Build Out Club

Advancing Our TailwindCSS Use

Henry Bley-Vroman

MARCH 10, 2021

viget

1

Style Convention Thoughts

Idiomatic names create friction

“Inconsistent class naming. This is probably the most annoying thing to me.”

— Trevor Davis, *What I Love & Hate About Tailwind CSS*

TailwindCSS diverges from CSS by inventing its own names for things. That creates friction when translating from Figma to code. There’s potential in the values we configure for reducing that friction while playing by TailwindCSS’s rules.

```
/* Desktop/Heading/H3 */
font-family: Montserrat;
font-style: normal;
font-weight: 600;
font-size: 28px;
line-height: 125%;
/* identical to box height, or
35px */
letter-spacing: 0.05em;
text-transform: uppercase;
```

Idiomatic names create friction

```
module.exports = {
  // ...
  fontWeight: {
    // ...
    semibold: '600',
    // ...
  },
  letterSpacing: {
    // ...
    wider: '0.05em',
    // ...
  },
  lineHeight: {
    // ...
    tight: '1.25',
    // ...
  },
  // ...
}
```

```
/* Desktop/Heading/H3 */
font-family: Montserrat;
font-style: normal;
font-weight: 600;
font-size: 28px;
line-height: 125%;
/* identical to box height, or
35px */
letter-spacing: 0.05em;
text-transform: uppercase;
```

```
<div class="font-semibold leading-tight tracking-wider">
```

“Semantic.” Idiomatic. Taxes memory. Does not align with Figma. Font weight may not even align with font names.

```
module.exports = {
  // ...
  fontWeight: {
    // ...
    600: '600',
    // ...
  },
  letterSpacing: {
    // ...
    '0.05': '0.05em',
    // ...
  },
  lineHeight: {
    // ...
    '125%': '125%',
    // ...
  },
  // ...
}
```

```
<div class="font-600 leading-125% tracking-0.05">
```

Standard. Matches CSS knowledge. Aligns with Figma.

Idiomatic names create friction

Font weight “thin” etc

Letter spacing “wider” etc

Line height “tight” etc

Opacity “75” etc

Scale “100” etc

Skew “3” etc

“Semantic.” Idiomatic. Taxes memory. Does not align with Figma. Font weight may not even align with font names.

Font weight “300” etc

Letter spacing “0.05” etc

Line height “125%” etc

Opacity “0.75” etc

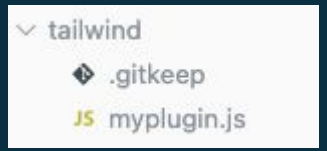
Scale “1” etc

Skew “3deg” etc

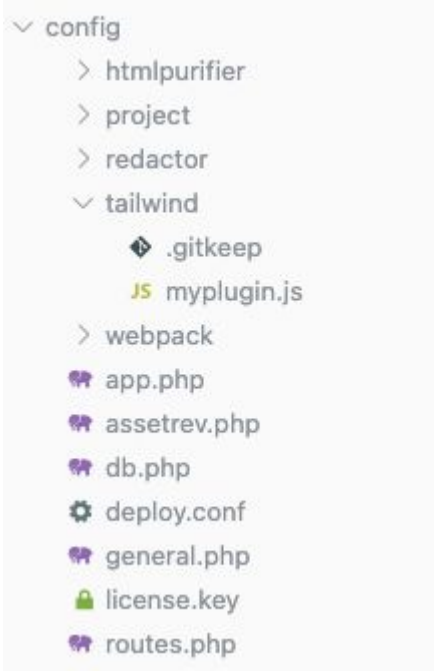
Standard. Matches CSS knowledge. Aligns with Figma.

Plugin source code is part of “source”

Custom plugins are a common feature in our TailwindCSS sites. As part of the site source code, it is a natural candidate for the `src` directory.



Plugin source code is part of “source”



Unique location not shared by other source files. Does not lend itself to CODEOWNERSing. Maybe the hot-reloading watcher doesn’t pick up on it? Was not able to test.

Same location as the rest of the code sourcefiles. Lends itself to CODEOWNERS. Build tooling seeing when it changes.

