Strongly-typed CSS with Vanilla-extract


Modern TS Web App
CSS

Today

- 🔍🤔
- 😍🤢
- 💭🥧
- 🧁✅

Hello!

 Georgia Loper

 @developerLoper

🏢 Large enterprises

🛖 Tiny startups

📈 Quickly scaling companies

🦕 Legacy systems

⚓ Static landing pages

🧮 SPAs

💪 Web applications

🗽 Patina Design System @ Upstart


SCAN
ME
Slides 👆


Hello, StirTrek!

Markup

```
<div>
  <h1>🧁 Hello, StirTrek!</h1>
</div>
```

Styles

Static

```
{
  background: "navy";
  color: "plum";
  padding: 1rem;
}
```

Parent component

Static

```
// App.tsx
<Heading>
  🧁 Hello, StirTrek!
</Heading>
```

Parent component

Dynamic

```
// App.tsx
<Heading background="secondary" color="primary" padding="m" >
  🧁 Hello, StirTrek!
</Heading>
```

Styles

Themed

```
{
  background: var(--color-primary);
  color: var(--color-secondary);
  padding: var(--space-small);
}

:root {
  --color-primary: "navy";
  --color-secondary: "plum";
  --space-small: "1rem";
}
```

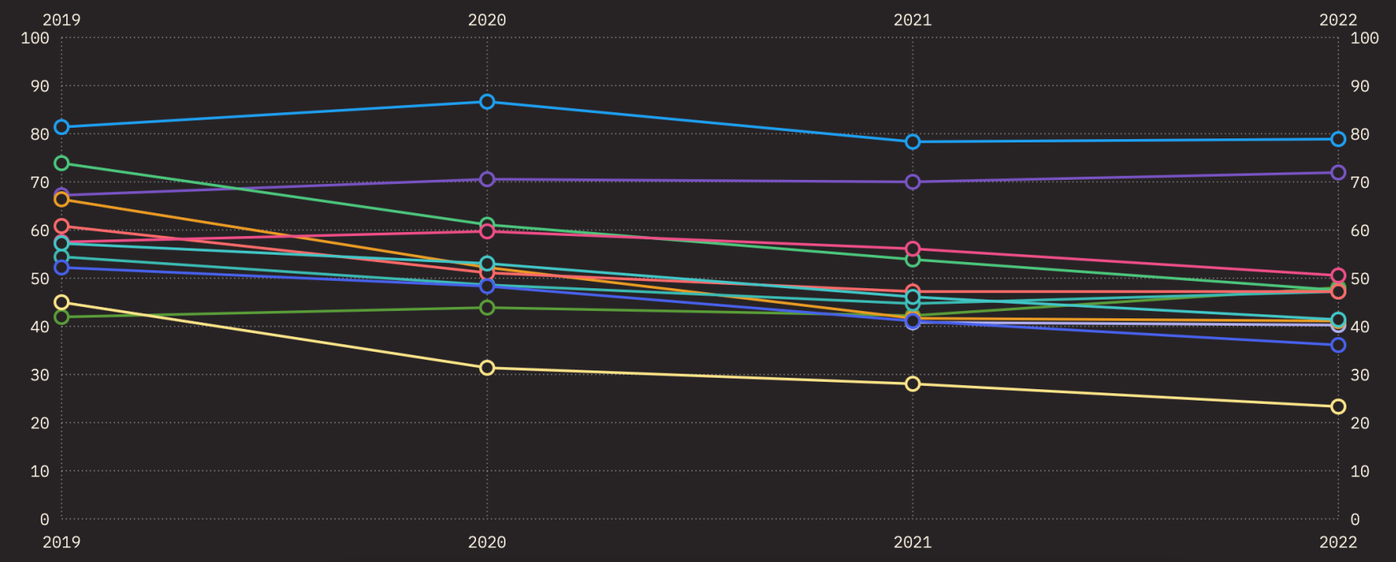
CSS FRAMEWORKS

Libraries that give you pre-made components and styles.

RATIOS OVER TIME

Percentages | Rankings

Retention, interest, usage, and awareness ratio over time.



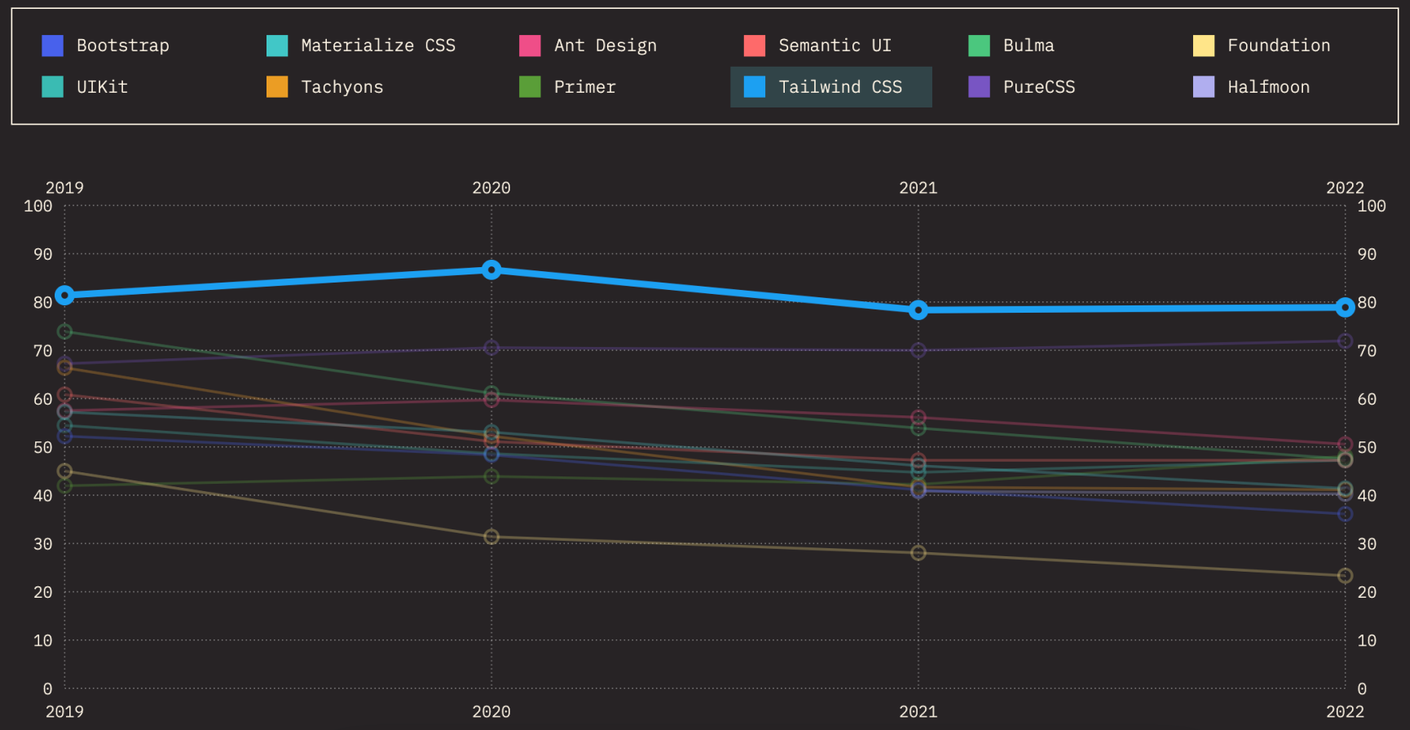
CSS FRAMEWORKS

Libraries that give you pre-made components and styles.

RATIOS OVER TIME

Percentages | Rankings

Retention, interest, usage, and awareness ratio over time.

