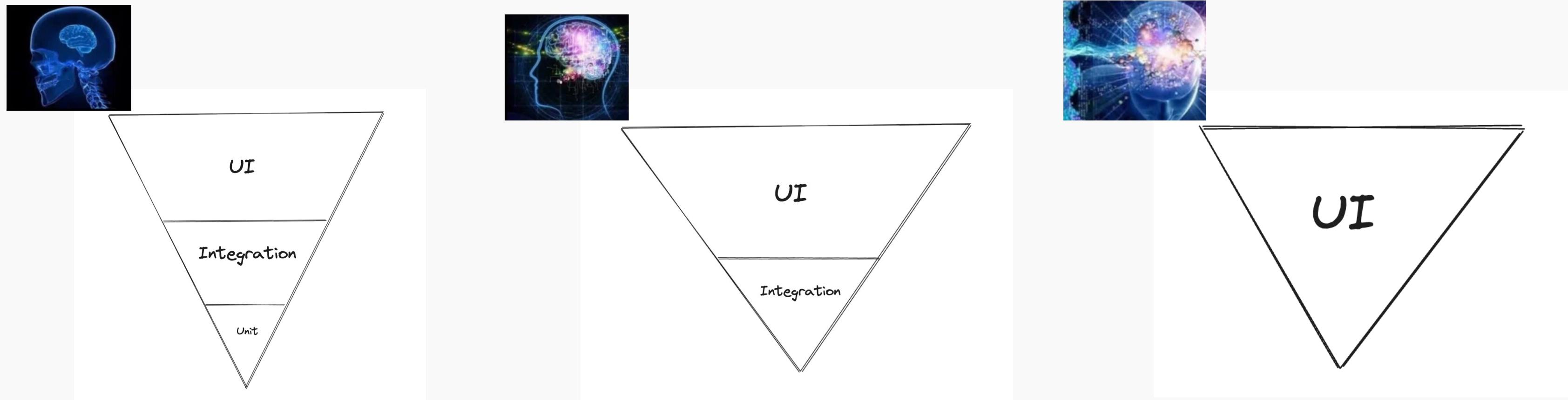
UI тесты

Приложение работает так, как ожидает пользователь

Пользователь такой



Но зачастую получается наоборот



Почему так происходит?

- Автоматизация
- Бизнес Value
- Мало экспертизы в Юнитах



Unit тесты



UI тесты

Почему это проблема?

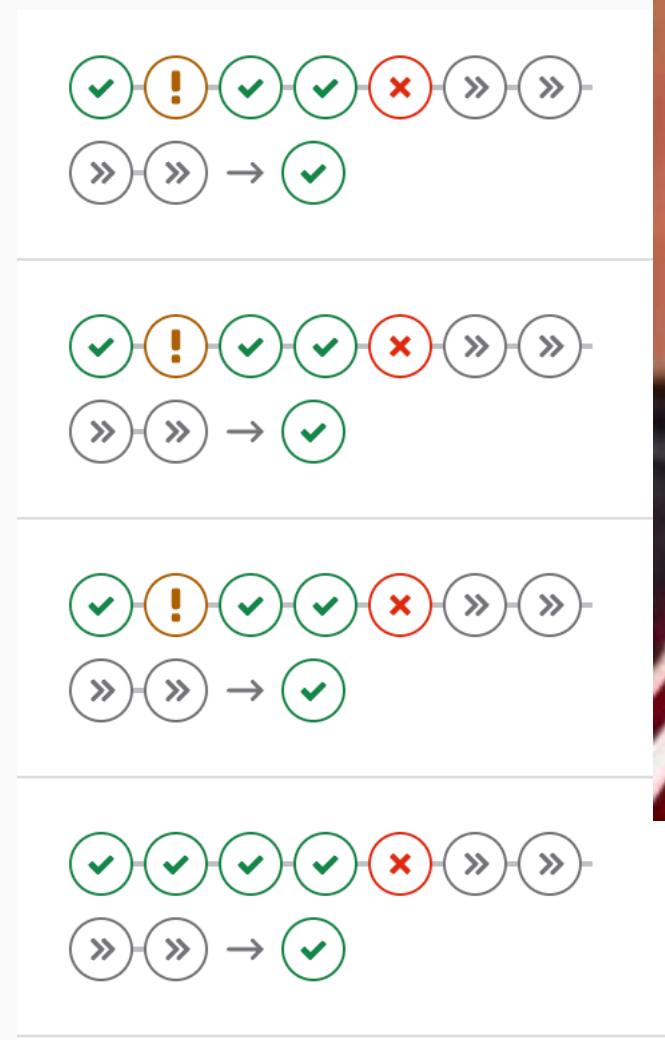
дорого и сложно поддерживать

Человеческие и инфраструктурные ресурсы

- Выбор инструментов, поддержка
- Сложность настройки CI
- Долго, нестабильно

Частые ошибки и flaky тесты

```
Only local connections are allowed.  
Please see https://chromedriver.chromium.org/security-considerations for suggestions on keeping ChromeDriver safe.  
ChromeDriver was started successfully.  
[0-0] RUNNING in chrome - /dist/generateScreenshotTests.js  
[0-0] 2023-04-07T09:12:34.764Z ERROR webdriver: Request failed with status 500 due to unknown error: unknown error: net::ERR_CONNECTION_REFUSED  
(Session info: headless chrome=112.0.5615.49)  
[0-0] unknown error in "generateTests "before all" hook for "stub""  
unknown error: net::ERR_CONNECTION_REFUSED  
(Session info: headless chrome=112.0.5615.49)
```



```
07:10] ERROR: Ваши версии chrome и chromedriver несовместимы.  
=====  
локальный chrome а также установите последнюю версию chromedriver командой:  
npm install chromedriver --chromedriver_version=LATEST  
=====
```

```
[0-6] 2023-03-06T20:15:50.980Z ERROR webdriver: Request failed with status 500 due to session not created: failed to allocate webdriver: browser  
chrome version 102.0 is not supported  
[0-6] 2023-03-06T20:15:50.981Z ERROR webdriver: session not created: failed to allocate webdriver: browser chrome version 102.0 is not supported
```

Частые ошибки и flaky тесты



Nikita Trofimov 16:51

Ребят, чет тесты упали, не могу MP влить из-за этого
Никто не в курсе, когда починят? Edited

