Styling

Nuxt is highly flexible when it comes to styling. Write your own styles, or reference local and external stylesheets. You can use CSS preprocessors, CSS frameworks, UI libraries and Nuxt modules to style your application.

Local Stylesheets

If you're writing local stylesheets, the natural place to put them is the `assets/` directory.

Importing Within Components

You can import stylesheets in your pages, layouts and components directly. You can use a javascript import, or a css `@import` statement.

pages/index.vue

```
<script>
// Use a static import for server-side compatibility
import '~/assets/css/first.css'

// Caution: Dynamic imports are not server-side compatible
import('~/assets/css/first.css')
</script>

<style>
@import url("~/assets/css/second.css");
</style>
```

The stylesheets will be inlined in the HTML rendered by Nuxt.

The CSS Property

You can also use the `css` property in the Nuxt configuration. The natural place for your stylesheets is the `assets/` directory. You can then reference its path and Nuxt will include it to all the pages of your application.

```
nuxt.config.ts
export default defineNuxtConfig({
  css: ['~/assets/css/main.css']
})
```

The stylesheets will be inlined in the HTML rendered by Nuxt, injected globally and present in all pages.

Working With Fonts

Place your local fonts files in your `~/public/` directory, for example in `~/public/fonts` . You can then reference them in your stylesheets using `url()` .

```
assets/css/main.css
@font-face {
  font-family: 'FarAwayGalaxy';
  src: url('/fonts/FarAwayGalaxy.woff') format('woff');
  font-weight: normal;
  font-style: normal;
  font-display: swap;
}
```

Then reference your fonts by name in your stylesheets, pages or components:

```
<style>
h1 {
  font-family: 'FarAwayGalaxy', sans-serif;
}
</style>
```

Stylesheets Distributed Through NPM

You can also reference stylesheets that are distributed through npm. Let's use the popular `animate.css` library as an example.

```
> npm install animate.css
```

Then you can reference it directly in your pages, layouts and components:

```
app.vue
<script>
import 'animate.css'
</script>

<style>
@import url("animate.css");
</style>
```

The package can also be referenced as a string in the css property of your Nuxt configuration.

```
nuxt.config.ts
export default defineNuxtConfig({
  css: ['animate.css']
})
```

External Stylesheets

You can include external stylesheets in your application by adding a link element in the head section of your nuxt.config file. You can achieve this result using different methods. Note that local stylesheets can also be included like this.

You can manipulate the head with the `app.head` property of your Nuxt configuration:

```
nuxt.config.ts
export default defineNuxtConfig({
  app: {
    head: {
      link: [{ rel: 'stylesheet', href: 'https://cdnjs.cloudflare.com/ajax/libs/animate.css/4.1.1/
    }
}})
```

Dynamically Adding Stylesheets

You can use the useHead composable to dynamically set a value in your head in your code.

👉 Read more in Docs > API > Composables > Use Head.

```
useHead({
  link: [{ rel: 'stylesheet', href: 'https://cdnjs.cloudflare.com/ajax/libs/animate.css/4.1.1/anim
})
```

Nuxt uses `unhead` under the hood, and you can refer to its full documentation here.

Modifying The Rendered Head With A Nitro Plugin

If you need more advanced control, you can intercept the rendered html with a hook and modify the head programmatically.

Create a plugin in `~/server/plugins/my-plugin.ts` like this:

server/plugins/my-plugin.ts

```
export default defineNitroPlugin((nitro) => {
  nitro.hooks.hook('render:html', (html) => {
    html.head.push('<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/animate.cs
  })
})
```

External stylesheets are render-blocking resources: they must be loaded and processed before the browser renders the page. Web pages that contain unnecessarily large styles take longer to render. You can read more about it on web.dev.

Using Preprocessors

To use a preprocessor like SCSS, Sass, Less or Stylus, install it first.

Sass & SCSS Less Stylus

```
Sass & SCSS
> npm install sass
```

The natural place to write your stylesheets is the `assets` directory. You can then import your source files in your `app.vue` (or layouts files) using your preprocessor's syntax.

```
pages/app.vue
<style lang="scss">
@use "~/assets/scss/main.scss";
</style>
```

Alternatively, you can use the `css` property of your Nuxt configuration.

```
nuxt.config.ts
export default defineNuxtConfig({
  css: ['~/assets/scss/main.scss']
})
```

In both cases, the compiled stylesheets will be inlined in the HTML rendered by Nuxt.

If you need to inject code in pre-processed files, like a sass partial with color variables, you can do so with the vite preprocessors options.

Create some partials in your `assets` directory:

assets/_colors.scss assets/_colors.sass

```
assets/_colors.scss
$primary: #49240F;
$secondary: #E4A79D;
```

Then in your `nuxt.config` :

SCSS SASS

```
SCSS
export default defineNuxtConfig({
  vite: {
    css: {
      preprocessorOptions: {
        scss: {
          additionalData: '@use "@/assets/_colors.scss" as *;'
        }
      }
    }
  }
})
```

Nuxt uses Vite by default. If you wish to use webpack instead, refer to each preprocessor loader documentation.

Single File Components (SFC) Styling

One of the best thing about Vue and SFC is how great it is at naturally dealing with styling. You can directly write CSS or preprocessor code in the style block of your components file, therefore you will have fantastic developer experience without having to use something like CSS-in-JS. However if you wish to use CSS-in-JS, you can find 3rd party libraries and modules that support it, such as pinceau.

You can refer to the Vue docs for a comprehensive reference about styling components in SFC.

Class And Style Bindings

You can leverage Vue SFC features to style your components with class and style attributes.

