



Разработка embedding SDK

Александр Полянкин



Кому это полезно

- В вашем продукте есть возможность кастомизации темы
- Вы разрабатываете точки расширения своего продукта сторонним кодом
- Код вашего приложения исполняется внутри стороннего приложения

Александр Полянкин

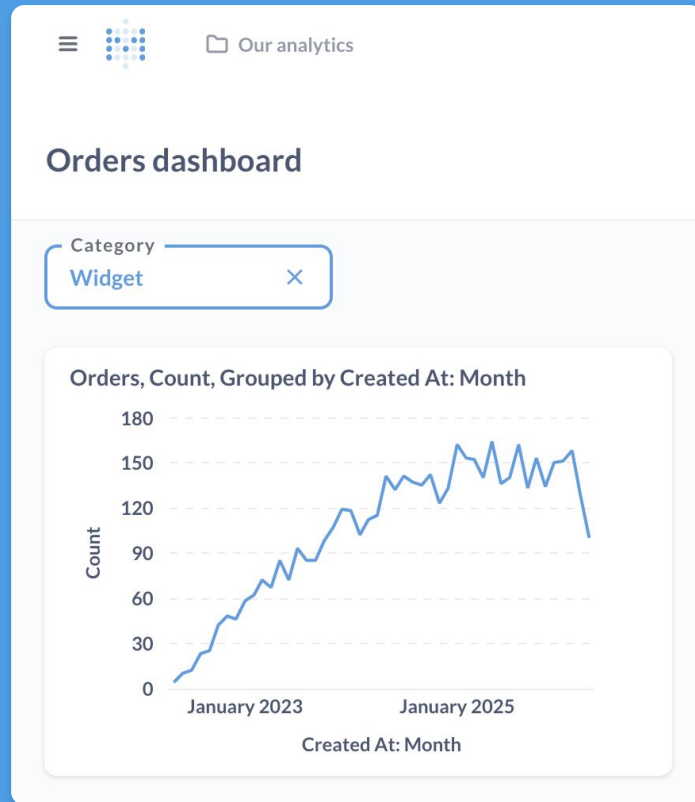
- Занимаюсь фронтендом последние 10 лет
- Metabase с 2021
- Wrike 2016-2020



Постановка задачи

Приложение

- TypeScript
- React
- CSS & Emotion
- Mantine
- Apache ECharts
- Jest & RTL, Cypress



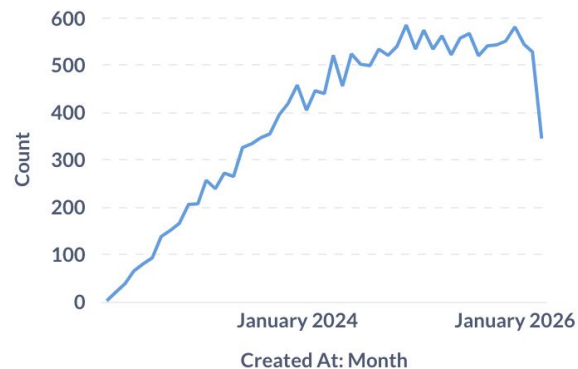
Embedding через iframe

- Отдельная страница, собранная из компонентов приложения
- Ограниченные возможности кастомизации