Labels

skip:ui

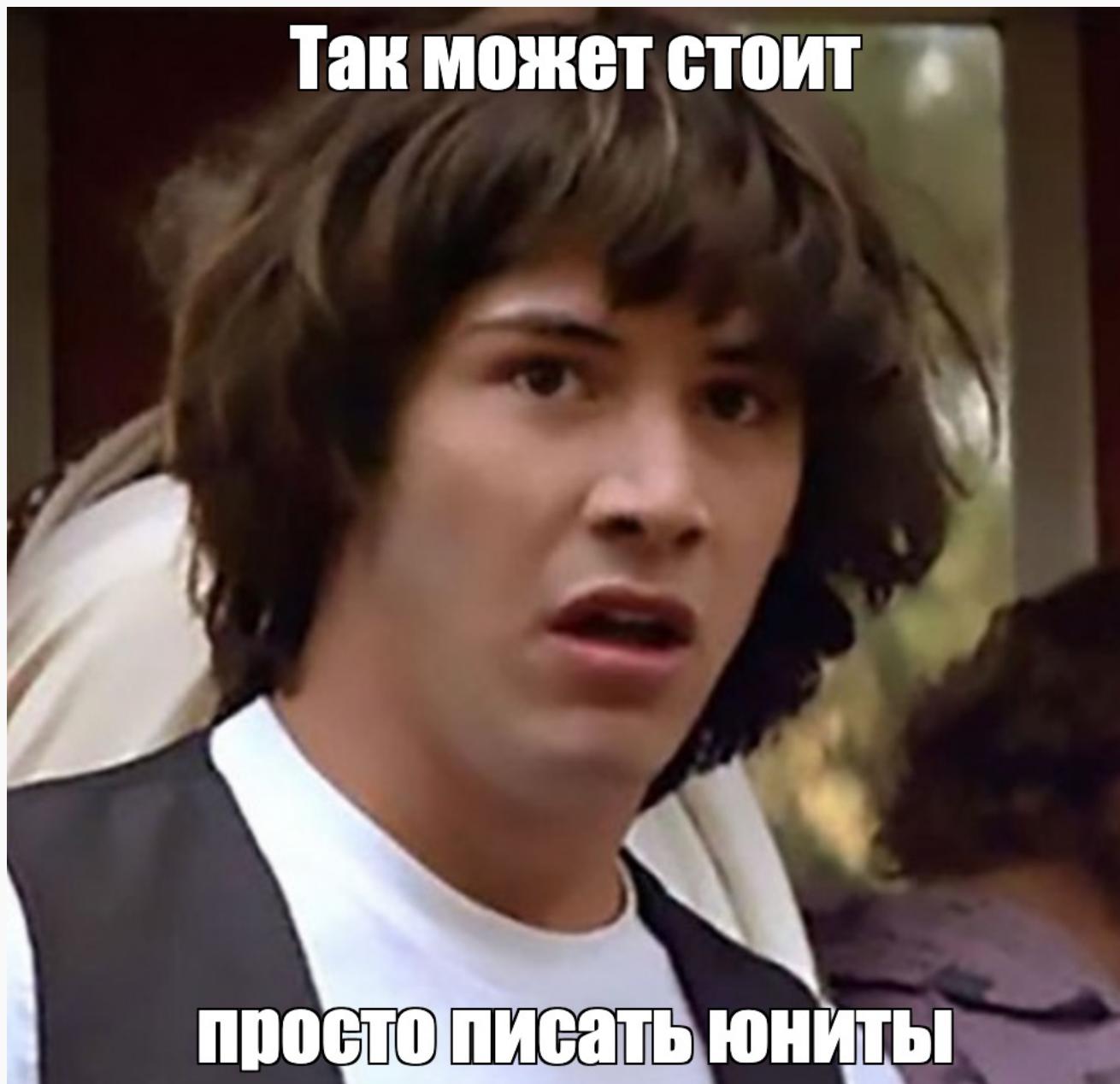
Удаленность от кода

- Растет технический долг
- Баги и сложность масштабирования

Почему Unit тесты это хорошо?

- Приближенны к коду
- Проще в поддержке
- Быстрее

И мы такие

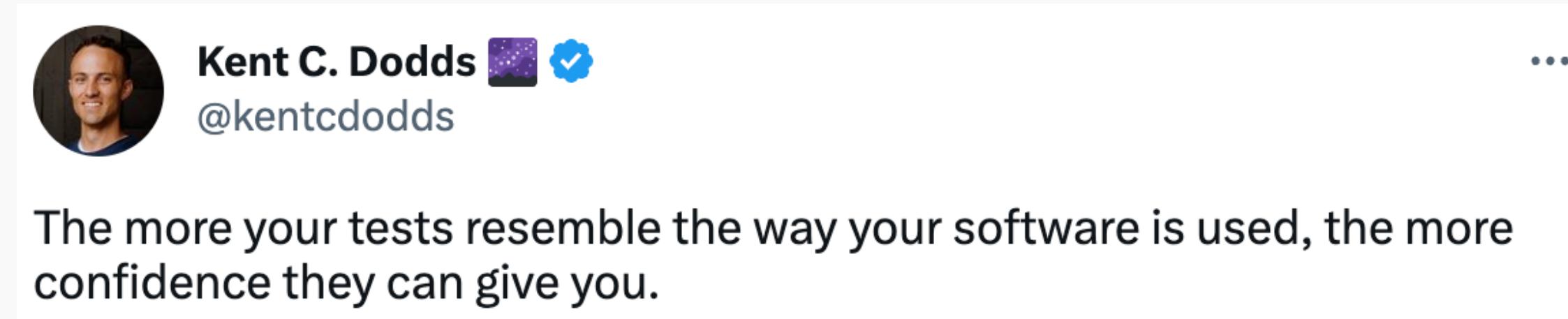


Так может стоит

просто писать юниты

А можно ли как в UI тестах?

Основная мысль



A screenshot of a Twitter post from Kent C. Dodds (@kentcdodds). The post features a profile picture of Kent, his name 'Kent C. Dodds' followed by a purple star icon and a blue checkmark, and his handle '@kentcdodds'. The main text of the tweet is 'The more your tests resemble the way your software is used, the more confidence they can give you.' Below the English text is a Russian translation: 'Чем больше ваши тесты похожи на то, как пользователи пользуются приложением, тем больше гарантий они могут вам дать'.

The more your tests resemble the way your software is used, the more confidence they can give you.

Чем больше ваши тесты похожи на то, как пользователи пользуются приложением, тем больше гарантий они могут вам дать

Что мы можем в UI тестах?

- Имитация действий пользователя (click, change, focus, ...)
- Проверка API (заглушки, вызовы, параметры)
- Внешний вид (dom, верстка)
- Браузерные хранилища (local storage, cookie)
- Переходы по страницам

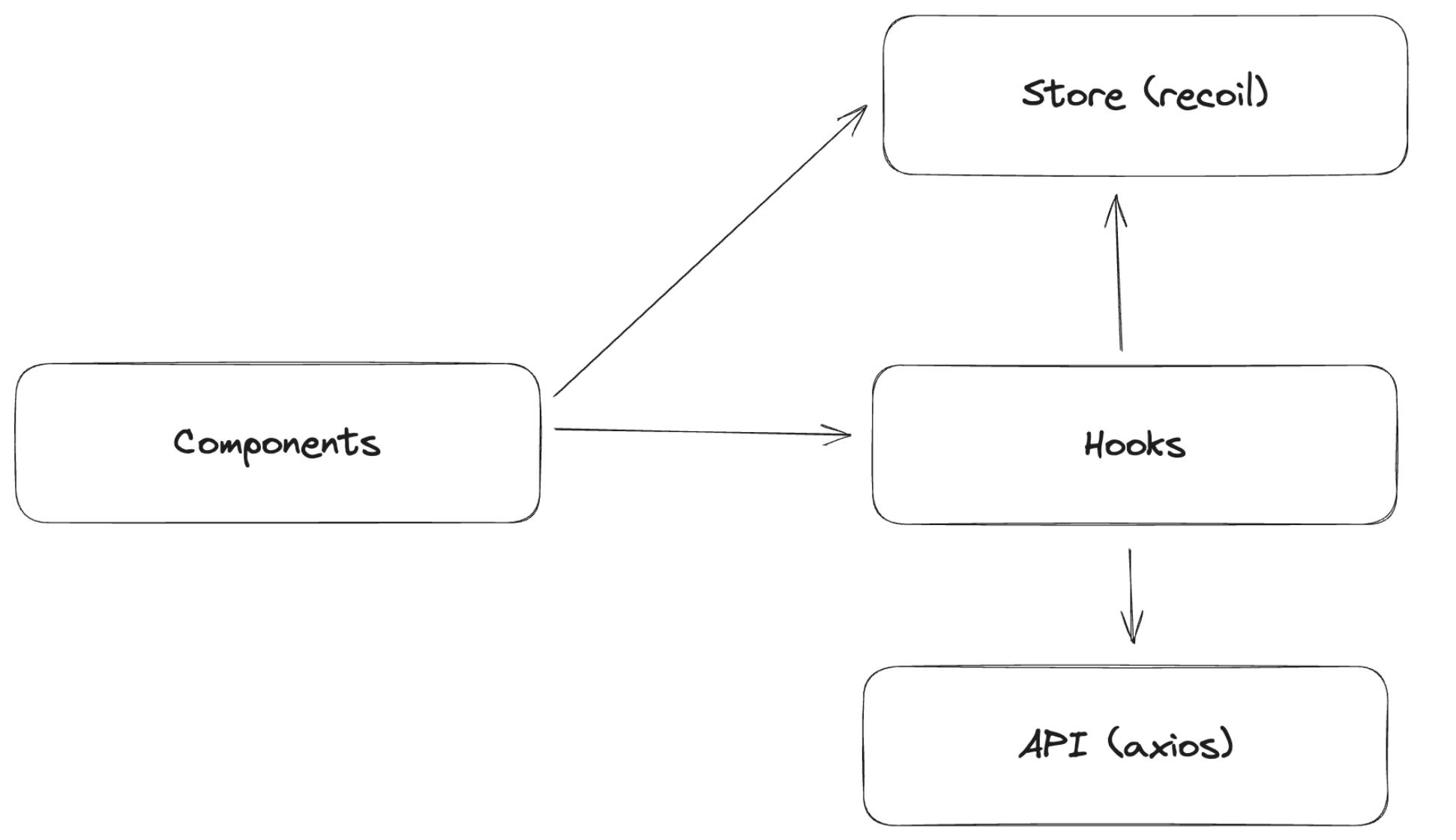
Что будем тестить?



