

Требования к разработке компонентов

Автор последнего обновления: Pavel.Tokarenko@evraz.com | 24 авг. 2023 г. в 22:34 GMT+3

Виды компонентов

- **UI Kit** — абстрактные компоненты, которые будут переиспользоваться другими модулями. Предметная область - UI.
- **Components** — абстрактные компоненты, которые специфичны для текущего модуля, гипотетически могут стать компонентами *UI Kit*. Предметная область - UI.
- **Containers** — компоненты, которые взаимодействуют с данными. Как правило привязаны к предметной области модуля. Проектная предметная область.
- **Pages** — компоненты, которые используются в качестве экрана. Проектная предметная область.

Структура компонента

Данная структура справедлива для всех компонентов.

```
<ComponentName>
  index.ts                (1)
  <ComponentName>.tsx     (2)
  <ComponentName>.module.css (3)
  types.ts               (4)
  utils.ts               (5)
  <ComponentFragmentName>.tsx (6)
  <PageContainerName>.tsx  (7)
```

1. `index.ts` используется для экспорта основного компонента. Позволяет убрать дублирование имён при импорте компонента.

```
// без index.ts
import Component from "../Components/Component/Component";

// с index.ts
import Component from "../Components/Component";
```

Содержимое `index.ts`

```
import Component from "../Component";

export default Component;
```

Важно! НЕЛЬЗЯ вести разработку компонента в `index.ts`.

2. Основной компонент. Название должно совпадать с названием папки в которой находится компонент. Расширение файла обязательно должно быть `.tsx`
3. Обязательно использование `css modules`. Название модуля должно совпадать с названием директории в которой находится таблица стилей. Применяется для обеспечения инкапсуляции компонента.
4. **Типы данных** используемые с компонентом

5. Все вспомогательные функции, используемые в компоненте, **ДОЛЖНЫ** выноситься в данный файл.
6. Если используются вспомогательные компоненты, все выносятся в отдельный компонент, размещаются на том же уровне, что и основной компонент. Используют ту же таблицу стилей.
7. Только для компонентов страниц, если необходим. Контейнер хранится в том же месте, где и реализация компонента.

Имена компонентов

Имена компонентов задаются в формате `PascalCase`.

```
# плохо
componentName
component-name
Componentname

# хорошо
ComponentName
```

Имя функции компонента должно совпадать с именем файла.

`ComponentName.tsx`

```
// ...
function ComponentName(/* props */) {
  // ...
}
```

Выбор имени

Имя компонента зависит от **вида компонента**. Для абстрактных компонентов (*UI Kit, Components*) имя задаётся в отрыве от данных, полагаясь только на внешний вид компонента. Для остальных опираясь на назначение в проекте.

Пример. На макете изображен элемент, который выводит данные по плавкам с фотографиями:

Плохо

В папке `Components` создаётся компонент `MeltsInfoGrid`. Компонент подключает данные, производит преобразования данных.

Хорошо

В папке `Components` или `UI Kit` создаётся компонент `ImagesWithDescriptionsGrid`. В папке `Containers` создаётся компонент `MeltsInfoGrid`, который получает данные, преобразует их для `ImagesWithDescriptionsGrid` затем выведет `ImagesWithDescriptionsGrid`

Порядок импортов

Данный порядок задаётся плагином к `ES Lint`, автоматизируется за счёт плагинов к IDE `ES Lint` [↗](#), `Prettier` [↗](#).

Внимание! Конфигурация линтеров стандартизирована и входит в стартовый шаблон проекта. Использование альтернативной конфигурации возможно только после согласования с лидером направления.

Типизация компонента

Типы компонента описываются в файле `types.ts` и ДОЛЖНЫ экспортироваться, если используются в интерфейсе компонента.

Для `props` компонента, как минимум, ДОЛЖЕН быть разработан интерфейс следующего содержания:

```
export interface ComponentNameProps {
  className?: string;
  style?: React.CSSProperties;
}
```


Сам компонент в обязательном порядке ДОЛЖЕН использовать атрибуты указанные выше. Пример:

```
function ComponentName({ className, style }: ComponentNameProps) {
  return (
    <div className={clsx(styles.root, className)} style={style}>
      {content}
    </div>
  );
}
```

Примечание. `clsx` модуль, отвечающий за конкатенацию имён CSS классов

НЕЛЬЗЯ использовать стилизующие атрибуты, кроме `style`. Например: `margin`, `size`, `height`, `width` и т.п.

Стили компонента

В компоненте ДОЛЖНЫ применяться [CSS Modules](#) . Конечное название стиля имеет следующую схему:


`[ИмяКомпонента]_[локальное_имя_класса]__[хЭш]`

Для того чтобы отличать названия атрибутов объектов от названий CSS классов, для последних применяется `snake_case` нотация. Корневой элемент компонента ДОЛЖЕН содержать класс `.root`.

БЭМ методология МОЖЕТ применяться частично. Не нужно указывать название блока, используется только название элемента. Модификатор отделяется от элемента двойным символом подчёркивания. Например: `styles.pane`, `styles.pane__visibly`

НЕЛЬЗЯ использовать значения в названиях классов. Например: `.margin_20`, `.size_500`.

Стилизация сложных компонентов

Компоненты, имеющие сложную структуру, ДОЛЖНЫ быть стилизованы с использованием [CSS Custom properties](#) . Названия CSS Custom properties ДОЛЖНЫ задаваться следующим шаблоном:

`--<имя-элемента>-<название-стиля>-<модификатор>`

Например:

```
function Switcher({ className, style, checked, onChange }: SwitcherProps) {  
  // Логика...  
  return (  
    <label className={clsx(styles.root, className)} style={style}>  
      <input  
        type="checkbox"  
        className={styles.input}  
        checked={checked}  
        onChange={onChange}  
      />  
      <div className={styles.marker} />  
    </label>  
  );  
}
```

```
.root {  
  --marker-background-color: #123;  
  --marker-border: none;  
  --marker-background-color-active: #321;  
  /* Остальные стили */  
}  
  
.marker {  
  background-color: var(--marker-background-color);  
  border: var(--marker-border);  
}  
  
.input:checked + .marker {  
  background-color: var(--marker-background-color-active);  
}
```