Orders dashboard

T Category ▾

Orders, Count, Grouped by Created At: Month



Требования

Inventory performance

Count of things

3

↓ 50% · vs. previous year: 6

Monthly Orders Trend

61

↑ 165.22% · vs. previous month: 23

Inventory performance

Count of things

8

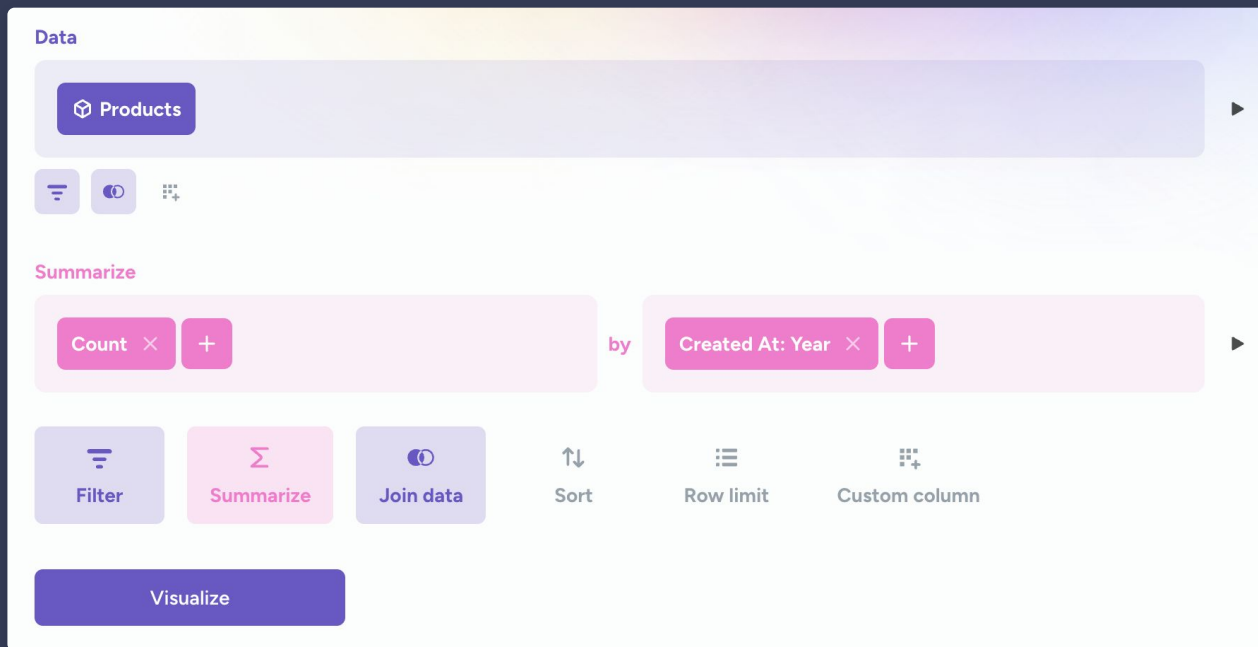
↑ 14.29% · vs. previous year: 7

Monthly Orders Trend

118

↑ 162.22% · vs. previous month: 45

Требования



Embedding SDK

- SDK на основе компонентов приложения
- Кастомизация внешнего вида и поведения



```
npm install @metabase/embedding-sdk-react
```

План

- 01 Стили
- 02 Whitelabel
- 03 Redux
- 04 Плагины
- 05 Тестирование

01

Стили

- Глобальные стили
- Emotion & CSP
- CSS переменные

Стили

Глобальные СТИЛИ

Глобальные стили

- Компоненты
- Утильные классы
- CSS reset

```
● ● ●  
  
.link {  
  color: var(--color-brand);  
}  
  
.flex-full {  
  flex: 1 0 auto;  
}
```

Миграция на CSS модули

- file.css - глобальные стили
- file.module.css - CSS модули
- Дублирование селектора с :local

```
.editor, :local(.editor) {  
  display: grid;  
}
```

CSS reset & :where()

- Обычный descendant selector повышает специфичность
- :where() имеет специфичность 0

```
:where(.mb-wrapper) {  
  h1, h2 {  
    font-weight: 700;  
  }  
}
```

Стили

Emotion & CSP

Emotion & CSP

- Emotion создает элементы style в рантайме и добавляет их в head
- Стандартная конфигурация Content Security Policy запрещает браузеру применять эти стили

```
  
<style>  
  .emotion-19ukec2 {  
    width:100%;  
  }  
</style>
```

Emotion & CSP

✘ ▶ Refused to apply inline style because it violates the following Content Security Policy directive: "style-src 'self' <http://localhost:8080> <http://localhost:9630> <https://accounts.google.com>". Either the 'unsafe-inline' keyword, a hash ('sha256-LRy5HaBYEKsjwKNt93QAQLyDR4eZqmD1oGlw8VHXH04='), or a nonce ('nonce-...') is required to enable inline execution.