Архитектура фронта

```
✓ feature/todo
  ✓ components
    > __mocks__
    > __tests__
    > TodoForm
    > TodoItem
    > TodoList
    > TodosWrapper
  TS TodoFeature.stories.tsx
  TS TodoFeature.tsx
  ✓ hooks
    TS useTodosActionsHook.ts
  ✓ services
    TS constants.ts
    TS todosApi.ts
  ✓ store
    TS todoSelectors.ts
    TS todoStore.ts
```

TodoFeature



Основные тестовые сценарии

- Пользователь добавил задачу
- Пользователь открыл страницу и увидел задачи
- Пользователь отметил задачу выполненной
- ...

Как будем тестировать?

- Тестирование состояний Storybook (Как выглядит приложение)
- Функциональное тестирование компонент testing-library (Как работает приложение)

Storybook тестирование для чего?

- Описываем истории для каждого компонента
- Пользовательский сценарий = набор историй
- Скриншотим и проверяем, что внешний вид не сломался

Конфигурируем и запускаем



```
1 npx storybook@latest init  
2 npm run storybook
```

Описываем истории



```
1  export const TodoListWithDoneTask: Story = {
2      parameters: {
3          msw: {
4              handlers: [
5                  rest.get(TODO_API, (_req, res, ctx) => {
6                      return res(ctx.json({ todos: [...TODO_TASKS, TASK_MOCK_4_DONE] }));
7                  }),
8              ],
9          },
10         viewport: {
11             defaultViewport: 'mobile2',
12         },
13     },
14 }
```

Можно обернуть в провайдеры



```
1 const meta: Meta<typeof TodoFeature> = {  
2     component: TodoFeature,  
3     decorators: [  
4         (Story) => (  
5             <QueryClientProvider client={queryClient}>  
6                 <RecoilRoot>  
7                     <Story />  
8                 </RecoilRoot>  
9             </QueryClientProvider>  
10        )  
11    ]  
12};
```

Как выглядят истории

The screenshot shows the Storybook interface displaying a todo list component. The left sidebar shows a tree structure under the 'FEATURE' section, with 'todo' expanded, showing 'components' and 'TodoFeature'. 'TodoList With Done Task' is selected and highlighted in blue. The main preview area shows a list of tasks:

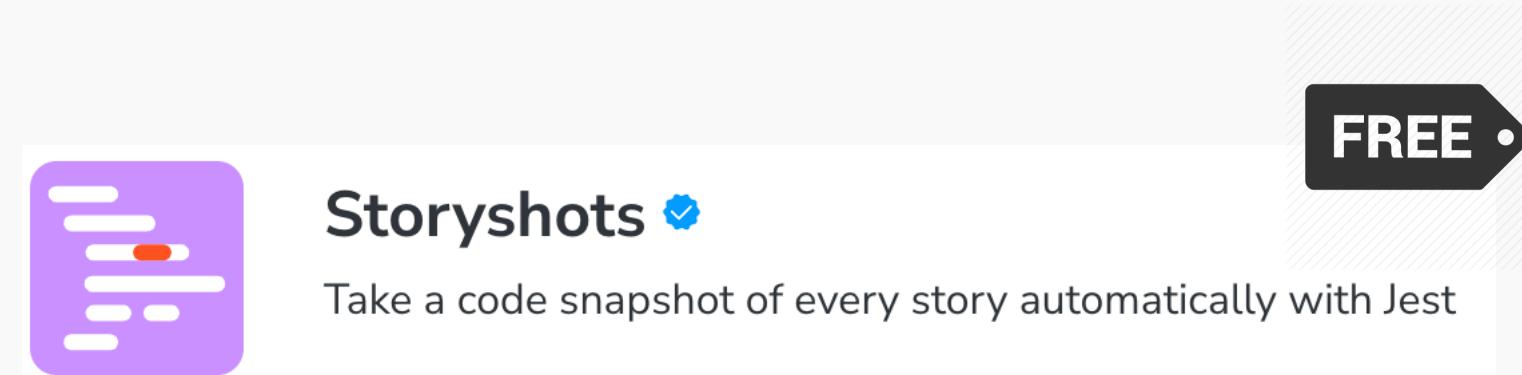
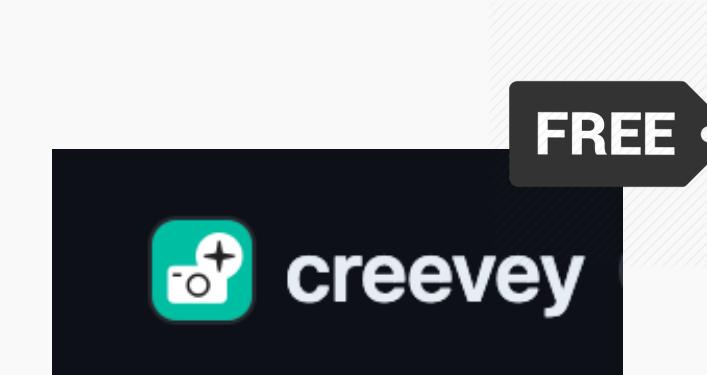
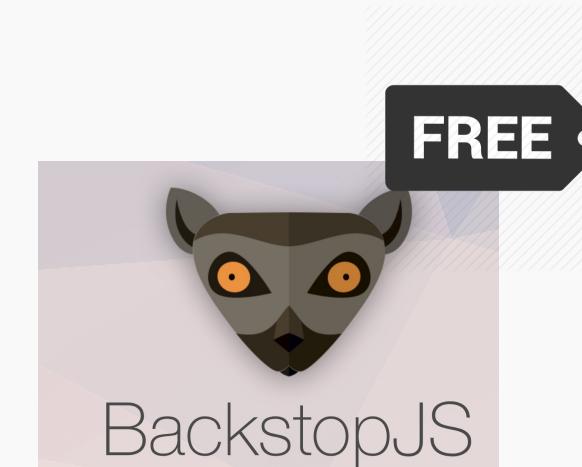
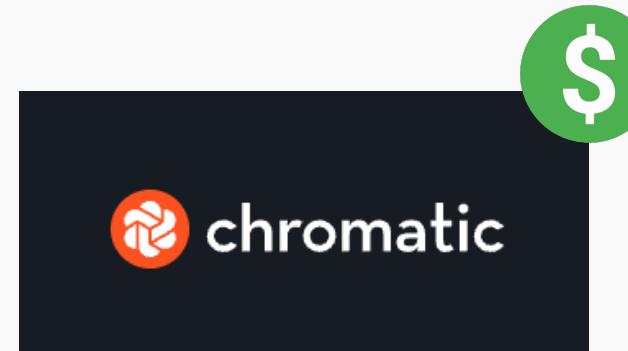
Что сделать?	
Проверить задачи в джире	<input checked="" type="checkbox"/> trash
Написать Ваше че там с API	<input checked="" type="checkbox"/> trash
Собрать шкаф	<input checked="" type="checkbox"/> trash
Сегодня точно доделал доклад	<input type="checkbox"/> trash

The preview toolbar at the top includes icons for refresh, search, image, grid, and a dropdown set to 'Large mobile (P)'. The status bar shows dimensions 414x896.