2

Plugins & Their Use

1

Utilities

```
const textDecorationColor = plugin(({ addUtilities, e, theme, variants }) => {
    const pluginConfig = theme('textDecorationColor', {})
    const pluginVariants = variants('textDecorationColor', [])
    const colors = flattenColorPalette(pluginConfig)

    const utilities = Object.entries(colors).map(([name, value]) => {
        return {
            [`.${e(`underline-${name}`)}`]: {
                textDecorationColor: value,
            }
        }
    })

    addUtilities(utilities, pluginVariants)
})
```

```
.underline-black { …
}

.underline-white { …
}

.underline-blue-400 { …
}
```

```
const reversed = plugin(({ addVariant, e }) => {
    addVariant('reversed', ({ modifySelectors, separator }) => {
        modifySelectors(({ className }) => {
            return `.reversed .${e(
                `reversed${separator}${className}`
            )}, .reversed.${e(`reversed${separator}${className}`)}`
        })
    })
})
```

```
.reversed .reversed\:ring-6, .reversed.reversed\:ring-6 {
```

“Stackable” variants

```
<div class="group">
  <div class="whatintent-keyboard:group-focus:scale-0.8">
```

```
[data-whatintent="keyboard"] .whatintent-keyboard\:group:focus [data-whatintent="keyboard"] .whatintent-keyboard\:group-focus\:scale-0\.8,
[data-whatintent="keyboard"] .whatintent-keyboard\:group:hover [data-whatintent="keyboard"] .whatintent-keyboard\:group-hover\:scale-0\.8 {
  --tw-scale-x: 0.8;
  --tw-scale-y: 0.8;
}
```

Plugin code gets long

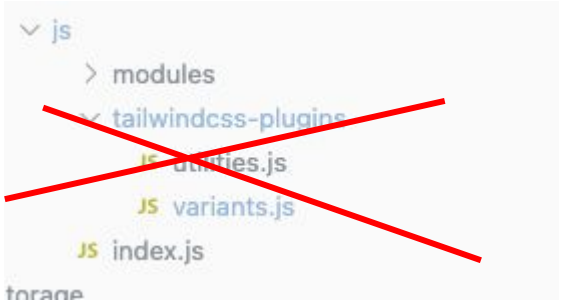
```
const whatintentKeyboard = plugin(({ addVariant }) => {
  addVariant(
    'whatintent-keyboard',
    ({ modifySelectors, separator }) => {
      return modifySelectors(({ selector }) => {
        return selectorParser((selectors) => {
          selectors.walkClasses((sel) => {
            sel.value = `whatintent-keyboard${separator}${sel.value}`
            sel.parent.insertBefore(sel, selectorParser().astSync('[data-whatintent="keyboard"] '))
          })
        }).processSync(selector)
      })
    },
    { unstable_stack: true }
  )
})
```

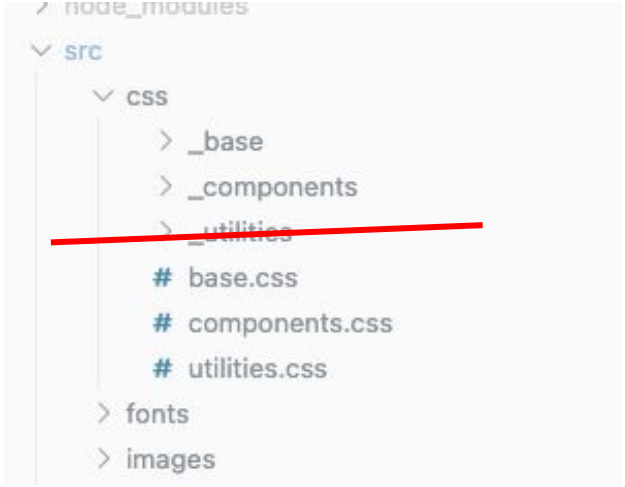
Let's refactor that

```
const variantPlugin = ({ name, prefix = null, suffix = null, unstable_stack = false }) => {
    return plugin(({ addVariant }) => {
        addVariant(
            name,
            ({ modifySelectors, separator }) => {
                return modifySelectors(({ selector }) => {
                    return selectorParser((selectors) => {
                        selectors.walkClasses((sel) => {
                            sel.value = `${name}${separator}${sel.value}`
                            prefix && sel.parent.insertBefore(sel, selectorParser().astSync(`${prefix}`))
                            suffix && sel.parent.insertAfter(sel, selectorParser().astSync(`${suffix}`))
                        })
                    }).processSync(selector)
                })
            },
            { unstable_stack: unstable_stack }
        )
    })
}
const whatintentKeyboard = variantPlugin({
    name: 'whatintent-keyboard',
    prefix: '[data-whatintent="keyboard"] ',
    unstable_stack: true,
})
```

Now that it's short, it can go in tailwind.config.js!