Кратко о CSP

- HTTP заголовок Content-Security-Policy, который определяет, какие ресурсы браузер может загрузить
- Директивы default-src, script-src, style-src, frame-src, ...



```
default-src 'self'
```

Что не работает с Emotion

- 'self'
- https://example.com
- https:
- 'sha256-abcdef'



```
style-src 'self' 'sha256-cd9827ad...'
```

'unsafe-inline'



```
style-src 'self' 'unsafe-inline'
```

- Разрешает тег `<style>` и атрибут `style=""` без каких-либо проверок.
- Не требует конфигурации `emotion`
- Уязвимости

'nonce-<nonce_value>'

```
style-src 'self' 'nonce-416d1177...'
```

- Разрешает тег <style> с совпадающим атрибутом "nonce".
- "nonce" должен генерироваться для каждого запроса
- Поддерживается emotion из коробки

Стили

CSS переменные

CSS переменные

- :root
- .mb-wrapper



```
:root {  
  --color-brand: #123456;  
}
```

CSS переменные

- Уникальный префикс
- Линтер

```
:root {  
  --mb-color-brand: #123456;  
}
```

Выводы

01

Глобальные стили → CSS
модули

02

CSS reset → :where()

03

Emotion & CSP → 'nonce'

04

CSS-переменные → префикс

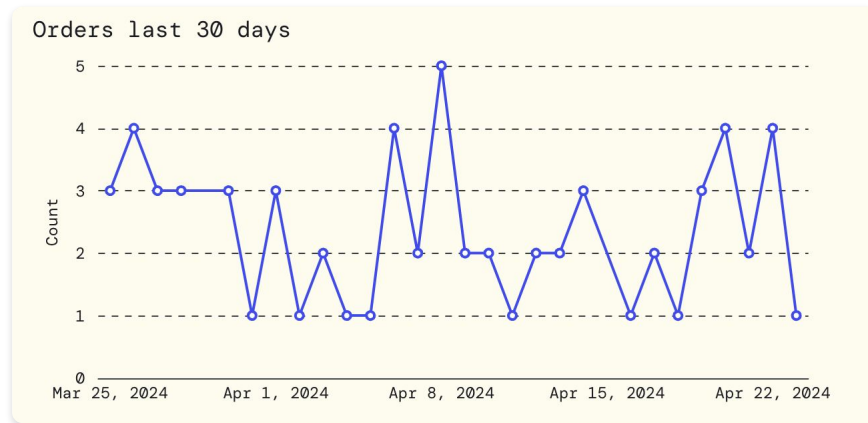
02

Whitelabel

- Цвета
- Шрифты
- Отступы и layout

Что такое whitelabeling

- Соответствие бренду другого продукта
- Лого, цветовая палитра, шрифты, отступы
- Локализация



Whitelabel

Цвета

Раньше

- Цвета объявлены в коде
- Функция color
- Функции alpha, lighten, darken



```
export const colors = {  
  brand: "#509EE3",  
};
```

Раньше



```
export function color(c: ColorName) {  
  return palette[c];  
}  
  
export function lighten(c: ColorName, f: number = 0.5) {  
  return Color(color(c)).lighten(f).string();  
};
```

Раньше

- emotion компоненты
- Цвета не изменить в рантайме

```
const Link = styled.div`  
  color: ${color("brand")};  
  
  &:hover {  
    color: ${lighten("brand")};  
  }  
`;  
;
```

CSS-переменные

- Тему приложения можно изменить на лету
- Требуется аналог функций alpha, lighten, darken



```
:root {  
  --mb-color-brand: "#509EE3";  
};
```

color-mix



```
/* alpha() */  
color-mix(in hsl, var(--mb-color-brand), transparent 50%);  
  
/* lighten() */  
color-mix(in hsl, var(--mb-color-brand), white 10%);  
  
/* darken() */  
color-mix(in hsl, var(--mb-color-brand), black 10%);
```

relative colors



```
/* alpha() */  
hsl(from var(--mb-color-brand) h s l / 0.8)  
  
/* lighten() */  
hsl(from var(--mb-color-brand) h s calc(l * 1.2))
```