Какие возможности в Story?

- Заглушки для API и Store (msw-storybook-addon, storybook-addon-recoil-flow)
- Использовать стори в jest тестах (@storybook/testing-react)
- Делать скриншоты и проверять в CI

Инструменты для скриншот тестиования



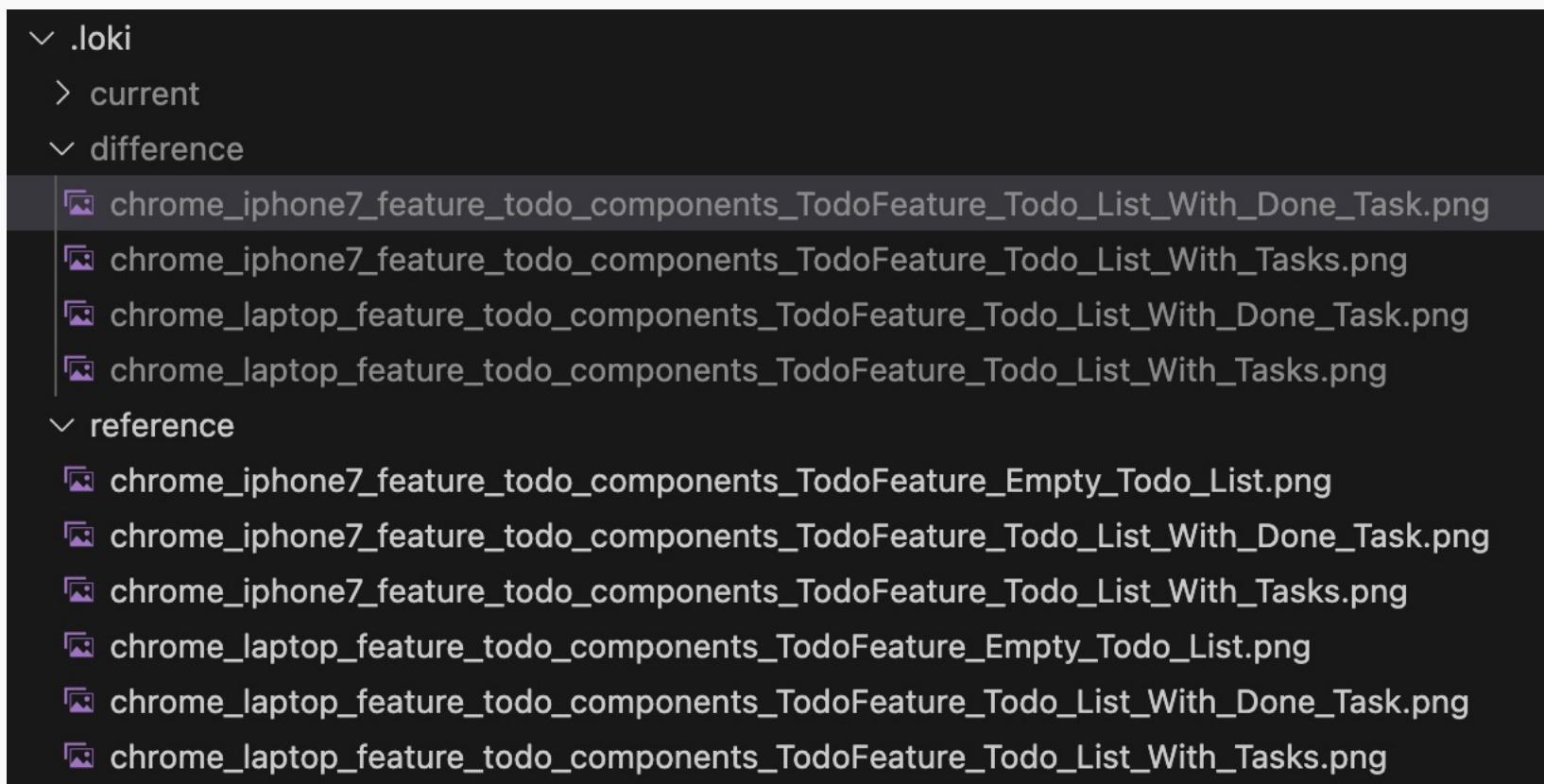
Скриншотим и проверяем



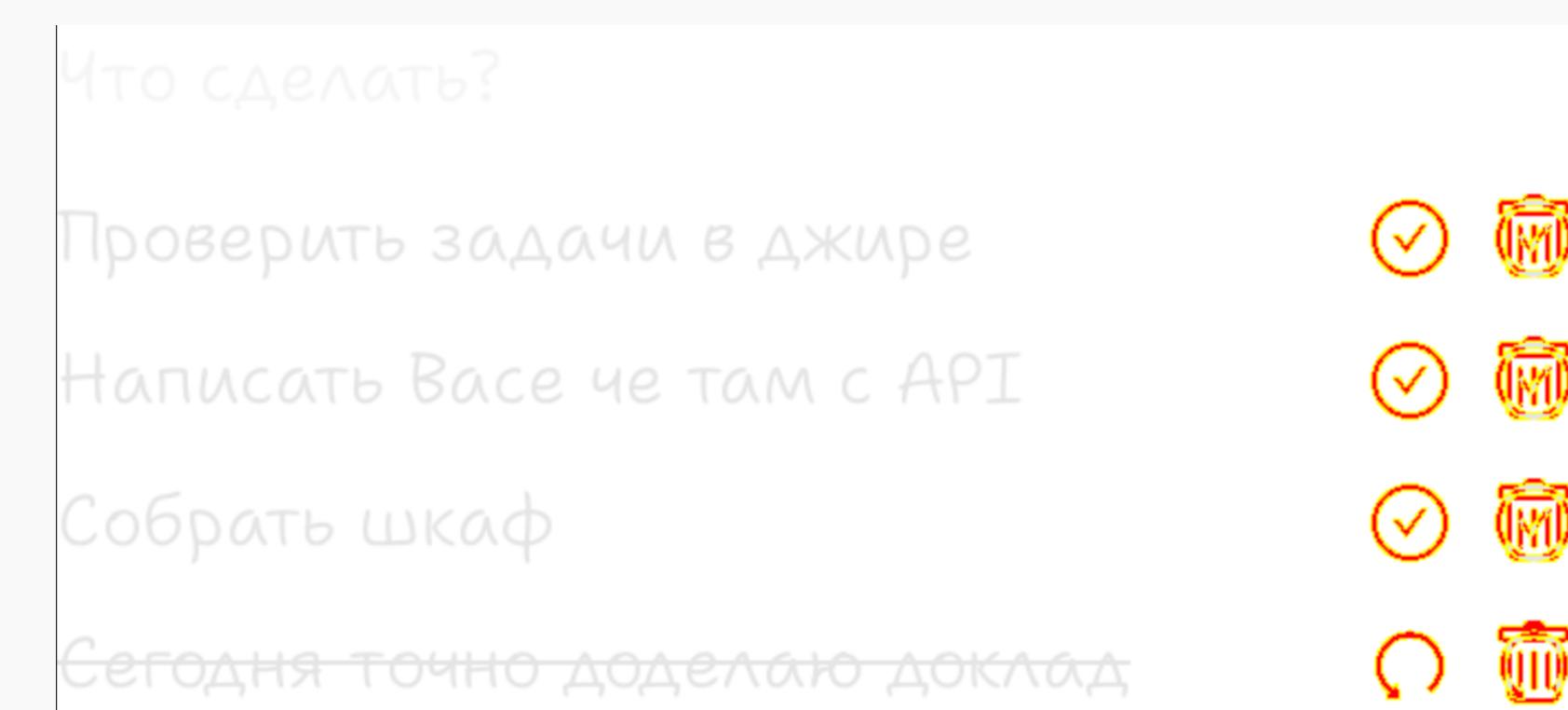
```
1 npm install loki
2 npx loki init
3 npx loki update
4 npx loki test
```