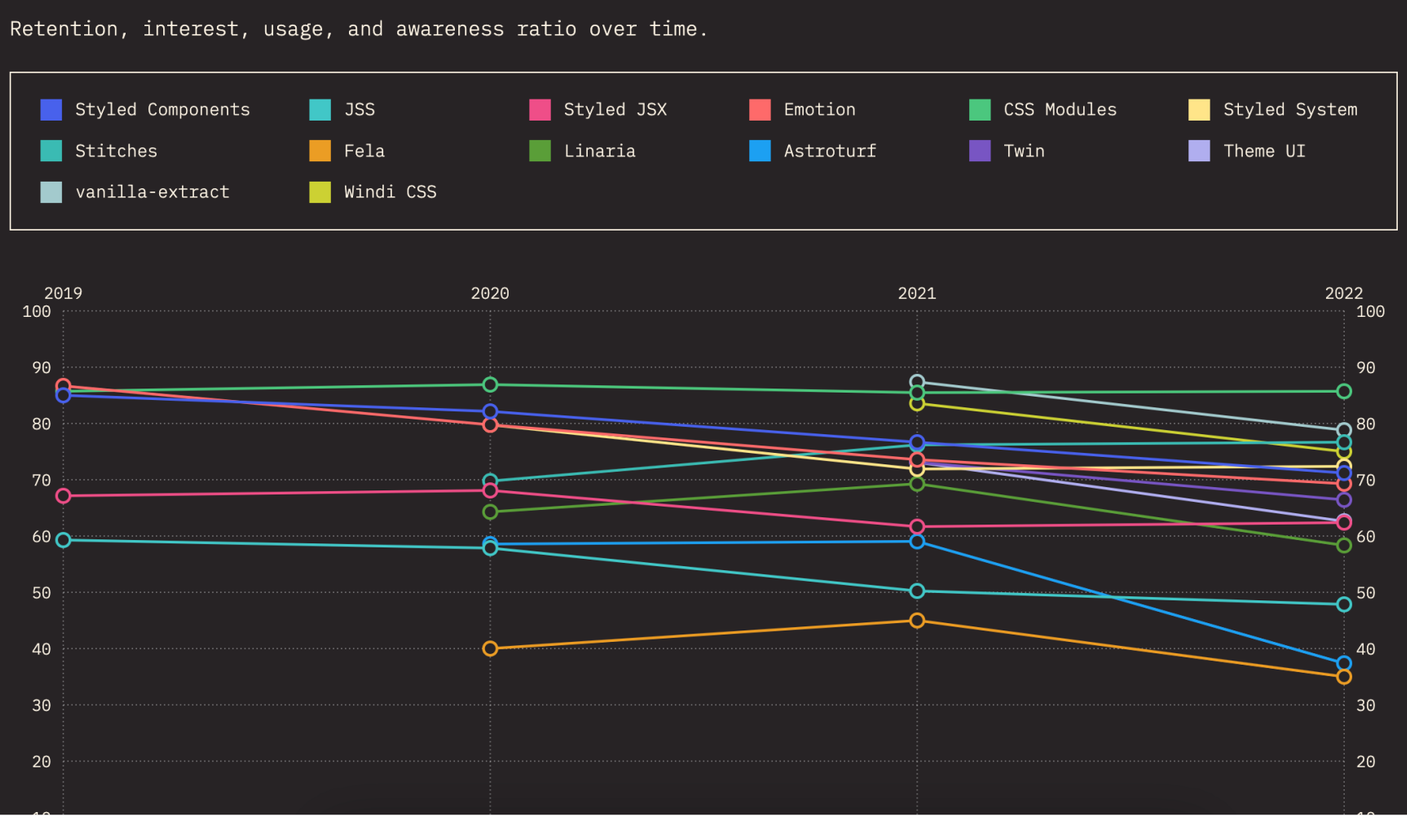


CSS-IN-JS

Libraries that help integrate CSS into JavaScript code.

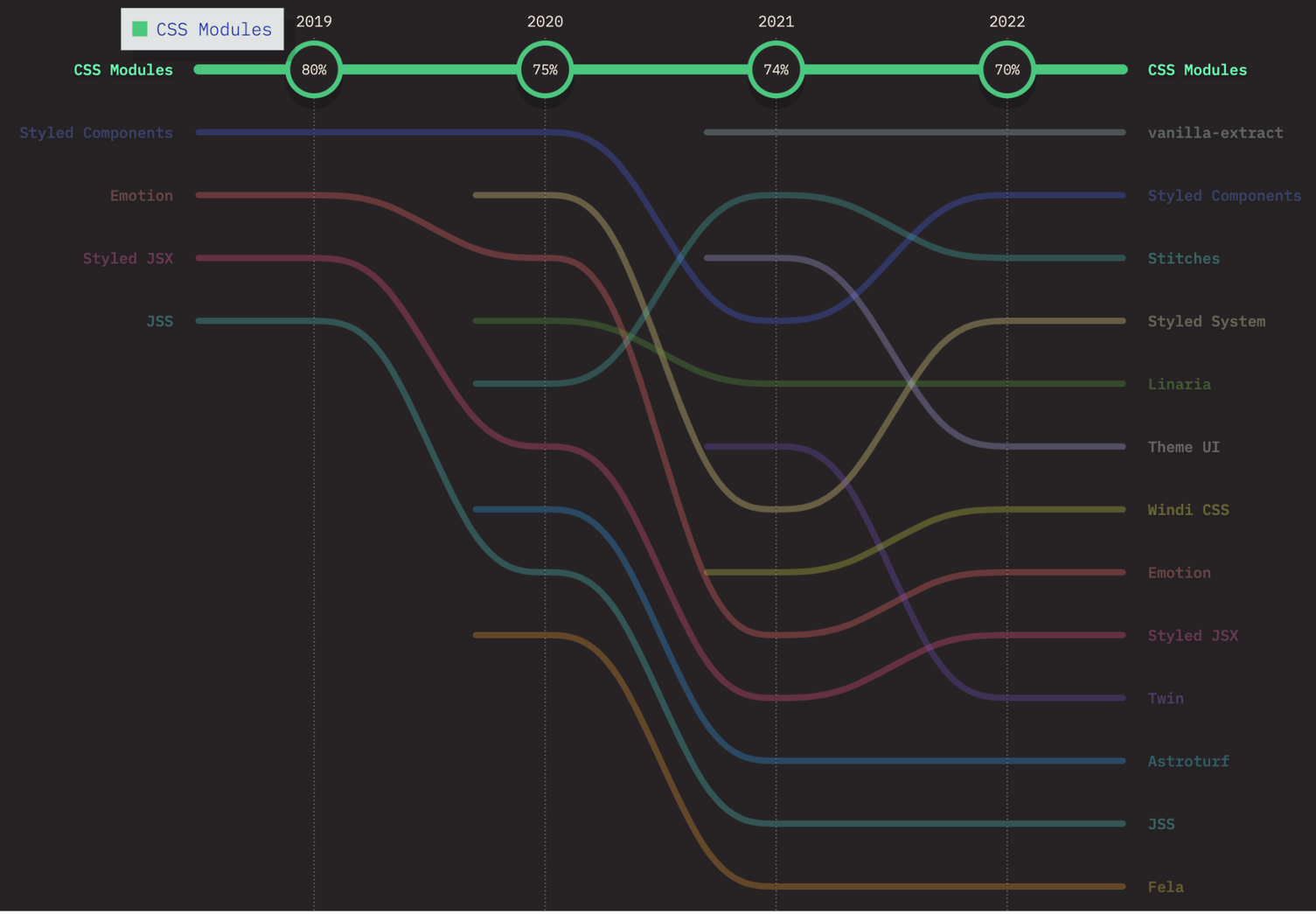
RATIOS OVER TIME

Retention, interest, usage, and awareness ratio over time.



RANKINGS OVER TIME

Retention, interest, usage, and awareness ratio rankings.



Runtime CSS-in-JS

Runtime CSS-in-JS

- Author styles in JS or TS
- Styles interpreted and applied at runtime

Runtime CSS-in-JS

```
// @emotion/react (css prop), with object styles
export const Heading = ({ children }) =>(
  <div
    css={{
      background: "navy",
      color: "plum",
      padding: "1rem"
    }}
  >
    <h1>{ children }</h1>
  </div>
)
```

Runtime CSS-in-JS

```
export const Heading = ({ background, color, padding, children }) => (
  <div
    css={{
      background,
      color,
      padding
    }}
  >
    <h1>{ children }</h1>
  </div>
)
```

```
.css-asdfgh {
  background: "navy";
  color: "plum";
  space: "1rem";
}
```

Runtime CSS-in-JS

- DevX
- Co-location
- Locally scoped
- *Can* be made typesafe

Runtime CSS-in-JS

- Runtime overhead
- SSR complexity
- Bundle size

Sam Magura

Posted on Oct 16, 2022 • Updated on Oct 25, 2022

Why We're Breaking Up with CSS-in-JS

#javascript #react #css #typescript

Hi, I'm Sam — software engineer at Spot and the 2nd most active maintainer of Emotion, a widely-popular CSS-in-JS library for React. This post will delve into what originally attracted me to CSS-in-JS, and why I (along with the rest of the Spot team) have decided to shift away from it.

We'll start with an overview of CSS-in-JS and give an overview of its pros & cons. Then, we'll do a deep dive into the performance issues that CSS-in-JS caused at Spot and how you can avoid them.

What is CSS-in-JS?