Whitelabel

Шрифты

Шрифт в React SDK

- Работает с кастомными шрифтами



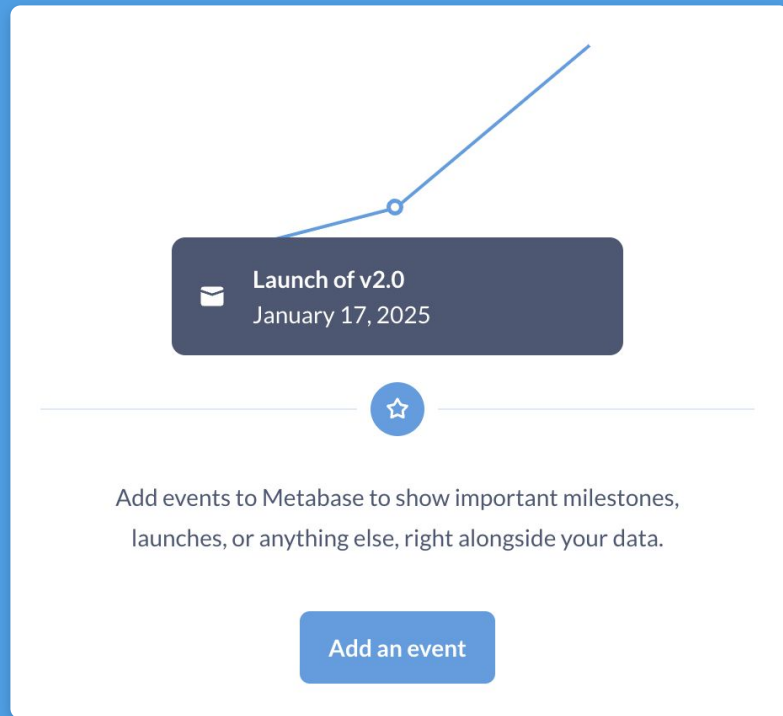
```
:root {  
  --mb-default-font-family: "Lato";  
};
```


Whitelabel

Строки и локализация

Строки и локализация

- Заменить имя приложения
- Должно работать с учетом локализации



The screenshot displays a dark blue event card with a white envelope icon, titled "Launch of v2.0" and dated "January 17, 2025". Below the card is a blue star icon on a white background. The text below the star reads: "Add events to Metabase to show important milestones, launches, or anything else, right alongside your data." At the bottom of the interface is a blue button labeled "Add an event".

ttag



```
import { t } from "ttag";

function TimelineEmptyState() {
  return (
    <Text>
      {t`Add events to Metabase`}
    </Text>
  );
}
```

ESLint плагин



```
~/metabase/frontend/src/TimelineEmptyState.tsx
```

```
13:18 error Metabase string must not be used directly.
```

```
Please import `getApplicationName` selector from `metabase/selectors/whitelabel`  
and use it to render the application name.
```

```
Or add comment to indicate the reason why this rule needs to be disabled.
```

getApplicationName

```
import { t } from "ttag";

function TimelineEmptyState() {
  const appName = useSelector(getApplicationName);

  return (
    <Text>
      {t`Add events to ${appName}`}
    </Text>
  );
}
```

Whitelabel

Layout

Темизация mantine

- mantine дает возможность задать отступы, border-radius и прочее на уровне темы
- SDK позволяет переопределить тему

```
function getThemeOverrides(): MantineThemeOverride {  
  return {  
    spacing: {  
      xs: rem(4),  
      sm: rem(8),  
      md: rem(16),  
      lg: rem(24),  
      xl: rem(32),  
    }  
  };  
}
```

Темизация mantine

- Использовать enum из темы вместо hardcoding
- Нестандартные значения через комбинацию стандартных

```
<Stack p="md" spacing="sm">
  {availableOptions.map(option => (
    <Radio
      key={option.value}
      value={option.value}
      label={option.name}
      size="xs"
    />
  ))}
</Stack>
```

Выводы

01

Цвета – CSS переменные,
color-mix, relative colors

02

Шрифты – CSS переменные,
генерация @font-face

03

Строки и локализация – ESLint
плагин

04

Layout – темизация mantine

03

Redux

- Проблемы
- RTK
- RTK-query

Redux

- 1 redux store
- Redux toolkit + RTK query
- Несколько библиотек, которые используют redux

Привязка к redux store

- UI state в redux
- Тест – отрендерить текущее приложение 2 раза в одном окне



Внешний redux store

- Конфликт между redux store внешнего приложения и SDK
- Добавляем свой контекст в RTK и RTK-query



```
const MetabaseReduxContext = createContext(null);  
  
const useSelector: TypedUseSelectorHook<State> =  
  createSelectorHook(MetabaseReduxContext);
```