Realized during the presentation that this can probably just be TailwindCSS's `addUtility`. If it gets long, ask whether it could move to markup or be split






```
plugins: [
    utilityPlugin({
        /* tbd */
    }),
    variantPlugin({
        name: 'myvariant',
        prefix: '.my ',
        suffix: ' .variant',
    }),
```

```
.my hover\:.myvariant\:text-18 .variant:hover {
    @apply text-18;
}
```

tailwind TYPOGRAPHY

Branded as a solution for rich text

```
typography: (theme) => ({
  'rtf': {
    css: {
      color: theme('colors.gray.400'),
      fontSize: theme('fontSize.16'),
      lineHeight: theme('lineHeight["1.5"]'),
      a: {
        textDecoration: 'underline',
        transitionDuration: theme('transitionDuration.DEFAULT'),
        transitionProperty: 'text-decoration-color',
        '&:hover': {
          textDecorationColor: theme('colors.yellow.400'),
        },
      },
      h1: {
        color: theme('colors.blue.400'),
        fontSize: theme('fontSize.21'),
        fontWeight: theme('fontWeight.600'),
        letterSpacing: theme('letterSpacing["0.05"]'),
        lineHeight: theme('lineHeight["1.25"]'),
        textTransform: 'uppercase',
      },
      h2: {
        color: theme('colors.blue.400'),
        fontSize: theme('fontSize.20'),
```

```
<div class="prose-rtf">
```

tailwind TYPOGRAPHY

Undocumented option which will open doors: the classes do not *have* to have anything to do with “typography.”

```
plugins: [
    typography({
        modifiers: [],
        className: 'some-class-name',
    }),
```

```
<div class="some-class-name-rtf">
```

tailwind TYPOGRAPHY

Branded as a solution for rich text… but really it's a first-party solution for defining components in the config file

tailwind TYPOGRAPHY

Branded as a solution for rich text… but really it's a first-party solution for defining components in the config file. Let's fix the surface implications.

```
plugins: [
    typography({
        modifiers: [],
        className: 'component',
    }),
```

```
typography: (theme) => ({
    'specialthing': {
        css: {
            // ...
        }
    }
    'rtf': {
        css: {
            color: theme('colors.gray.400'),
            fontSize: theme('fontSize.16'),
```

```
<div class="component-rtf">

<div class="component-specialthing">
```

tailwind TYPOGRAPHY

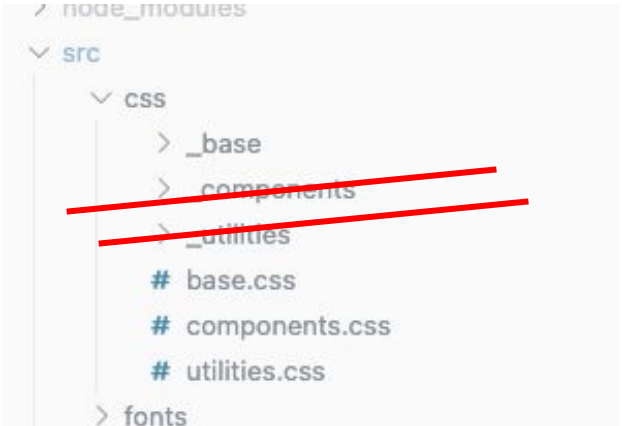
Components can now go in tailwind.config.js!

```
plugins: [
    typography({
        modifiers: [],
        className: 'component',
    }),
```

```
typography: (theme) => ({
    'specialthing': {
        css: {
            // ...
        }
    }
    'rtf': {
        css: {
            color: theme('colors.gray.400'),
            fontSize: theme('fontSize.16'),
```