Atomic CSS

Atomic CSS

- ~1:1 class to CSS property
- Classes are composed together in markup

Atomic CSS

```
export const Heading = ({ children }) ⇒ (
  <div className="bg-primary text-secondary p-4">
    <h1>{ children}</h1>
  </div>
)
```

Atomic CSS

```
.bg-primary {
  background: "navy";
}

.text-secondary {
  color: "plum";
}

.p-4 {
  padding: 1rem;
}
```

Atomic CSS

```css
.bg-primary {
  background: var(--color-primary);
}

.text-secondary {
  color: var(--color-secondary);
}

.p-4 {
  padding: 1rem;
}

:root {
  --color-primary: "navy";
  --color-secondary: "plum";
}
```

Atomic CSS

```
import cx from 'classnames'

export const Heading = ({ background, color, space, children }) => (
  <div
    className={cx(
      backgroundMap[background],
      colorMap[color],
      spaceMap[space]
    )}
  >
    <h1>{ children }</h1>
  </div>
)
```

Atomic CSS

```
const backgroundMap = {
  primary: "bg-primary",
  secondary: "bg-secondary",
  tertiary: "bg-tertiary",
   ...
}

const colorMap = {
   ...
}

const spaceMap = {
   ...
}
```

Atomic CSS

- Finite CSS
- Static Stylesheet
- Bundle size

Atomic CSS

- Global scope
- Mental overhead
- DevX

CSS Modules

CSS Modules

- CSS-like syntax
- Build time CSS-in-JS

CSS Modules

```scss
/* Heading.module.scss */
.root {
  background: "navy";
  color: "plum";
  padding: 1rem;
}
```

CSS Modules

```tsx
// Heading.tsx
import styles from './Heading.module.css'

export const Heading = ({ children }) => (
  <div className={ styles.root }>
    <h1>{ className }</h1>
  </div>
)
```

CSS Modules

```
.Heading-root-cnq0T {
  background: "navy";
  color: "plum";
  padding: 1rem;
}
```

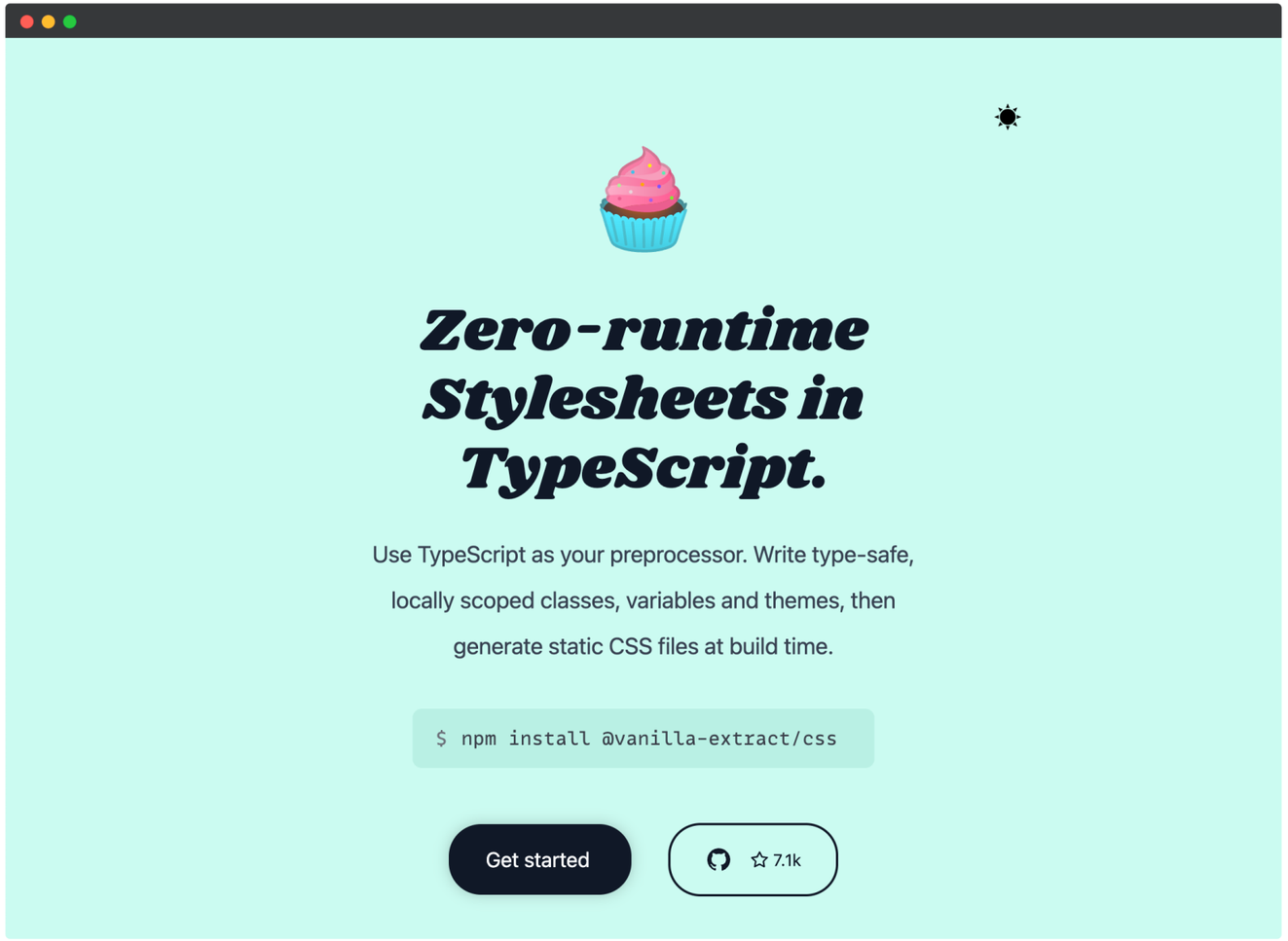
CSS Modules

- Locally scoped
- Explicit dependencies
- Familiar syntax

CSS Modules

- DevX
- Runtime theming
- Typesafety

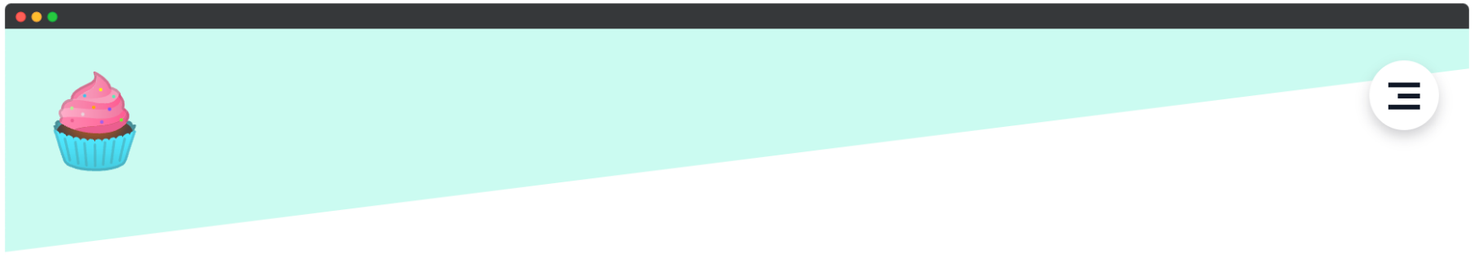
- Performance
- DevX
- Scoped styles
- Typesafe
- Runtime theming
- Atomic styles



vanilla-extract.style

Integrations

- Vite
- esbuild
- Webpack
- Next.js
- Rollup
- Gatsby



style

Creates a style rule with a locally scoped class name.