Внешний redux store

- Конфликт между redux store внешнего приложения и SDK
- Добавляем свой контекст в RTK и RTK-query

```
const createApi = buildCreateApi(
  coreModule(),
  reactHooksModule({
    hooks: {
      useDispatch: createDispatchHook(MetabaseReduxContext),
      useSelector: createSelectorHook(MetabaseReduxContext),
      useStore: createStoreHook(MetabaseReduxContext),
    },
  }),
);
```

Выводы

01

Убираем
зависимость 1:1
между
компонентами и
redux store

02

Кастомный
контекст

03

Поддержка
кастомного redux
контекста
сторонними
библиотеками

04

Плагины

- Плагины Metabase
- Плагины SDK

Плагины

Плагины Metabase

Metabase OSS vs EE

- Бесплатная (OSS) и коммерческая (EE) версии
- Весь код в одном репозитории с разными файлами лицензий
- В OSS есть точки расширения для реализации платных фич

Metabase OSS vs EE

 Add to dashboard

 Turn into a model

 Move

 Duplicate

 Move to trash

 Add to dashboard

 Verify this question

 Turn into a model

 Edit settings

 Move

 Duplicate

 Move to trash

Объявление плагина

- Плагины определяются в коде основного продукта
- Во многих случаях реализация по умолчанию – пустой компонент, колбек и т.п.



```
export type ModerationPlugin = {  
  QuestionMenuItem: ComponentType<{ question: Question }>;  
};  
  
export const PLUGIN_MODERATION: ModerationPlugin = {  
  QuestionMenuItem: EmptyComponent,  
};
```

Использование плагина

- Реализация должна учитывать, что объект плагина может быть изменен

```
function QuestionMenu({ question }: QuestionMenuProps) {  
  return (  
    <Menu>  
      ...  
      <PLUGIN_MODERATION.QuestionMenuItem question={question} />  
      ...  
    </Menu>  
  );  
}
```

Переопределение плагина

- Код EE переопределяет плагина в зависимости от активированных фич
- Этот код должен быть выполнен до кода основного приложения

```
if (hasPremiumFeature("content_verification")) {  
  Object.assign(PLUGIN_MODERATION, {  
    QuestionMenuItem,  
  });  
}
```

Плагины и сборка

- В основном приложении есть импорт-placeholder
- В зависимости от версии приложения, импорт подменяется на путь к модулю с EE плагинами

```
const config = {
  resolve: {
    alias: {
      "ee-plugins":
        process.env.MB_EDITION === "ee"
          ? ENTERPRISE_SRC_PATH + "/plugins"
          : SRC_PATH + "/lib/noop",
    },
  },
};
```

Плагины и сборка

- Возможность инициализировать плагин один раз до импорта основного кода
- Плагины для внутреннего использования для коммерческой версии продукта

Плагины – EE & SDK

- SDK – другой вариант собрать продукт с такой же системой плагинов
- SDK использует плагины для базовой функциональности, например, авторизации

```
type ApiPlugin = {  
  onBeforeRequest: () => void;  
};  
  
export const PLUGIN_API: ApiPlugin = {  
  onBeforeRequest: emptyCallback,  
}
```

Плагины

Плагины SDK

Плагины SDK

- В SDK есть своя система плагинов, не связанная с продуктом
- Плагины SDK основываются на React-контексте

```
<MetabaseProvider
  plugins={{
    dashboard: {
      DashboardCardMenuItems,
    }
  }}
/>
```

Плагины SDK

- Для поддержки плагинов SDK в плагины приложения добавляются хуки

```
type SdkPlugin = {  
  useDashboardPlugin: (): DashboardSdkPlugin | undefined,  
}  
  
const PLUGIN_SDK: SdkPlugin = {  
  useDashboardPlugin: createEmptyHook(),  
};
```

Плагины SDK

- Хуки позволяют мержить настройки с общего провайдера и отдельных корневых компонентов SDK



```
export function useDashboardPlugin() {  
  const context = useContext(MetabaseContext);  
  return context.plugins.dashboard;  
}
```