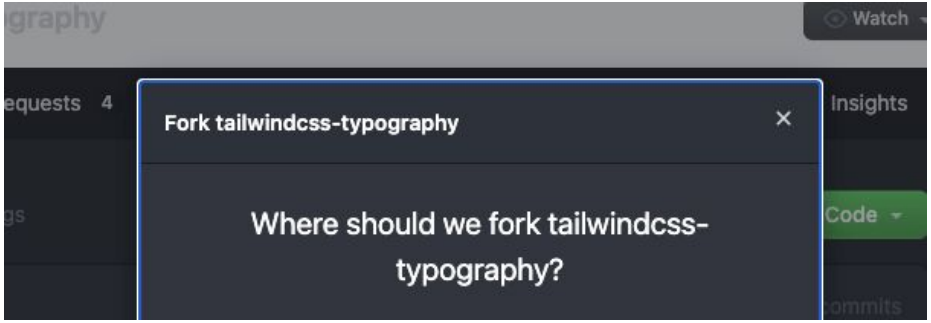
```
<div class="component-rtf">

<div class="component-specialthing">
```



tailwind TYPOGRAPHY

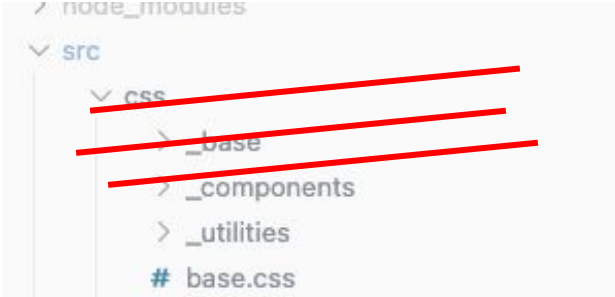
The only thing we definitely still need stylesheets for is the base layer. That could be taken care of by forking tailwind-typography and making some small changes to generate base layer styles! (post-presentation “aha”: maybe it’s just `addBase()` in `plugins`??)




```
<!DOCTYPE html>
<html dir="ltr" lang="en-US" class="base"
```

```
<div class="component-rtf">

<div class="component-specialthing">
```



The Future?

Base layer via tailwindcss-base-layer™
... or *update since presenting: maybe this is just the built-in `addBase` in `plugins`*

```
baseLayer: {
    'base': {
        css: {
            //...
        }
    }
}
```

```
<!DOCTYPE html>
<html dir="ltr" lang="en-US" class="base"
```

Component layer via tailwindcss-typography

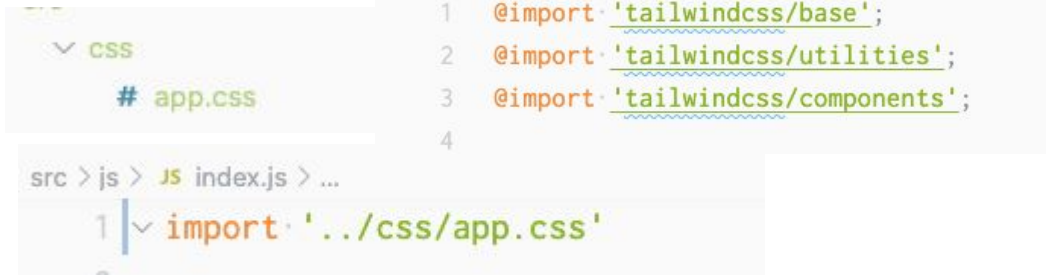
```
<div class="component-rtf">

<div class="component-specialthing">
```

```
plugins: [
    typography({
        modifiers: [],
        className: 'component',
    }),
```

```
}),
typography: (theme) => ({
    'specialthing': {
        css: {
            //...
        }
    }
    'rtf': {
        css: {
```

Single entry point and never have to look in the css directory



Plugins written in a form short enough to live in the config file. *Update since presenting: utilityPlugin() is probably just the built-in `addUtility`*

```
plugins: [
    utilityPlugin({/* a utility */}),
    utilityPlugin({/* another utility */}),
    variantPlugin({/* a variant */}),
    variantPlugin({/* another variant */}),
    typography({
        modifiers: [],
        className: 'component',
    }),
    //... imported Viget plugins / etc
```