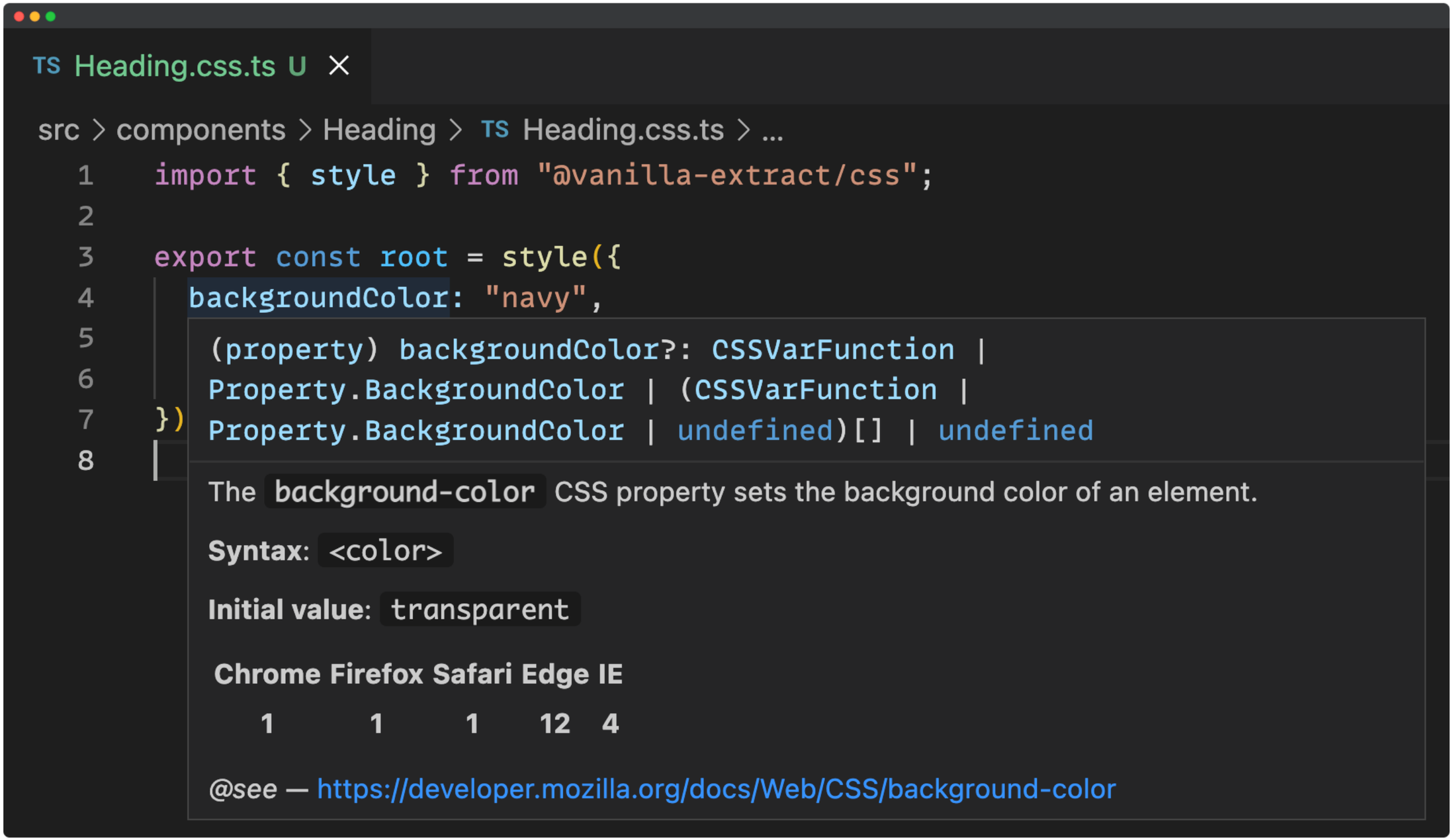
> 🖌️ New to styling with vanilla-extract? Make sure you've read the styling overview first.

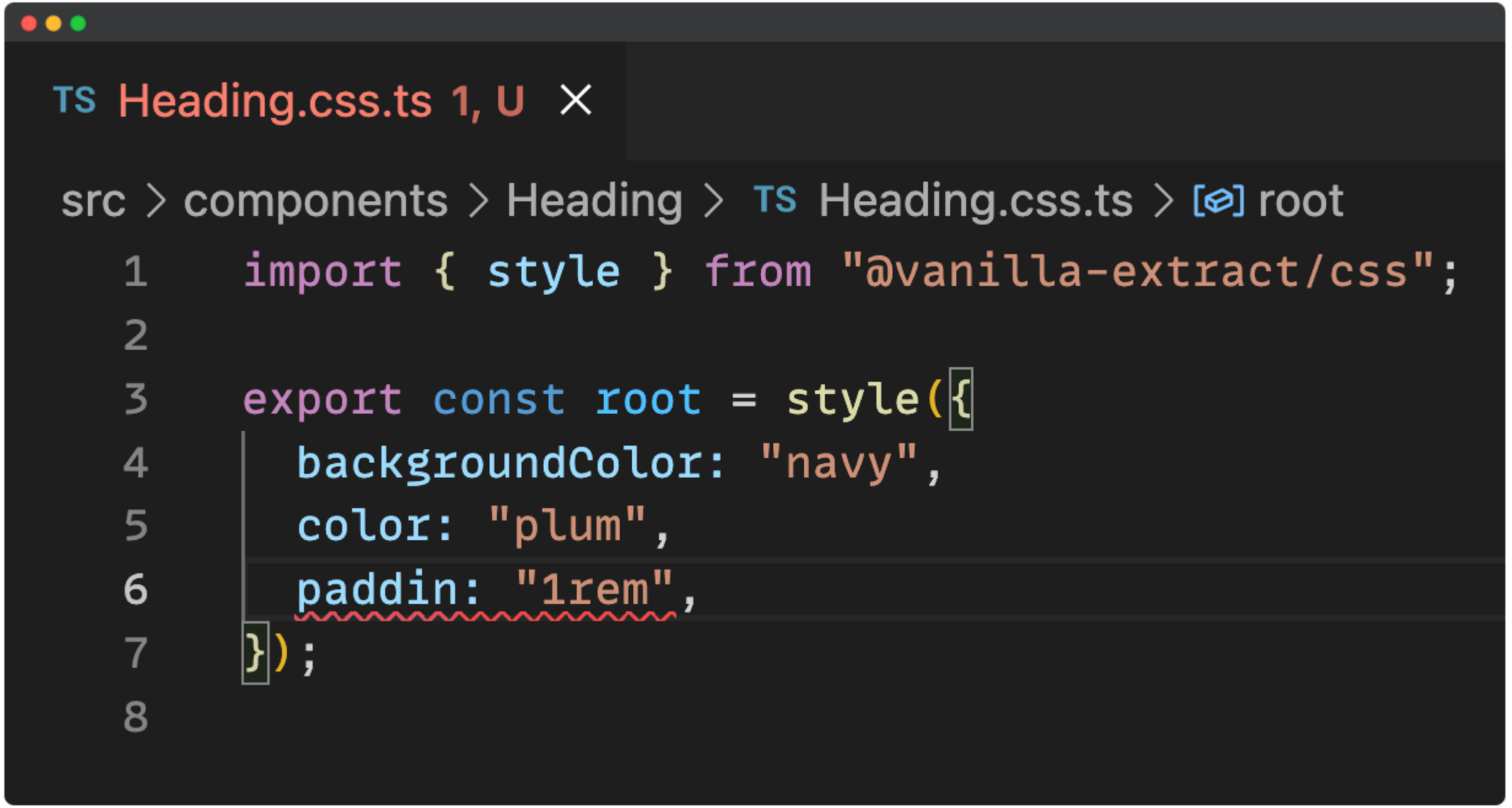
This class can then be imported directly into your consuming application code, creating a first-class contract between your CSS and JavaScript.

#style

VE style function

```
// ./components/Heading.css.ts
import { style } from '@vanilla-extract/css'

export const root = style({
  background: "navy",
  color: "plum",
  padding: "1rem"
})
```


TS Heading.css.ts U
src > components > Heading > TS Heading.css.ts > ...
1 import { style } from "@vanilla-extract/css";
2
3 export const root = style({
4 backgroundColor: "navy",
5
6
7 })
8
(property) backgroundColor?: CSSVarFunction | Property.BackgroundColor | (CSSVarFunction | Property.BackgroundColor | undefined)[] | undefined
The background-color CSS property sets the background color of an element.
Syntax: <color>
Initial value: transparent
Chrome Firefox Safari Edge IE
1 1 1 12 4
@see — https://developer.mozilla.org/docs/Web/CSS/background-color


TS Heading.css.ts 1, U
src > components > Heading > TS Heading.css.ts > root
1 import { style } from "@vanilla-extract/css";
2
3 export const root = style({
4 backgroundColor: "navy",
5 color: "plum",
6 paddin: "1rem",
7 });
8

```tsx
// ./components/Heading.tsx
import React from 'react'

export const Heading = ({ children }) => (
  <div>
    <h1>{ children }</h1>
  </div>
)
```

```tsx
// ./components/Heading.tsx
import React from 'react'
import styles from './Heading.css'

export const Heading = ({ children }) => (
  <div className={ styles.root }>
    <h1>{ children }</h1>
  </div>
)
```

```tsx
// ./components/Heading.tsx
import React from 'react'
import { root } from './Heading.css'

export const Heading = ({ children }) => (
  <div className={ root }>
    <h1>{ children }</h1>
  </div>
)
```


Hello, StirTrek!