Выводы

01

Плагины позволяют модифицировать приложение под разные варианты сборки

02

Хуки в плагинах позволяют получать доступ к контексту глобального провайдера SDK

05

Тестирование

- Тестирование плагинов
- Визуальное тестирование
- E2E тестирование

Тестирование

Тестирование плагинов

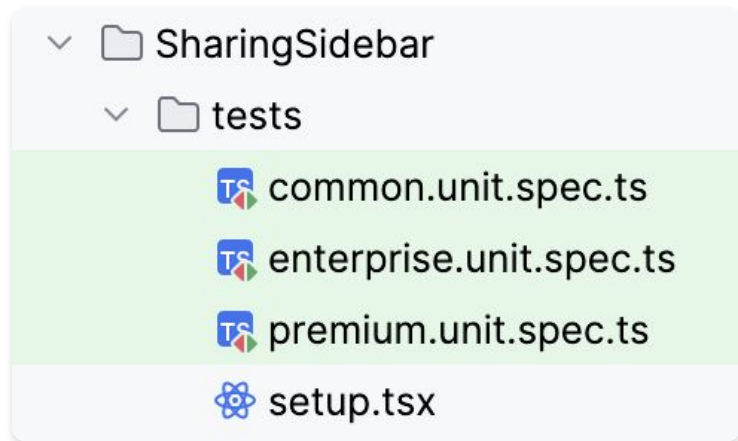
Тестирование плагинов с Jest

- Импорт EE плагинов
- Порядок импортов

```
function setupEnterprisePlugins() {  
  require("metabase-enterprise/plugins");  
}  
  
beforeEach(() => {  
  setupEnterprisePlugins();  
});
```

Тестирование плагинов с Jest

- Выгрузка модуля
- Отдельные файлы для тестирования разных комбинаций плагинов
- Тестирование платной версии без активированных фич



Тестирование

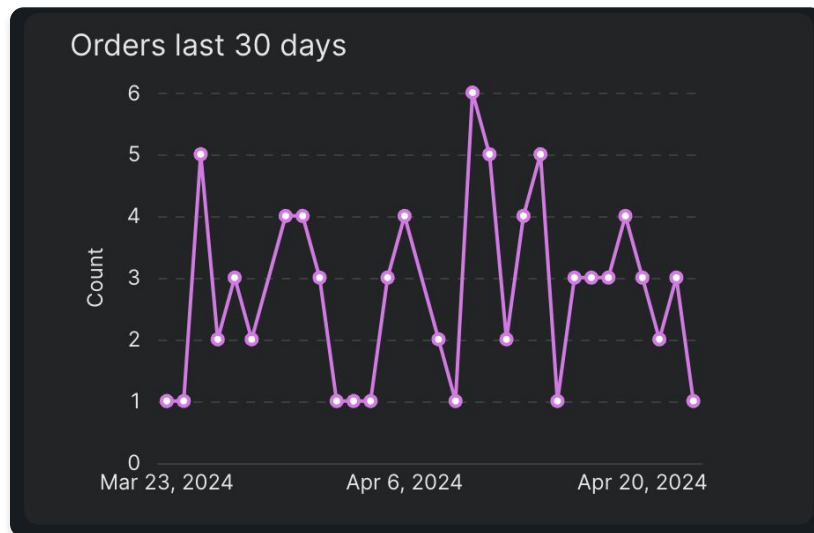
Визуальное тестирование

Тестирование кастомизации

- Цвета, шрифты, отступы
- Расчетные значения

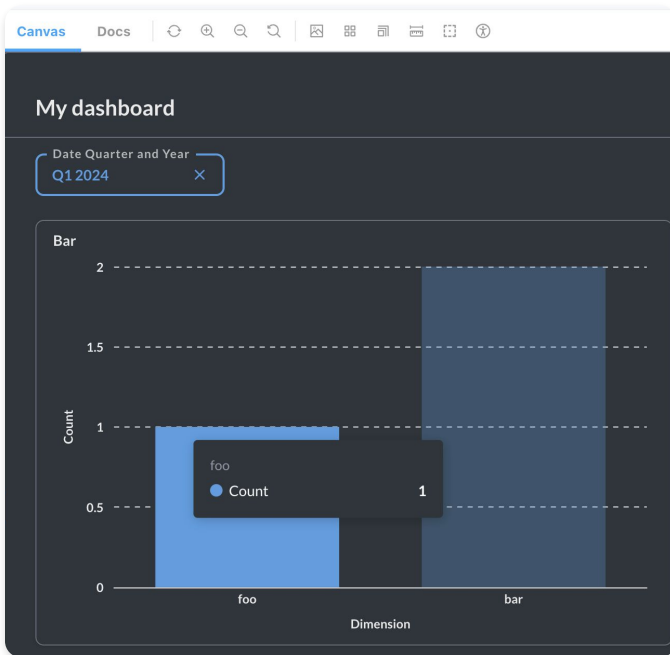
Темная тема

- Семантические цвета – темный текст vs «основной» текст
- Может потребовать дополнительной кастомизации
- Контрастность



