TypeScript convention

Component

> **Public components** are considered all components exported from `@mui/material` or `@mui/lab`.
>
> **Internal components** are considered all components that are not exported from the packages, but only used in some public component.

`Props Interface`

- export interface `{ComponentName}classes` from `{component}Classes.ts` and add comment for generating api docs (for internal components, may or may not expose classes but don't need comment)
- export interface `{ComponentName}Props`
- always export props interface (use `interface` over `type`) from the component file

Public component

```tsx
// fooClasses.tsx

export interface FooClasses {
  /** Styles applied to the root element. */
  root: string;
  /** Styles applied to the foo element. */
  foo: string;
  /** Styles applied to the root element if `disabled=true`. */
  disabled: string;
}

const fooClasses: FooClasses = generateUtilityClasses('MuiFoo', ['root', 'foo', 'disabled']);

export default fooClasses;
```

```tsx
// Foo.tsx
import { FooClasses } from './fooClasses';

export interface FooProps {
  /**
   * Override or extend the styles applied to the component.
   */
  classes?: Partial<FooClasses>;
  // ...other props
  /**
   * The system prop that allows defining system overrides as well as additional CSS styles
   */
```

```
  sx?: SxProps<Theme>;
}
```

internal component

```
// Bar.tsx

// if this internal component can accept classes as prop
export interface BarClasses {
  root: string;
}

export interface BarProps {
  classes?: Partial<BarClasses>;
  sx?: SxProps<Theme>;
}
```

ClassKey

- naming as `{ComponentName}ClassKey`
- export if `classes` exists in props interface using `keyof` from `{component}Classes.ts`

```
// fooClasses.ts
export interface FooClasses {
  ...
}

export type FooClassKey = keyof FooClasses;
// verify that FooClassKey is union of string literal
```

Classes generator & Utility

- export if `classes` exists in props interface from the component file
- use `{Component}Classes` as type to preventing typo and missing classes
- use `Private` prefix for internal component

Public component

```
// fooClasses.ts
export function getFooUtilityClass(slot: string) {
  return generateUtilityClass('MuiFoo', slot);
}

const useUtilityClasses = (ownerState: FooProps & { extraProp: boolean }) => {
  // extraProp might be the key/value from react context that this component access
  const { foo, disabled, classes } = ownerState;

  const slots = {
```

```
    root: ['root', foo && 'foo', disabled && 'disabled'],
  };

  return composeClasses(slots, getFooUtilityClass, classes);
};
```

internal component

```
// Bar.tsx
// in case that classes is not exposed.
// `classes` is used internally in this component
const classes = generateUtilityClasses('PrivateBar', ['root', 'bar']);
```

StyledComponent

- naming using slot `{ComponentName}{Slot}`
- to extend interface of the styled component, pass argument to generic

public component

```
const FooRoot = styled(Typography, {
  name: 'MuiFoo',
  slot: 'Root',
  overridesResolver: (props, styles) => styles.root,
})({
  // styling
});
```

internal component

```
const BarRoot = styled(Typography)({
  // styling
});
```

extends interface

```
const BarRoot = styled(Typography)<{
  component?: React.ElementType;
  ownerState: BarProps;
}>(({ theme, ownerState }) => ({
  // styling
}));
// passing `component` to BarRoot is safe and we don't forget to pass ownerState
// <BarRoot component="span" ownerState={ownerState} />
```

Component declaration

- prefer `function Component() {}` over `React.FC`
- naming the render function in `React.forwardRef` (for devtools)
- `useThemeProps` is needed only for public component

- pass `ownerState` to StyledComponent for styling

public component

```tsx
const Foo = React.forwardRef<HTMLSpanElement, FooProps>(function Foo(inProps, ref) => {
  // pass args like this, otherwise will get error about theme at return section
  const props = useThemeProps<Theme, FooProps, 'MuiFoo'>({
    props: inProps,
    name: 'MuiFoo',
  });
  const { children, className, ...other } = props

  // ...implementation

  const ownerState = { ...props, ...otherValue }

  const classes = useUtilityClasses(ownerState);

  return (
    <FooRoot
      ref={ref}
      className={clsx(classes.root, className)}
      ownerState={ownerState}
      {...other}
    >
      {children}
    </FooRoot>
  )
})
```

internal component

```tsx
const classes = generateUtilityClasses('PrivateBar', ['selected']);

const BarRoot = styled('div')(({ theme }) => ({
  [`&.${classes.selected}`]: {
    color: theme.palette.text.primary,
  },
}));

// if this component does not need React.forwardRef, don't use React.FC
const Bar = (props: BarProps) => {
  const { className, selected, ...other } = props;
  return <BarRoot className={clsx({ [classes.selected]: selected })} {...other} />;
};
```