📘Шаблон и памятка для команды: как писать Storybook Stories (React + TypeScript)

Этот документ — единый стандарт для команды, чтобы **все стори были одинаковыми, понятными и удобными для просмотра**, независимо от автора. Используй этот шаблон при создании новых компонентов.

🕑Цели шаблона

- Единый стиль написания сторис
- Удобное тестирование состояния компонентов
- Понятная документация
- Поддержка автогенерации Storybook Docs
- Уменьшение количества ошибок и дублирования

📁Структура файлов

Каждый компонент должен иметь следующую структуру:

```
ComponentName/
├─ ComponentName.tsx
├─ ComponentName.stories.tsx
└─ ComponentName.types.ts (опционально)
```

📄Базовый шаблон стори

Используйте **Component Story Format (CSF 3)** — официальный и рекомендуемый Storybook API.

```ts
// ComponentName.stories.ts
import type { Meta, StoryObj } from "@storybook/react";
import { ComponentName } from "./ComponentName";

// 🎛️ Метаданные — настройки для Storybook
const meta: Meta<typeof ComponentName> = {
  title: "Components/ComponentName",
  component: ComponentName,
  parameters: {
    layout: "centered", // По умолчанию компоненты центрируются
  },
  argTypes: {
```

```
    onClick: { action: "clicked" }, // Пример событий
  },
};

export default meta;
type Story = StoryObj<typeof ComponentName>;

// 🧹 Базовое состояние компонента
export const Primary: Story = {
  args: {
    label: "Primary button",
    variant: "primary",
  },
};

// 🪔 Альтернативные варианты
export const Disabled: Story = {
  args: {
    label: "Disabled button",
    disabled: true,
  },
};

// 🎴 Сложные примеры с декораторами
export const WithTheme: Story = {
  args: {
    label: "Themed button",
  },
  decorators: [
    (Story) => (
      <ThemeProvider value="dark">
        <Story />
      </ThemeProvider>
    ),
  ],
};
```

📏Правила для команды

✂️Каждый компонент должен иметь минимум 1–2** стори**:

- стандартное состояние (Primary)
- альтернативное состояние (Secondary/Disabled/Loading)
- особый кейс (в пустом состоянии, с ошибкой, etc)

✒️Названия должны быть понятны и соответствовать UI:

```
Primary
Secondary
Disabled
WithIcon
ErrorState
EmptyState
```

🛏️Все стори должны использовать args, а не вручную писать JSX

Плохо:

```
export const Example = () => <Button label="Hi" />;
```

Хорошо:

```
export const Example = {
  args: { label: "Hi" },
};
```

🌂Сложные состояния должны использовать decorators

- темы
- провайдеры
- контекст
- роутер

🌡️ Никакого сетевого API в сторис

Если компонент требует данные — замени их моками.

🎒Если компонент имеет важные edge cases — обязателен отдельный сторис

Например:

- ошибка
- большое количество данных
- пустое состояние

🎓Стори не должны содержать логики

Только отображение компонента.

🧱Расширенный шаблон (для сложных компонентов)

```
export const Loading: Story = {
  args: {
    loading: true,
  },
  parameters: {
    docs: {
      description: {
        story: "Показывает состояние загрузки кнопки.",
      },
    },
  },
};
```

🕐Глобальные декораторы (добавляются 1 раз в `.storybook/preview.ts`)

```
export const decorators = [
  (Story) => (
    <AppProviders>
      <Story />
    </AppProviders>
  ),
];
```

🧪Тестирование через Storybook

Каждый компонент должен иметь хотя бы один сторис, который используется для тестов (визуальных или интеграционных).

Пример:

```
Primary.play = async ({ canvasElement }) => {
  const canvas = within(canvasElement);
  await userEvent.click(canvas.getByRole("button"));
};
```

📌Когда писать сторис (кратко)

Пишем сторис для: - UI-компонентов (кнопки, инпуты, модалки) - компонентов со состояниями (loading, error, empty) - переиспользуемых и общих компонентов

Не пишем сторис для: - страниц - контейнерных компонентов с логикой - хуков и утилит

📌Чеклист перед пушем

- Компонент имеет минимум 2 стори (основной + альтернативный)
- Названия сторис понятные (Primary, Disabled, WithIcon...)
- Все примеры через `args`, без JSX-компонентов
- Нет реальных API-запросов, только мок-данные
- Edge-cases покрыты (empty, error, loading — если актуально)
- Декораторы используются только когда нужны
- Стори запускается без ошибок и выглядит корректно