Ref and Reactive   Computed   Array   Style

Ref and Reactive

```
<script setup lang="ts">
const isActive = ref(true)
const hasError = ref(false)
const classObject = reactive({
  active: true,
  'text-danger': false
})
</script>

<template>
  <div class="static" :class="{ active: isActive, 'text-danger': hasError }"></div>
  <div :class="classObject"></div>
</template>
```

Refer to the Vue docs for more information.

Dynamic Styles With `v-bind`

You can reference JavaScript variable and expression within your style blocks with the v-bind function. The binding will be dynamic, meaning that if the variable value changes, the style will be updated.

```
<script setup lang="ts">
const color = ref("red")
</script>

<template>
  <div class="text">hello</div>
</template>

<style>
.text {
  color: v-bind(color);
}
</style>
```

Scoped Styles

The scoped attribute allows you to style component in insolation. The styles declared with this attribute will only apply to this component.

```
<template>
  <div class="example">hi</div>
</template>

<style scoped>
.example {
  color: red;
}
</style>
```

CSS Modules

You can use CSS Modules with the module attribute. Access it with the injected `$style` variable.

```
<template>
  <p :class="$style.red">This should be red</p>
</template>

<style module>
.red {
  color: red;
}
</style>
```

Preprocessors Support

SFC style blocks support preprocessors syntax. Vite come with built-in support for .scss, .sass, .less, .styl and .stylus files without configuration. You just need to install them first, and they will be available directly in SFC with the lang attribute.

SCSS Sass LESS Stylus

SCSS

```
<style lang="scss">
  /* Write scss here */
</style>
```

You can refer to the Vite CSS docs and the @vitejs/plugin-vue docs. For webpack users, refer to the vue loader docs.

Using PostCSS

Nuxt comes with postcss built-in. You can configure it in your `nuxt.config` file.

nuxt.config.ts

```
export default defineNuxtConfig({
  postcss: {
    plugins: {
      'postcss-nested': {}
      "postcss-custom-media": {}
    }
  }
})
```

For proper syntax highlighting in SFC, you can use the postcss lang attribute.

```
<style lang="postcss">
  /* Write stylus here */
</style>
```

By default, Nuxt comes with the following plugins already pre-configured:

- postcss-import: Improves the `@import` rule
- postcss-url: Transforms `url()` statements
- autoprefixer: Automatically adds vendor prefixes
- cssnano: Minification and purge

Leveraging Layouts For Multiple Styles

If you need to style different parts of your application completely differently, you can use layouts. Use different styles for different layouts.

```
<template>
  <div class="default-layout">
    <h1>Default Layout</h1>
    <slot />
  </div>
</template>

<style>
.default-layout {
  color: red;
}
</style>
```

👉 Read more in Docs > Guide > Directory Structure > Layouts.

Third Party Libraries And Modules

Nuxt isn't opinionated when it comes to styling and provides you with a wide variety of options. You can use any styling tool that you want, such as popular libraries like UnoCSS or Tailwind CSS.

The community and the Nuxt team have developed plenty of Nuxt modules to makes the integration easier. You can discover them on the modules section of the website. Here are a few modules to help you get started:

- UnoCSS: Instant on-demand atomic CSS engine
- Tailwind CSS: Utility-first CSS framework
- Fontaine: Font metric fallback
- Pinceau: Adaptable styling framework
- Nuxt UI: A UI Library for Modern Web Apps

Nuxt modules provide you with a good developer experience out of the box, but remember that if your favorite tool doesn't have a module, it doesn't mean that you can't use it with Nuxt! You can configure it yourself for your own project. Depending on the tool, you might need to use a Nuxt plugin and/or make your own module. Share them with the community if you do!

Easily Load Webfonts

You can use the Nuxt Google Fonts module to load Google Fonts.

If you are using UnoCSS, note that it comes with a web fonts presets to conveniently load fonts from common providers, including Google Fonts and more.

Advanced

Transitions

Nuxt comes with the same `<Transition>` element that Vue has, and also has support for the experimental View Transitions API.

👉 Read more in Docs > Features > Transitions.

Font Advanced Optimization

We would recommend using Fontaine to reduce your CLS. If you need something more advanced, consider creating a Nuxt module to extend the build process or the Nuxt runtime.

Always remember to take advantage of the various tools and techniques available in the Web ecosystem at large to make styling your application easier and more efficient. Whether you're using native CSS, a preprocessor, postcss, a UI library or a module, Nuxt has got you covered. Happy styling!

LCP Advanced optimizations

You can do the following to speed-up the download of your global CSS files:

- Use a CDN so the files are physically closer to your users
- Compress your assets, ideally using Brotli
- Use HTTP2/HTTP3 for delivery
- Host your assets on the same domain (do not use a different subdomain)

Most of these things should be done for you automatically if you're using modern platforms like Cloudflare, Netlify or Vercel. You can find an LCP optimization guide on web.dev.

If all of your CSS is inlined by Nuxt, you can (experimentally) completely stop external CSS files from being referenced in your rendered HTML. You can achieve that with a hook, that you can place in a module, or in your Nuxt configuration file.

nuxt.config.ts

```
export default defineNuxtConfig({
  hooks: {
    'build:manifest': (manifest) => {
      // find the app entry, css list
      const css = manifest['node_modules/nuxt/dist/app/entry.js']?.css
      if (css) {
        // start from the end of the array and go to the beginning
        for (let i = css.length - 1; i >= 0; i--) {
          // if it starts with 'entry', remove it from the list
          if (css[i].startsWith('entry')) css.splice(i, 1)
        }
      }
    },
  },
})
```

Edit on Github