Скриншотим и проверяем

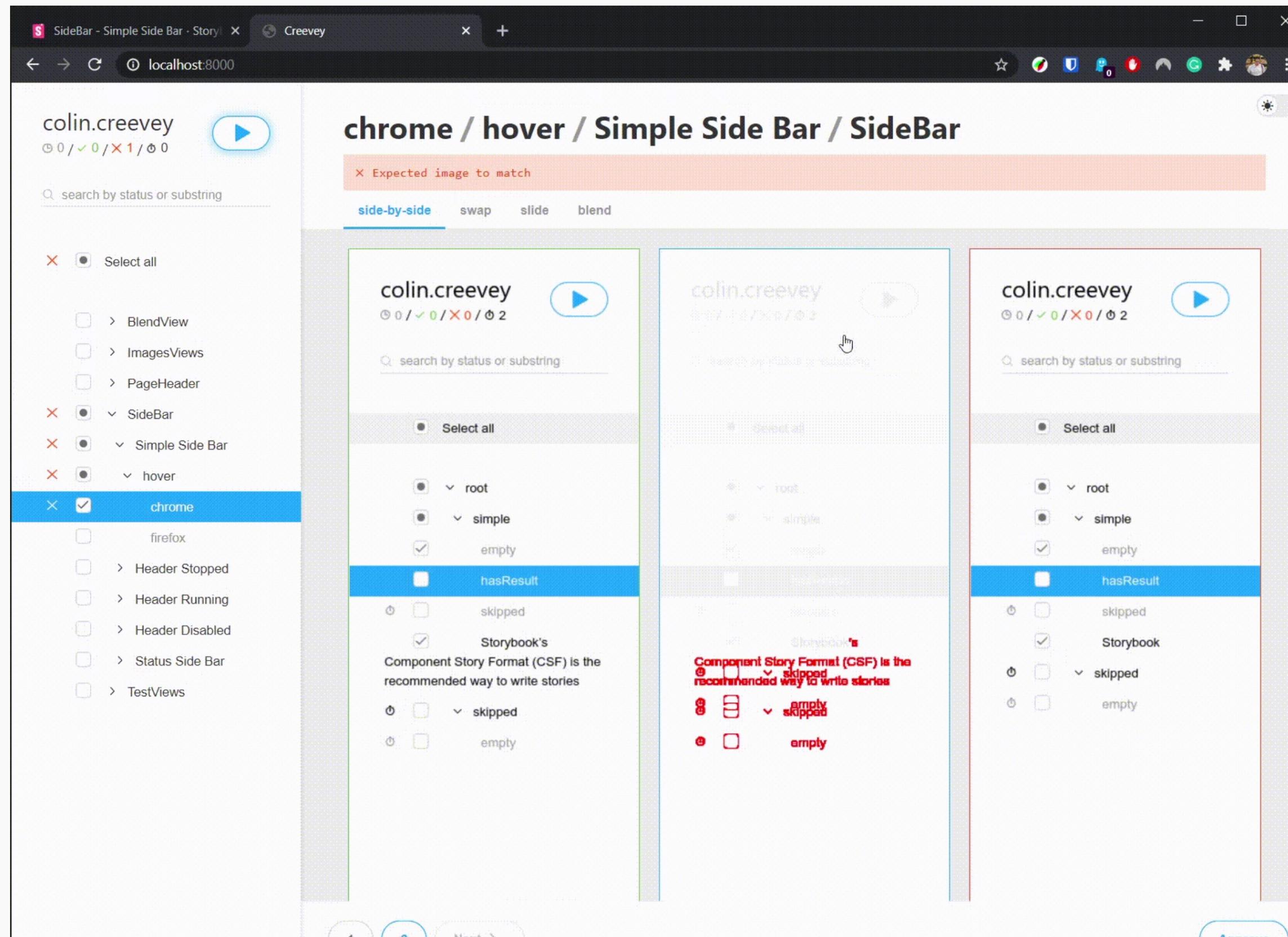
Как хранятся скриншоты



Как отображаются различия



Как это выглядит в creevey



Summary по storybook

- QA пишет тестовую модель
- Описываем истории на основе модели
- QA и дизайнер апрувят скриншоты
- Упал тест? Фиксим баг или обновляем скриншоты
- Можно настроить CI и выгружать в allure
- Любой фреймворк

**В верстке уверены, что по
функционалу?**

Что умеет функциональный тест

- Имитация действий пользователя
- Проверка вызова API
- Проверка рендеринга элементов
- Проверка обновления состояния в store
- Проверка вызова нужных методов

Примеры функционального теста testing-library



```
1 it('Отображаем фичу и проверяем, что список задач рендерится', async () => {  
2  
3 })
```

Мокируем API

(axios-mock-adapter)



```
1 it('Отображаем фичу и проверяем, что список задач рендерится', async () => {
2   // 0. Мокируем апи
3   mock.onGet(TODO_API).reply(200, {
4     todos: [TASK_MOCK_1_NOT_DONE]
5   });
6 })
```

Рендерим компонент



```
1 it('Отображаем фичу и проверяем, что список задач рендерится', async () => {
2   // 0. Мокируем апи
3   mock.onGet(TODO_API).reply(200, {
4     todos: [TASK_MOCK_1_NOT_DONE]
5   });
6
7   // 1. Рендерим компонент фичи
8   const { container } = render(<TodoFeature />, { wrapper: AppProviders })
9 })
```

Проверяем, что метод API вызвался



```
1 it('Отображаем фичу и проверяем, что список задач рендерится', async () => {
2   // 0. Мокируем апи
3   mock.onGet(TODO_API).reply(200, {
4     todos: [TASK_MOCK_1_NOT_DONE]
5   });
6
7   // 1. Рендерим компонент фичи
8   const { container } = render(<TodoFeature />, { wrapper: AppProviders })
9
10  // 2. Проверяем, что вызвался нужный нам метод
11  expect(mock.history.get.some(method => method.url === TODO_API)).toBeTruthy();
12})
```