Dev

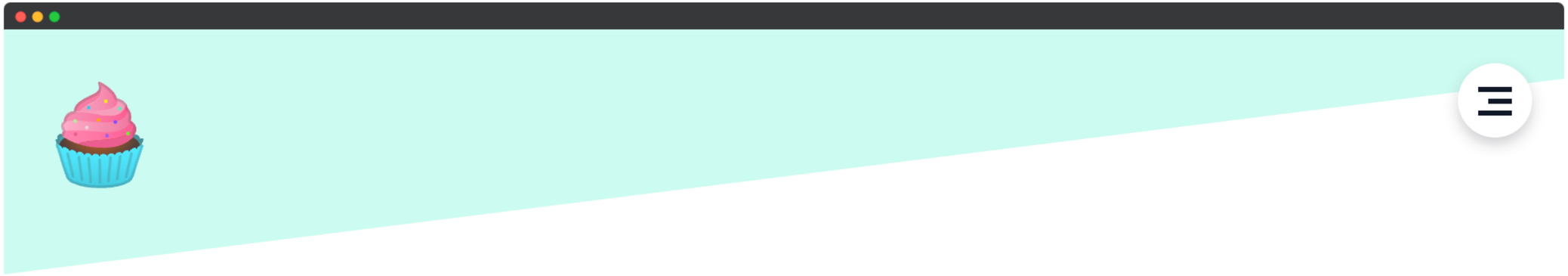
```
.Heading_root_asdfg0 {
  background: "navy";
  color: "plum";
  padding: "1rem";
}
```

Prod

```
._asdfg0 {
  background: "navy";
  color: "plum";
  padding: "1rem";
}
```

```ts
// ./components/Heading.css.ts
import { style } from '@vanilla-extract/css'

export const root = style({
  background: "navy",
  color: "plum",
  padding: "1rem"
})
```

```
// ./components/Heading.css.ts
import { style } from '@vanilla-extract/css'

export const root = style({
  background: "navy",
  color: "plum",
  padding: "1rem"
  selectors: {
    "&:hover": {
      background: "indigo"
    }
  }
})
```



globalStyle

Creates styles attached to a global selector.

Requires a selector string as the first parameter, followed by a style object.

#globalStyle

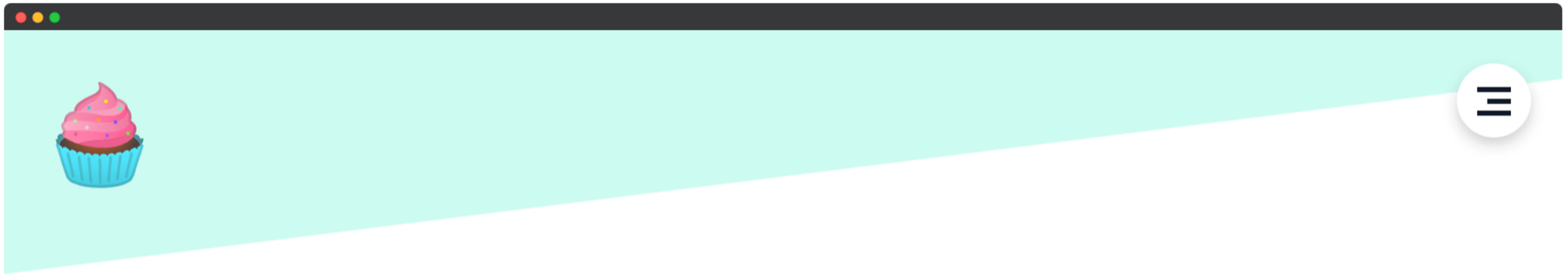
```ts
// global.css.ts
import { globalStyle } from '@vanilla-extract/css'

globalStyle('html, body', {
  margin: 0
})
```

```
html, body {
  margin: 0;
}
```

```ts
// global.css.ts
import { globalStyle, style } from '@vanilla-extract/css'

export const rootClass = style({})

globalStyle(`${rootClass} > a`, {
  color: "navy"
})
```

```
.app_rootClass__asdfg0 > a {
  color: "navy";
}
```

```tsx
// app.tsx
import { rootClass } from '@styles/global.css'

export const App = () => (
  <div className={rootClass}>
    ...
  </div>
)
```



styleVariants

Creates a collection of named style rules.

This is useful for mapping component props to styles, for example:

```
<button className={styles.background[props.variant]}>
```

#styleVariants

```
// ./components/Heading.css.ts
import { styleVariants } from '@vanilla-extract/css'

export const variants = styleVariants({
  primary: {
    background: "navy",
    color: "plum",
    padding: "1rem"
  },
  secondary: {
    background: "plum",
    color: "navy",
    padding: "1rem"
  },
})
```

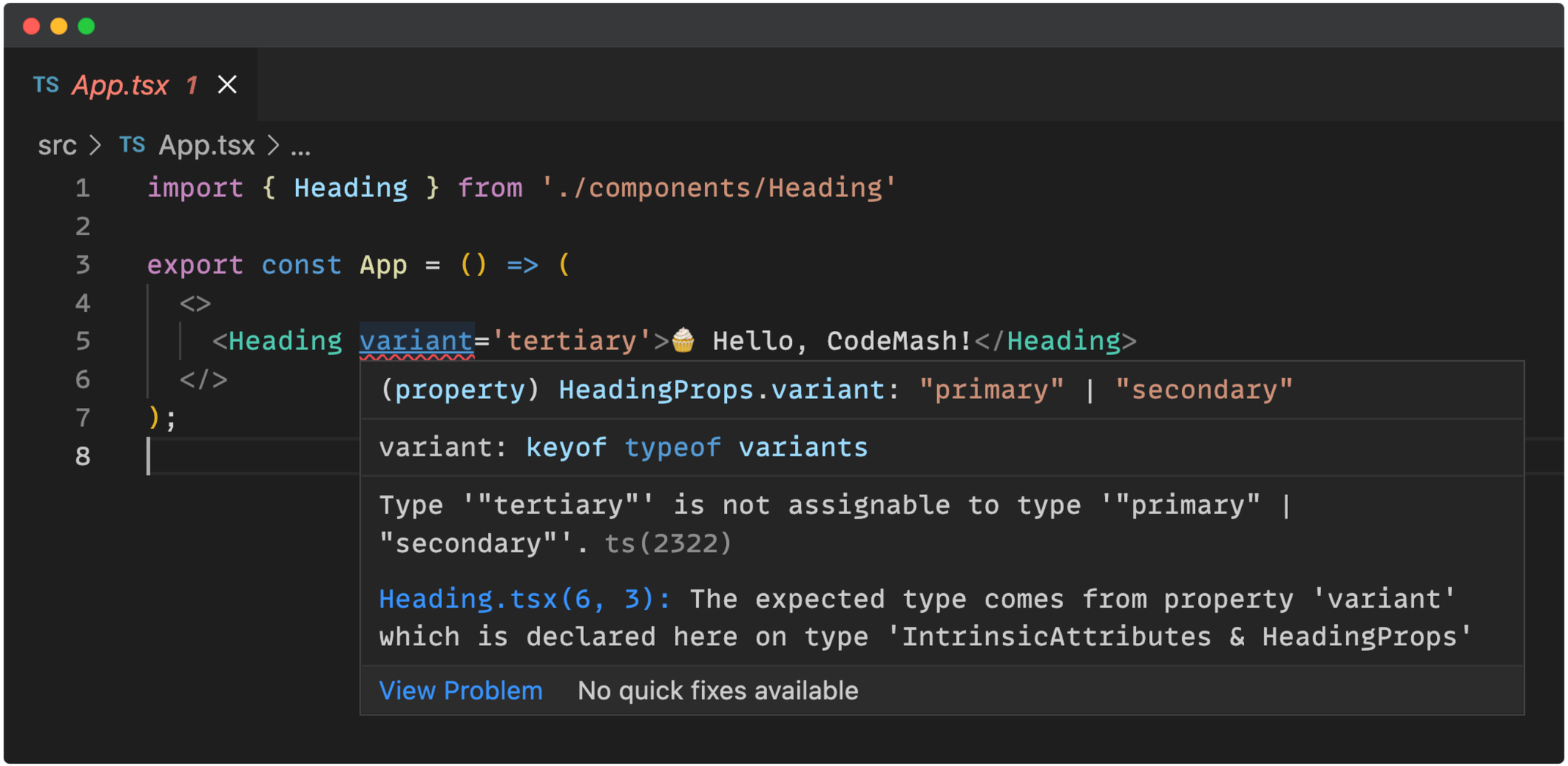
```tsx
// ./components/Heading.tsx
import React from 'react'
import { variants } from './Heading.css'

export const Heading = ({ variant, children }) => (
  <div className={ variants[variant] }>
    <h1>{ children }</h1>
  </div>
)
```

```tsx
// app.tsx
import { Heading } from './components/Heading'

export const App = () => (
  <>
    <Heading variant="primary">🧁 Hello, StirTrek!</Heading>
  </>
);
```

```tsx
// app.tsx
import { Heading } from './components/Heading'

export const App = () => (
  <>
    <Heading variant="secondary">🧁 Hello, StirTrek!</Heading>
  </>
);
```


Hello, StirTrek!