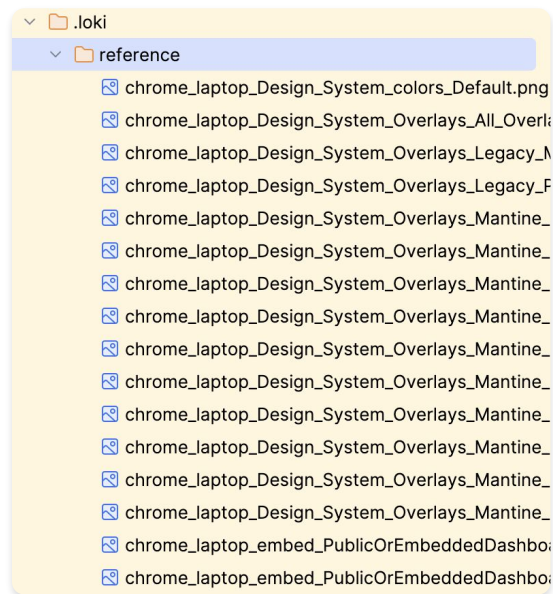
Storybook

- Рендерим основные компоненты SDK с разными темами
- `play()` позволяет сэмулировать действие пользователя



Storybook + Loki

- Loki запускает Chrome в Docker и делает скриншоты
- Требуется самим хранить референсные скриншоты

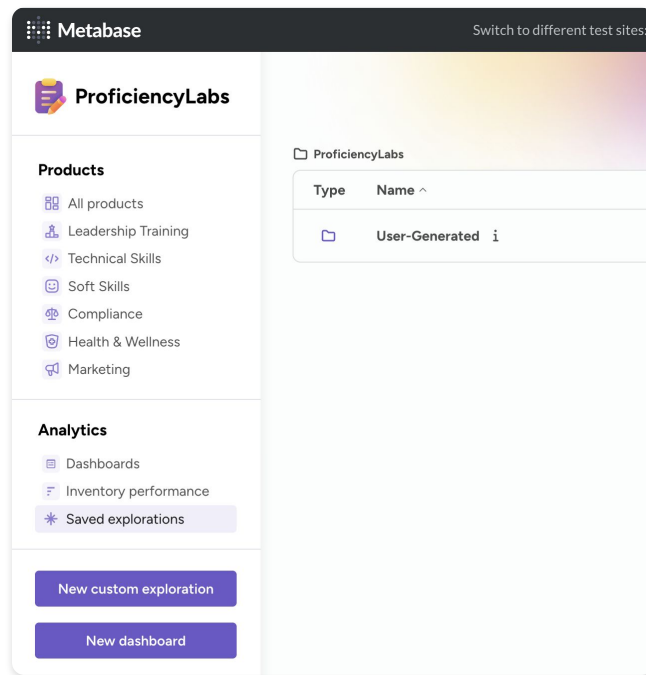


Тестирование

E2E тестирование

Cypress

- Несколько разных демо-приложений на SDK
- Тестируем основные пользовательские сценарии



Cypress component testing

- Монтируется корневой компонент SDK
- `cy.intercept()` для мокирования авторизации

```
cy.mount(  
  <MetabaseProvider  
    {...sdkProviderProps}  
    authConfig={{  
      ...DEFAULT_SDK_AUTH_PROVIDER_CONFIG,  
      ...sdkProviderProps?.authConfig,  
    }}  
  >  
    {children}  
  </MetabaseProvider>,  
);
```

Cypress cross-version tests

- Версия SDK может отличаться от версии бекенда
- Запускаем прошлые версии продукта (бекенда для SDK) в Docker
- Policy по обратной совместимости

Выводы

01

Тестирование
реальных
плагинов, не моков

02

Визуальные
регрессии

03

E2E тестирование

Спасибо!