Проверяем DOM

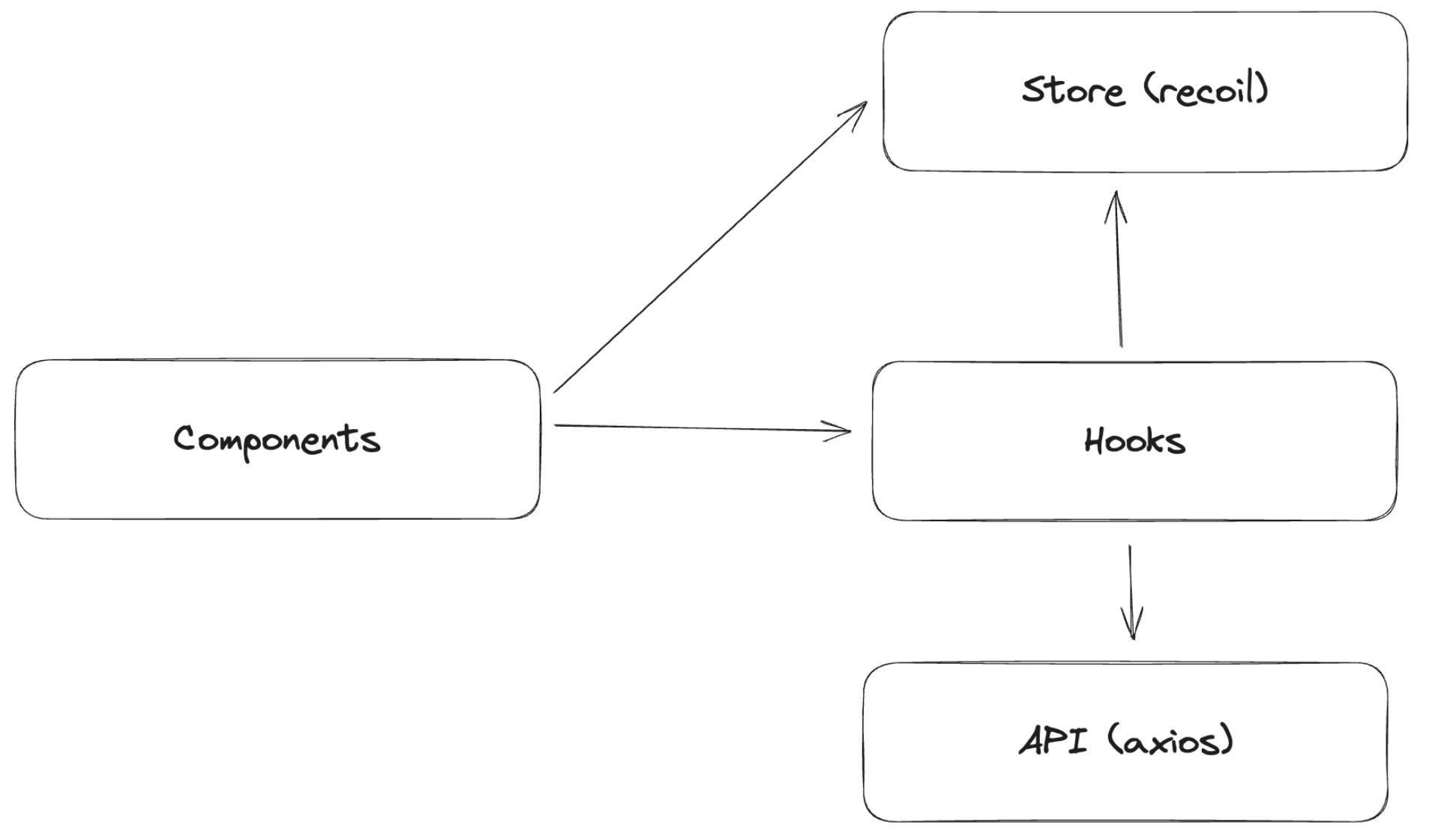
```
● ● ● test

1 it('Отображаем фичу и проверяем, что список задач рендерится', async () => {
2
3   const todoList = queryHelpers.queryAllByAttribute('data-qa-type', container, 'todo-item');
4   const checkBtn = queryHelpers.queryByAttribute('data-qa-type', todoList[0], 'check-btn');
5   const unCheckBtn = queryHelpers.queryByAttribute('data-qa-type', todoList[0], 'uncheck-btn');
6   const checkTasksInJiraTodoItem = screen.queryByText('Проверить задачи в джире')
7
8   // 3. Проверяем, рендер
9   expect(todoListContainer.length).toBe(1);
10  expect(checkTasksInJiraTodoItem).toBeInTheDocument();
11  expect(checkBtn).toBeInTheDocument();
12  expect(unCheckBtn).not.toBeInTheDocument();
13})
```

Архитектура фронта

```
✓ feature/todo
  ✓ components
    > __mocks__
    > __tests__
    > TodoForm
    > TodoItem
    > TodoList
    > TodosWrapper
  TS TodoFeature.stories.tsx
  TS TodoFeature.tsx
  ✓ hooks
    TS useTodosActionsHook.ts
  ✓ services
    TS constants.ts
    TS todosApi.ts
  ✓ store
    TS todoSelectors.ts
    TS todoStore.ts
```

TodoFeature



Проверка post запросов с параметрами

```
1 it('Пользователь отмечает задачу выполненной', async () => {
2   ...
3
4   // Мокаем post запрос
5   const checkSpy = jest.fn();
6   mock.onPost(`${TODO_API}/check`).replyOnce((request) => {
7     checkSpy(request.data);
8     return [200, {}]
9   });
10
11  // Кликаем кнопку
12  await act(async () => {
13    fireEvent.click(firstTodoItemCheckBtn)
14  });
15
16  // Проверяем, что запрос отправился с нужными параметрами
17  expect(checkSpy).toHaveBeenCalledWith(JSON.stringify({ id: "1" }));
18})
```

Какие еще есть инструменты мокирования?

- miragejs
- msw
- nock
- axios-mock-adapter

Какие еще возможности?

Мок вызова функций



test

```
1 // Мок вызова метода хука
2 const useSomeHookMock = jest.fn();
3 jest.doMock('src/hook/useSomeHook', () => ({
4   __esModule: true,
5   useSomeHook: () => useSomeHookMock,
6 }));
7
8 // Проверяем в тесте
9 expect(useSomeHookMock).toHaveBeenCalledTimes(1);
```

Мок библиотек



test

```
1 jest.doMock('@some-ui-lib/popup', () => ({
2   PopupComponent: () => () => <div />,
3 }));
```

Check routing



test

```
1 it('Пользователь переходит на страницу настроек', async () => {
2
3   const {container} = renderWithRouter(<TodoPage />)
4   expect(screen.getByText(/Simple Todo App/i)).toBeInTheDocument()
5
6   await container.click(screen.getByText(/settings/i))
7
8   expect(screen.getByText(/Settings Page/i)).toBeInTheDocument()
9 })
```

Хочу все это видеть в allure

The screenshot shows the Allure UI interface. On the left is a sidebar with navigation links: Overview, Categories, Suites, Graphs, Timeline, Behaviors, and Packages. The main area is titled "Suites" and lists test results. A summary at the top right shows 0 failed, 0 skipped, 8 passed, 0 pending, and 0 broken tests. Below this, a "Marks" section includes icons for each status. The test suite "io.qameta.allure.IssuesRestTest" is expanded, showing four test cases: #1 (passed), #2 (passed), #3 (passed), and #4 (passed). Each case has a duration listed (110ms, 3ms, 3ms, 3ms) and a note description. The "io.qameta.allure.IssuesWebTest" and "io.qameta.allure.PullRequestsWebTest" suites are also listed with their respective counts. To the right of the main table, a detailed view of the first test case (#1) is shown. It includes the test ID, name, status (Passed), severity (normal), duration (110ms), owner (baev), parameters (Title: First Note), execution details, and a "Test body" section with two steps: "Create issue with title 'First Note'" and "Check note with content 'First Note' exists".