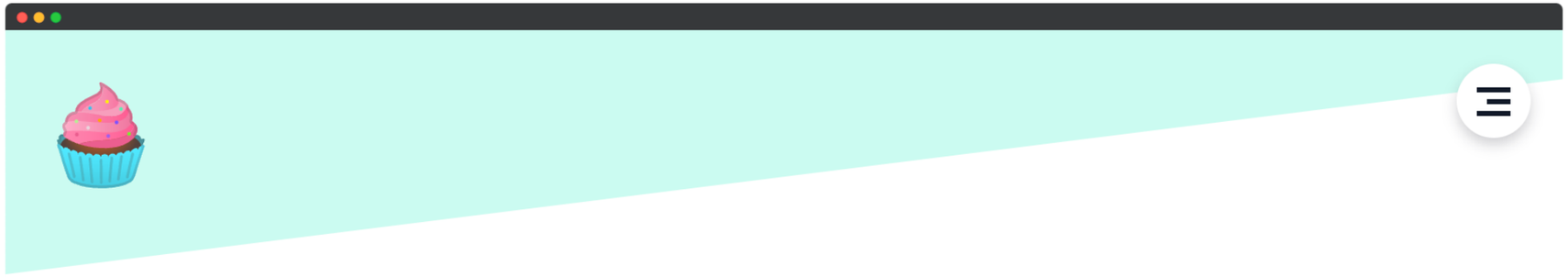
```tsx
// ./components/Heading.tsx
import React from 'react'
import { variants } from './Heading.css'

type HeadingProps = {
  variant: keyof typeof variants
}

export const Heading = ({ variant, children }: HeadingProps) => (
  <div className={ variants[variant] }>
    <h1>{ children }</h1>
  </div>
)
```


TS App.tsx 1
src > TS App.tsx > ...
1 import { Heading } from './components/Heading'
2
3 export const App = () => (
4 <>
5 <Heading variant='tertiary'>🧁 Hello, CodeMash!</Heading>
6 </>
7);
8
(property) HeadingProps.variant: "primary" | "secondary"
variant: keyof typeof variants
Type '"tertiary"' is not assignable to type '"primary" | "secondary"'. ts(2322)
Heading.tsx(6, 3): The expected type comes from property 'variant' which is declared here on type 'IntrinsicAttributes & HeadingProps'
View Problem
No quick fixes available

```
// ./components/Heading.css.ts
import { styleVariants } from '@vanilla-extract/css'

export const variants = styleVariants({
  primary: {
    background: "navy",
    color: "plum",
    padding: "1rem"
  },
  secondary: {
    background: "plum",
    color: "navy",
    padding: "1rem"
  },
})
```

```ts
// ./components/Heading.css.ts
import { style, styleVariants } from '@vanilla-extract/css'

const space = style({
  padding: "1rem",
});

export const variants = styleVariants({
  primary: [
    space,
    {
      background: "navy",
      color: "plum",
    },
  ],
  secondary: [
    space,
    {
      background: "plum",
      color: "navy",
    },
  ],
})
```

```
.Heading_variants_primary__asdfg1 {
    background: navy;
    color: plum;
}

.Heading_space__asdfg0 {
    padding: 1rem;
}
```



createGlobalTheme

Creates a theme attached to a global selector, but with locally scoped variable names.

🎨 New to theming in vanilla-extract? Make sure you've read the theming overview first.

#createGlobalTheme

```ts
// theme.css.ts
import { createGlobalTheme } from "@vanilla-extract/css";

export const vars = createGlobalTheme(":root", {
  color: {
    primary: "navy",
    secondary: "plum",
  },
  space: {
    s: ".5rem",
    m: "1rem",
    l: "2rem",
  },
});
```

```
:root {
  --color-primary__asdfg0: "navy";
  --color-secondary__asdfg1: "plum";
  --space-s__asdfg2: "0.5rem";
  --space-m__asdfg3: "1rem";
  --space-l__asdfg4: "2rem";
}
```

```ts
// Heading.css.ts
import { style, styleVariants } from '@vanilla-extract/css'

const space = style({
  padding: "1rem",
});

export const variants = styleVariants({
  primary: [
    space,
    {
      background: "navy",
      color: "plum",
    },
  ],
  secondary: [
    space,
    {
      background: "plum",
      color: "navy",
    },
  ],
})
```

```ts
// Heading.css.ts
import { style, styleVariants } from '@vanilla-extract/css'
import { vars } from '../../styles/theme.css'

const space = style({
  padding: vars.space.s,
});

export const variants = styleVariants({
  primary: [
    space,
    {
      background: vars.color.primary,
      color: vars.color.secondary,
    },
  ],
  secondary: [
    space,
    {
      background: vars.color.secondary,
      color: vars.color.primary,
    },
  ],
})
```

```
.Heading_variants_primary__asdfg1 {
    background: var(--color-primary__asdfg0);
    color: var(--color-secondary__asdfg1);
}

.Heading_space__asdfg0 {
    padding: var(--space-s__asdfg2);
}
```

```ts
// theme.css.ts
import { createGlobalTheme } from "@vanilla-extract/css";

export const vars = createGlobalTheme(":root", {
  color: {
    primary: "navy",
    secondary: "plum",
  },
  space: {
    s: ".5rem",
    m: "1rem",
    l: "2rem",
  },
});
```

createTheme

Creates a locally scoped theme class and a theme contract which can be consumed within your styles.

#createTheme

Dynamic

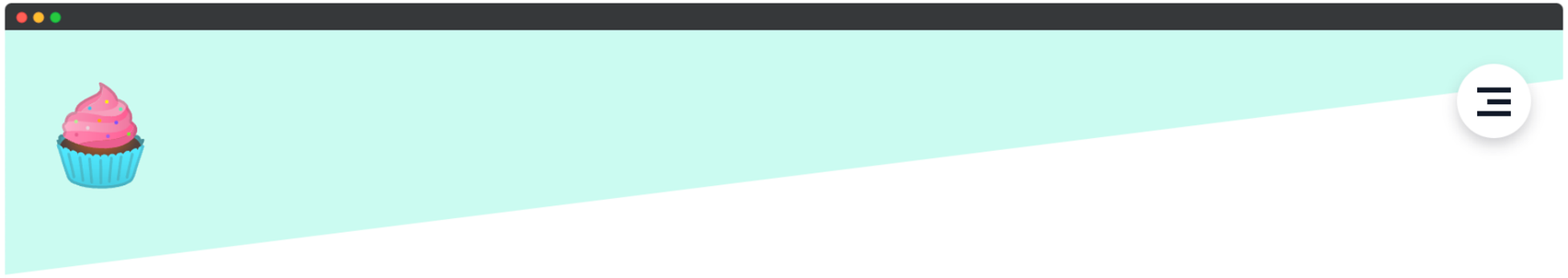
A tiny (< 1kB compressed) runtime for performing dynamic updates to scoped theme variables.

```
npm install @vanilla-extract/dynamic
```

#dynamic

```tsx
// app.tsx
import { assignInlineVars } from '@vanilla-extract/dynamic'
import { vars } from '@styles/theme.css.ts'

export const App = () => (
  <div
    style={assignInlineVars(vars, {
      color: {
        primary: "aqua",
        secondary: "teal",
      },
      space: {
        s: "1rem",
        m: "3rem"
        l: "5rem"
      }
    )}>
    { children }
  </div>
)
```

- Performance ✅
- DevX ✅
- Scoped styles ✅
- Typesafe ✅
- Runtime theming ✅
- Atomic styles ❓



Sprinkles

A zero-runtime atomic CSS framework for vanilla-extract.

Generate a static set of custom utility classes and compose them either statically at build time, or dynamically at runtime, without the usual style generation overhead of CSS-in-JS.