Suites

order name duration status

Marks: ● ✖ ✓ ! ⌚

Status: 0 0 8 0 0

io.qameta.allure.IssuesRestTest 4

- #1 shouldCreateUserNote(String) Create issue via api First Note 110ms
- #2 shouldCreateUserNote(String) Create issue via api Second Note 3ms
- #3 shouldDeleteUserNote(String) Close issue via api First Note 3ms
- #4 shouldDeleteUserNote(String) Close issue via api Second Note 3ms

> io.qameta.allure.IssuesWebTest 2

> io.qameta.allure.PullRequestsWebTest 2

io.qameta.allure.IssuesRestTest.shouldCreateUserNote

Passed **shouldCreateUserNote(String)** Create issue via api

Overview History Retries

Tags: smoke api

Severity: normal

Duration: 110ms

Owner baev

Parameters

Title: First Note

Execution

Test body

- Create issue with title 'First Note'
3 parameters, 1 sub-step
- Check note with content 'First Note' exists
3 parameters, 2 sub-steps

Выгружаем в allure

- Есть open source решения (jest-allure)
- Можно написать обвязку

Выгружаем в allure локально

```
jest.config.js
```

```
1  {
2  ...
3  testRunner: "jest-jasmine2",
4  setupFilesAfterEnv: ["jest-allure/dist/setup"],
5  ...
6 }
```

```
jest allure api
```

```
1 npm install jest-allure
2 npm run test
3
4 brew install allure
5 allure server
```

Запускаем отчет

The screenshot shows the Allure test reporting interface. On the left is a dark sidebar with navigation links: Overview, Categories, Suites (selected), Graphs, Timeline, Behaviors, and Packages. The main area has a light gray header with the title "Suites". Below it is a search bar and a status summary: Status: 1 0 1 0 0. A "Marks" section shows five icons: a green checkmark, a red X, a blue checkmark, a yellow info icon, and a purple info icon. The main content area displays a tree view under the heading "Тестирование фичи списка задач". Two items are listed: "#1 Отображаем фичу и проверяем, что список задач рендерится" (Status: 1 1, Duration: 0s) and "#2 Пользователь отмечает задачу выполненной" (Status: 0 1, Duration: 0s). The second item is highlighted with a yellow background. To the right of the tree view is a detailed report for the failed test "#2". The report title is "Пользователь отмечает задачу выполненной" (Failed). The "Overview" tab is selected. The error message is: "Error: expect(received).toBe(expected) // Object.is equality". Below it, the expected and received values are shown: "Expected: 3" and "Received: 1". Further down, categories, severity, duration, and execution details are listed: "Categories: Product defects", "Severity: normal", "Duration: 0s", and "Execution: No information about test execution is available."

jest-allure API



jest allure api

```
1  description(description: string): this;
2  severity(severity: Severity): this;
3  epic(epic: string): this;
4  feature(feature: string): this;
5  story(story: string): this;
6  startStep(name: string): this;
7  endStep(status?: Status): this;
8  addArgument(name: string): this;
9  addEnvironment(name: string, value: string): this;
10 addAttachment(name: string, buffer: any, type: string): this;
11 addLabel(name: string, value: string): this;
12 addParameter(paramName: string, name: string, value: string): this;
```

Как это выглядит у нас



The screenshot shows a terminal window with three colored status indicators (red, yellow, green) at the top. The title bar reads "jest allure api". The main area contains a block of code:

```
1 describe('Описание тестовых сценариев', () => {
2   it(
3     'Что тестируем',
4     withAllure({ id: 906227, labels: [] }), async () => {
5
6       step('Рендерим фичу', () => {
7         ...
8       });
9       step('Нажимаем на кнопку', () => {
10      ...
11    });
12    step('Проверяем, что цена изменилась', () => {
13      ...
14    });
15  });
16);
17
```

Summary по функциональным тестам

- Можно проверять (API, рендер, вызовы функций, стор, роуты)
- Описываем в формате BDD и выгружаем в allure
- Любой фреймворк

Полезные практики

Общий метод рендеринга

```
✓ const AppProviders: React.JSXElementConstructor<children: React.ReactElement> = ({ children }) => {
  const initialState: InitialStateType = {}
  return <QueryClientProvider client={queryClient}>
    <RecoilRoot initializeState={initialState}>
      <BrowserRouter>
        <AuthProvider>
          <HeaderProvider>
            {children}
          </HeaderProvider>
        </AuthProvider>
      </BrowserRouter>
    </RecoilRoot>
  </QueryClientProvider>
}

export const renderWithAppProviders = (ui: React.ReactElement) => render(ui, { wrapper: AppProviders })
```

Организация моков

```
export const CABIN_IN_AST_MOCK: RuleUnit = { ... };
export const PRIORITY_WITH_PARENS_AST_TOKEN: RuleUnit = { ... };
export const ROUTE_IS_BETWEEN_AST_MOCK: RuleUnit = { ... };
export const FORMULA_WITH_PREFIX_AND_MULTIPLY_OP_AST_MOCK: RuleUnit = { ... };
export const FLOAT_FORMULA_WITH_COMPLICATED_INTERVAL_AST_MOCK: RuleUnit = { ... };
export const FORMULA_WITH_PREFIX_AND_PARENS_PRIORITY_AST_MOCK: RuleUnit = { ... };
export const COMPLICATED_VALUE_WITH_FACT_AS_BOOL_ARG: RuleUnit = { ... };
export const INTERVAL_CLOSED_OPEN_AST_MOCK: RuleUnit = { ... };
export const FIXED_PREFIX_OPS_COMPARE_AST_MOCK: RuleUnit = { ... };
export const DATE_TIME_COMPARE_WITH_REGEX_AST_MOCK: RuleUnit = { ... };
export const IN_RANGE_DATE_AST_MOCK: RuleUnit = { ... };
export const IN_RANGE_TIME_AST_MOCK: RuleUnit = { ... };
export const AST_WITH_NOT_OPERATOR: RuleUnit = [ ... ];
```



Общий билдер для моков

```
export class StoreBuilder {
    private builder = new CommonStoreBuilder();

    public prepareCompletePage() {}

    withAuthUser(params) {
        const userStoreBuilder = new UserStoreBuilder(params);
        this.builder.setUserStoreFromBuilder(userStoreBuilder);
        return this;
    }

    withOffer(params) {
        const offerStoreBuilder = new OfferStoreBuilder(params);
        this.builder.setOfferStoreBuilder(offerStoreBuilder);
        return this;
    }

    getResult() {
        return this.builder.getResult();
    }
}
```



```
const makeMock = (params) => {
    const mock = new FixtureBuilder()
        .withAuthUser(params)
        .withOffer(params)
        .withTravelConfig()
        .withUser();

    return mock.getResult();
};
```



Описание элементов страницы

```
export const getPaymentPageElements = (container: HTMLElement) => {
  const paymentButton = container.querySelector('[data-qa-type=paymentButton]');
  const totalPrice = container.querySelector('[data-qa-type=totalPriceBlock]');
  const discount = container.querySelector('[data-qa-type=discountBlock]');

  return {
    paymentButton,
    totalPrice,
    discount
  };
};
```

Проблемы с ожиданием асинхронных операций

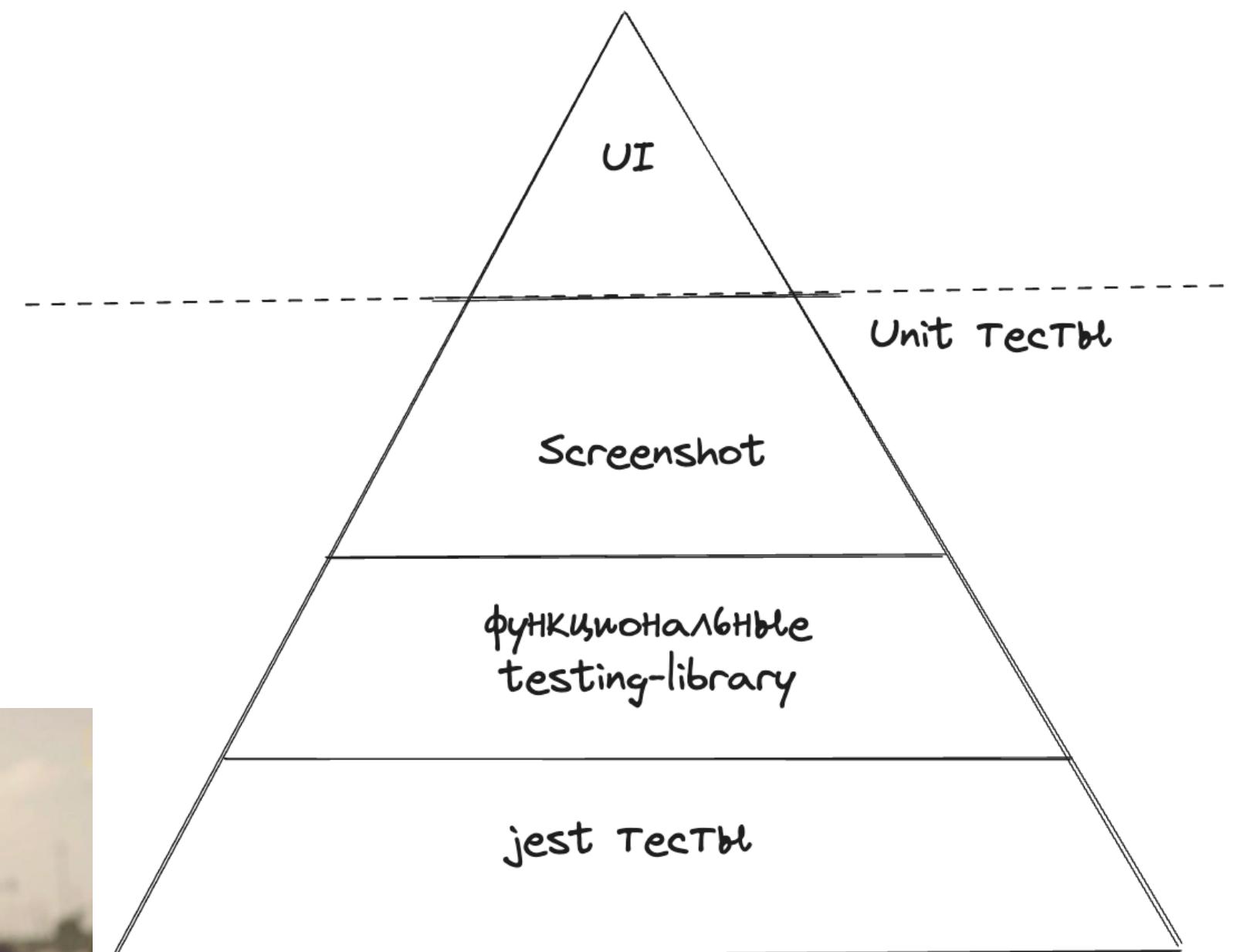
```
export const makeDelay =  
  async (timeout = 0) =>  
    async () =>  
      new Promise((resolve) => setTimeout(resolve, timeout));
```

```
act(  
  () =>  
    new Promise(resolve => {  
      setTimeout(resolve, 100);  
      jest.runAllTimers();  
    }),  
);
```

```
await act(async () => await customRender(<TodoFeature />, { wrapperProps: { store: { todoStore: TODO_TASKS } } }))
```

Придется повозиться с конфигурацией

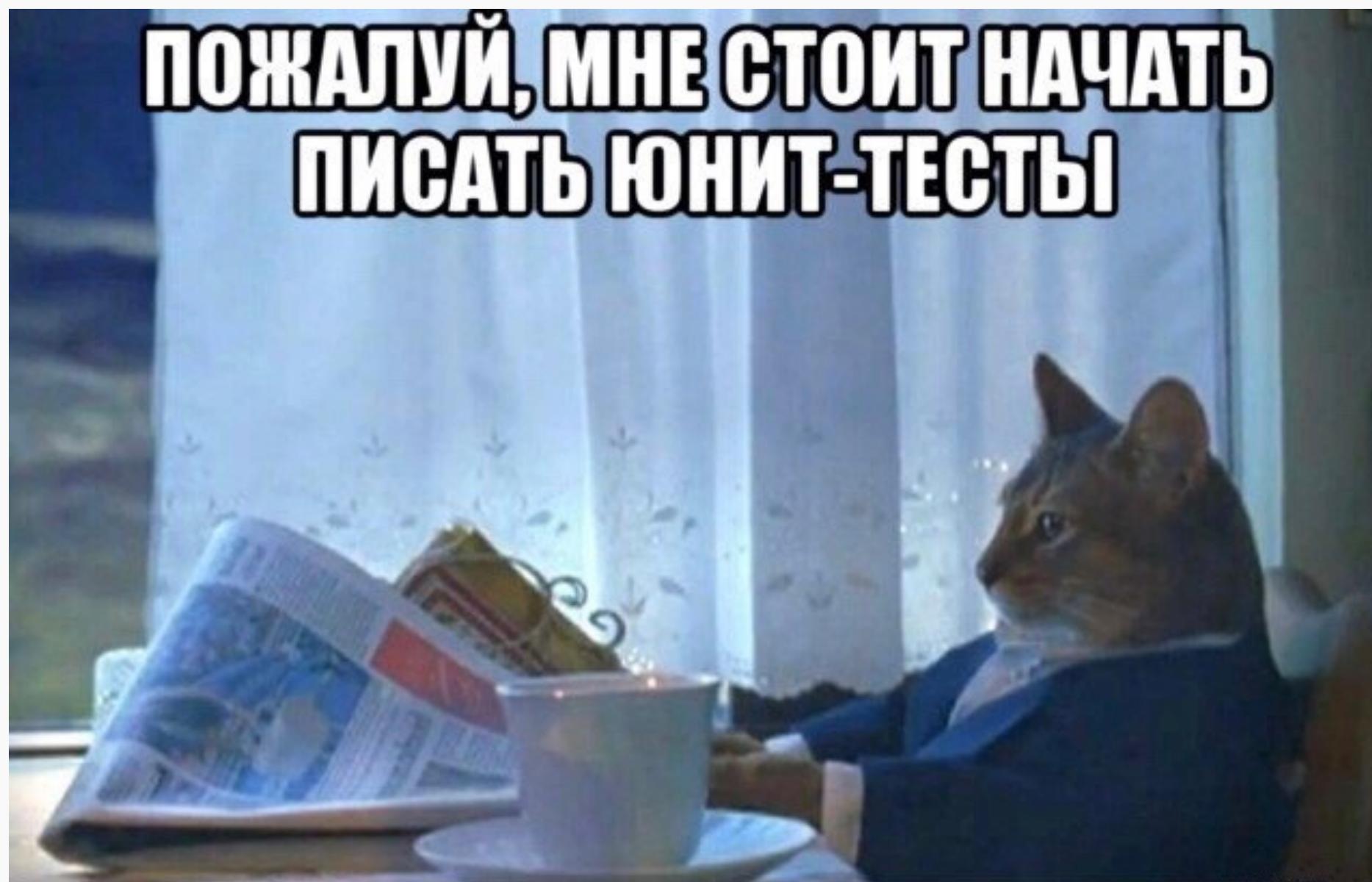
Что имеем в итоге?



Summary

- Большинство кейсов можно покрыть юнитами
- Меньше ресурсов на поддержку тестов
- Возрастает стабильность и скорость прогонов тестов в CI
- Возрастает культура тестирования
- Снижается барьер понимания между разработкой и QA

**ПОЖАЛУЙ, МНЕ СТОИТ НАЧАТЬ
ПИСАТЬ ЮНИТ-ТЕСТЫ**





Проект и ссылки