Basically, it's like building your own zero-runtime, type-safe version of Tailwind, Styled System, etc.

#sprinkles

```ts
// sprinkles.css.ts
import { defineProperties, createSprinkles } from "@vanilla-extract/sprinkles";
import { vars } from "./theme.css";

const properties = defineProperties({
  conditions: {
    mobile: {},
    tablet: { "@media": "screen and (min-width: 768px)" },
    desktop: { "@media": "screen and (min-width: 1024px)" },
  },
  defaultCondition: "mobile",
  properties: {
    color: vars.color,
    background: vars.color,
    padding: vars.space,
  }
});

export const sprinklesFn = createSprinkles(properties);
```

```
.sprinkles_color_primary_mobile__asdfg0 {
  color: var(--color-primary__asdfg0)
}
.sprinkles_color_secondary_mobile__asdfg0 {
  color: var(--color-secondary__asdfg1)
}
.sprinkles_background_primary_mobile__asdfg0 {
  background: var(--color-primary__asdfg2)
}
.sprinkles_background_secondary_mobile__asdfg0 {
  background: var(--color-secondary__asdfg3)
}
.sprinkles_space_s_mobile__asdfg0 {
  padding: var(--space-s__asdfg4)
}
.sprinkles_space_m_mobile__asdfg0 {
  padding: var(--space-m__asdfg4)
}
.sprinkles_space_m_mobile__asdfg0 {
  padding: var(--space-m__asdfg4)
}
```

```
@media screen and (min-width: 768px) {
  .sprinkles_color_primary_tablet__asdfg0 {
    color: var(--color-primary__asdfg0)
  }
  .sprinkles_color_secondary_tablet__asdfg0 {
    color: var(--color-secondary__asdfg1)
  }
  .sprinkles_background_primary_tablet__asdfg0 {
    background: var(--color-primary__asdfg2)
  }
  .sprinkles_background_secondary_tablet__asdfg0 {
    background: var(--color-secondary__asdfg3)
  }
  .sprinkles_space_s_tablet__asdfg0 {
    padding: var(--space-s__asdfg4)
  }
  .sprinkles_space_m_tablet__asdfg0 {
    padding: var(--space-m__asdfg4)
  }
  .sprinkles_space_m_tablet__asdfg0 {
    padding: var(--space-m__asdfg4)
  }
}
```

```
@media screen and (min-width: 1024px) {
  .sprinkles_color_primary_desktop__asdfg0 {
    color: var(--color-primary__asdfg0)
  }
  .sprinkles_color_secondary_desktop__asdfg0 {
    color: var(--color-secondary__asdfg1)
  }
  .sprinkles_background_primary_desktop__asdfg0 {
    background: var(--color-primary__asdfg2)
  }
  .sprinkles_background_secondary_desktop__asdfg0 {
    background: var(--color-secondary__asdfg3)
  }
  .sprinkles_space_s_desktop__asdfg0 {
    padding: var(--space-s__asdfg4)
  }
  .sprinkles_space_m_desktop__asdfg0 {
    padding: var(--space-m__asdfg4)
  }
  .sprinkles_space_m_desktop__asdfg0 {
    padding: var(--space-m__asdfg4)
  }
}
```

```ts
// sprinkles.css.ts
import { defineProperties, createSprinkles } from "@vanilla-extract/sprinkles";
import { vars } from "./theme.css";

const properties = defineProperties({
  // ...
  properties: {
    // ...
    paddingTop: vars.space,
    paddingBottom: vars.space,
    paddingLeft: vars.space,
    paddingRight: vars.space,
  }
  shorthands: {
    padding: ["paddingTop", "paddingBottom", "paddingLeft", "paddingRight"],
    paddingX: ["paddingLeft", "paddingRight"],
    paddingY: ["paddingTop", "paddingBottom"],
  },
});

export const sprinklesFn = createSprinkles(properties);
```

```ts
// Heading.css.ts
import { style } from '@vanilla-extract/css'
import { vars } from '../../styles/theme.css'

export const root = style({
  background: vars.color.primary,
  color: vars.color.secondary,
  padding: vars.space.s,
})
```

```ts
// ./components/Heading.css.ts
import { style } from '@vanilla-extract/css'
import { sprinklesFn } from '../../styles/sprinkles.css'

export const root = style([
  sprinklesFn({
    background: "primary",
    color: "secondary",
    padding: "s"
  })
])
```

```
.sprinkles_background_primary_mobile__asdfg6 {
    background: var(--color-primary__asdfg1);
}
.sprinkles_color_secondary_mobile__asdfg3 {
    color: var(--color-secondary__asdfg2);
}
.sprinkles_paddingRight_s_mobile__asdfg13 {
    padding-right: var(--space-s__asdfg3);
}
.sprinkles_paddingLeft_s_mobile__asdfgu {
    padding-left: var(--space-s__asdfg3);
}
.sprinkles_paddingBottom_s_mobile__asdfgl {
    padding-bottom: var(--space-s__asdfg3);
}
.sprinkles_paddingTop_s_mobile__asdfgc {
    padding-top: var(--space-s__asdfg3);
}
```

```ts
// ./components/Heading.css.ts
import { style } from '@vanilla-extract/css'
import { sprinklesFn } from '../../styles/sprinkles.css'

export const variants = style([
  sprinklesFn({
    background: "primary",
    color: "secondary",
    paddingX: "s"
  })
])
```

```
.sprinkles_background_primary_mobile__asdfg6 {
    background: var(--color-primary__asdfg1);
}
.sprinkles_color_secondary_mobile__asdfg3 {
    color: var(--color-secondary__asdfg2);
}
.sprinkles_paddingRight_s_mobile__asdfg13 {
    padding-right: var(--space-s__asdfg3);
}
.sprinkles_paddingLeft_s_mobile__asdfgu {
    padding-left: var(--space-s__asdfg3);
}
```

```ts
// ./components/Heading.css.ts
import { style } from '@vanilla-extract/css'
import { sprinklesFn } from '../../styles/sprinkles.css'

export const variants = style([
  sprinklesFn({
    background: "primary",
    color: "secondary",
    paddingLeft: {
      mobile: "s",
      tablet: "m",
      desktop: "l"
    }
  })
])
```

```
.sprinkles_background_primary_mobile__asdfg6 {
  background: var(--color-primary__asdfg1);
}
.sprinkles_color_secondary_mobile__asdfg3 {
  color: var(--color-secondary__asdfg2);
}
.sprinkles_paddingLeft_s_mobile__asdfgu {
  padding-left: var(--space-s__asdfg3);
}
@media screen and (min-width: 768px) {
  .sprinkles_paddingLeft_m_tablet__asdfgu {
    padding-left: var(--space-m__asdfg3);
  }
}
@media screen and (min-width: 1024px) {
  .sprinkles_paddingLeft_l_desktop__asdfgu {
    padding-left: var(--space-l__asdfg3);
  }
}
```

```tsx
// ./components/Heading.tsx
import React from 'react'
import { sprinklesFn } from '../../styles/sprinkles.css'

export const Heading = ({ children }) => (
  <div
    className={sprinklesFn({
      background: "primary",
      color: "secondary",
      padding: "s"
    })}
  >
    <h1>{ children }</h1>
  </div>
)
```

```ts
// ./styles/sprinkles.css.ts
import { defineProperties, createSprinkles } from "@vanilla-extract/sprinkles";
import { vars } from "./theme.css";

const properties = defineProperties({
  conditions: {
    ...
  },
  defaultCondition: "mobile",
  properties: {
    ...
  },
  shorthands: {
    ...
  },
});

export const sprinklesFn = createSprinkles(properties);

export type Sprinkles = Parameters<typeof sprinklesFn>[0];
```

```tsx
// ./components/Heading.tsx
import React from 'react'
import { sprinklesFn, Sprinkles } from '../../styles/sprinkles.css'

type HeadingProps = {
  background?: Sprinkles["background"];
  color?: Sprinkles["color"]
  padding?: Sprinkles["padding"]
}

export const Heading = ({ background, color, padding, children }: HeadingProps) => (
  <div
    className={sprinklesFn({
      background,
      color,
      padding
    })}
  >
    <h1>{ children }</h1>
  </div>
)
```

Search... ⌘K 3

Edit this page

Box

The Box component serves as a wrapper component for most of the CSS utility needs.

The Box component packages all the style functions that are exposed in `@mui/system`.

Box

Box is the most abstract component on top of which all other Chakra UI components are built. By default, it renders a `div` element

Source @chakra-ui/layout

Usage Props

Import

```
import { Box } from '@chakra-ui/react'
```

COPY

Usage

The Box component is useful because it helps with three common use cases:

- Create responsive layouts with ease.
- Provide a shorthand way to pass styles via props (`bg` instead of `backgroundColor`).
- Compose new component and allow for override using the `as` prop.

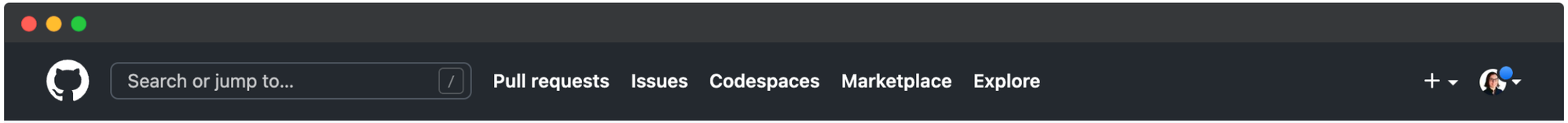
```tsx
// ./components/Box.tsx
import React from 'react'

type BoxProps = {
  children: React.ReactNode;
};

export const Box = ({ children }: BoxProps) => (
  <div>{ children }</div>
);
```

```tsx
// ./components/Box.tsx
import React from 'react'

type BoxProps = {
  as: React.ElementType
  children: React.ReactNode;
};

export const Box = ({ as: As, children }: BoxProps) => (
  <As>{ children }</As>
);
```

```tsx
// ./components/Box.tsx
import React from 'react'

import { sprinklesFn, Sprinkles } from "../../styles/sprinkles.css";

type BoxProps = {
  as: React.ElementType;
  children: React.ReactNode;
  sprinkles: Sprinkles;
};

export const Box = ({ as: As, sprinkles, children }: BoxProps) => (
  <As className={sprinklesFn(sprinkles)}>{children}</As>
);
```

```tsx
// ./components/Heading.tsx
import React from 'react'
import { sprinklesFn, Sprinkles } from '../../styles/sprinkles.css'

type HeadingProps = {
  background?: Sprinkles["background"];
  color?: Sprinkles["color"]
  padding?: Sprinkles["padding"]
}

export const Heading = ({ background, color, padding, children }: HeadingProps) => (
  <div
    className={sprinklesFn({
      background,
      color,
      padding
    })}
  >
    <h1>{ children }</h1>
  </div>
)
```

```tsx
// ./components/Heading.tsx
import React from 'react'
import { Sprinkles } from '../../styles/sprinkles.css'
import { Box } from './Box'

type HeadingProps = {
  background?: Sprinkles["background"];
  color?: Sprinkles["color"]
  padding?: Sprinkles["padding"]
}

export const Heading = ({ background, color, padding, children }: HeadingProps) => (
  <Box as="div" sprinkles={{ background, color, padding }}>
    <h1>{ children }</h1>
  </Box>
)
```



TheMightyPenguin / dessert-box Public

Watch 4 | Fork 15 | Star 220

Code | Issues 4 | Pull requests 4 | Actions | Projects | Security | Insights

main | dessert-box / README.md | Go to file | ...

TheMightyPenguin Add escape hatch for Box — Latest commit 1d99a5c on Jun 5, 2022 — History

3 contributors

269 lines (189 sloc) | 9.1 KB | Raw | Blame

🍰 Dessert Box

A library to easily consume your design tokens from a React component, meant to be used with vanilla-extract.

This library will make consuming your sprinkles from a react component a breeze. It provides a zero-CSS-runtime `<Box />` component (similar to the one in Braid or Chakra).

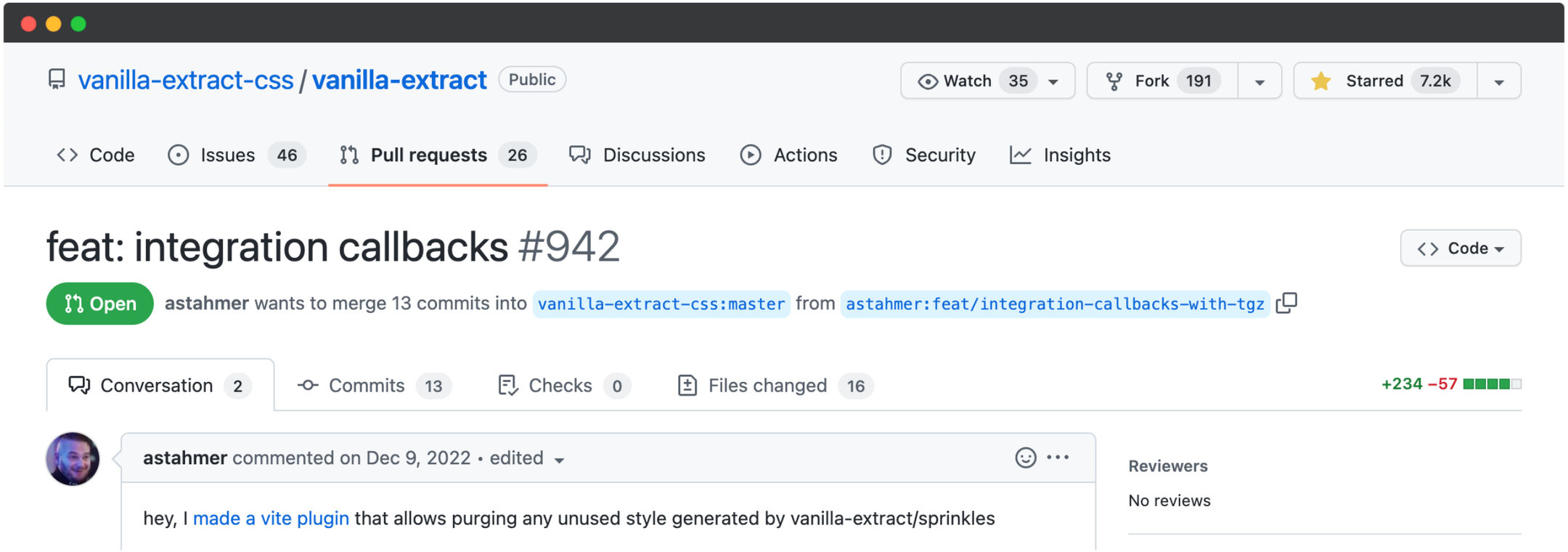
Try it on CodeSandbox!

It works by consuming `atoms` created with `vanilla-extract`) and `sprinkles`. Shout out to the team at Seek for making these awesome libraries!

- Performance ✅
- DevX ✅
- Scoped styles ✅
- Typesafe ✅
- Runtime theming ✅
- Atomic styles ✅

Drawbacks

- Atomic classes purging
- Dynamic inline style assignments


vanilla-extract-css / vanilla-extract
Public
Watch 35
Fork 191
Starred 7.2k
Code
Issues 46
Pull requests 26
Discussions
Actions
Security
Insights
feat: integration callbacks #942
Code
Open
astahmer wants to merge 13 commits into vanilla-extract-css:master from astahmer:feat/integration-callbacks-with-tgz
Conversation 2
Commits 13
Checks 0
Files changed 16
+234 −57
astahmer commented on Dec 9, 2022 • edited
hey, I made a vite plugin that allows purging any unused style generated by vanilla-extract/sprinkles
Reviewers
No reviews

Atomic classes purging

- Limit sprinkles to commonly used values
- Lean on `style()` for idiosyncratic values/properties
- Use multiple `defineProperties` to limit conditions

Drawbacks

- Atomic classes purging
- Dynamic inline style assignments

Rainbow Sprinkles 🧁

Dynamic, theme-driven, style props for vanilla-extract

release rainbow-sprinkles@0.15.1 License MIT Contributor Covenant 2.0 Maintainer Wayfair

Rainbow sprinkles works similarly to `@vanilla-extract/sprinkles`. Like sprinkles, it generates custom CSS utility classes at build time. While sprinkles requires a pre-defined list of available values, Rainbow Sprinkles uses CSS custom properties to allow dynamic values using inline style variable assignments.

Compared to sprinkles:

- **Rainbow sprinkles ships a fraction of the CSS.** For each property, Sprinkles produces CSS that's a factor of `[pre-defined values] * [possible conditions]`. Rainbow sprinkles produces CSS that only scales with the number of conditions.
- **Supports dynamic values.** Rainbow Sprinkles uses dynamic inline style assignments to set the value of each property. You still get the TypeScript editor suggestions, but the ability to use any valid CSS value for that property.


Modern TS Web App
with Vanilla-extract


SCAN
ME
Slides 👆