

# Export sitemap

📁 My New Project

🕒 3 October 2024




# Project Info

CSS uses a global namespace for CSS Selectors that can easily result in style conflicts throughout your application when building an application using modern web components. You can avoid this problem by nesting CSS selectors or use a styling convention like BEM but this becomes complicated quickly and won't scale.

CSS-in-JS avoids these problems entirely by generating unique class names when styles are converted to CSS. This allows you to think about styles on a component level without worrying about styles defined elsewhere.

In this course, you will learn how to express popular SCSS (Sass) language features using latest JavaScript features. We will convert simple examples from SCSS to CSS-in-JS. As a designer or (S)CSS developer, you should be able to follow without extensive JavaScript knowledge, understanding SCSS is required though. We will not be using any particular CSSinJS libraries. Instead, we will focus on a base knowledge you need later if you use any CSSinJS library.

## Learner Reviews

- How will you use what you learned from this course?  
the javascript concepts are useful, but not sure about implementing a css framework inside javascript is  
 **technokon** 5 years ago
- How will you use what you learned from this course?  
Nice usage JS for CSS but little chaotic for me and would be better great with more examples :)  
 **Kamil L** 5 years ago
- What would make this course a 7 for you?  
Maybe the pronunciation could be better, and the speed of explanations could be a little bit slower.  
But I liked it very much, thanks !  
 **Erica** 5 years ago

- 

What did you like about this course?

Very crisp and straight to the point. I was struggling to understand need for CSS Loaders in any project. This tiny powerful course resolved my queries.

 **Varun Goel** 6 years ago

- 

How will you use what you learned from this course?

Thanks for this course, Oleg. The pace of the course was perfect since I already know Javascript. But I feel like it would be good to have a 'where to go next' video at the end. We currently use a SASS/BEM solution to create a monolithic global CSS file for a React/Redux app and would like some guidance on how to go about converting to a CSS-in-JS approach with some hints on the pros and cons of this approach.

 **Jamie** 6 years ago

- 

What did you like about this course?

Great stuff. And great form of each course whose each lesson starts with 'We have two buttons'... All in all, very useful stuff. Thank you!

 **Artificial Labs** 6 years ago

# Structure

New page	7
Second page <span>NEW LABEL</span> <span>IN PROGRESS</span> <span>LABEL</span>	9
Blog page 2	11
new	12
Blog page <span>IN PROGRESS</span>	13
rrr <span>NEW LABEL</span>	14

## Separate Pages

Separete <span>LABEL</span>	15
-----------------------------	----

# Estimation

Item	Rate	Hours	Amount
QA	11 USD	20h	220 USD
Design	12 USD	3h	36 USD
Copyright	13 USD	15h	195 USD
Development	14 USD	400h	5,600 USD
New Cost	0 USD	0h	0 USD
Additional Cost			Amount
Domain Purchase			220 USD
Total:		438h	6,271 USD

# New page / Description



35H / 415 USD

## `alwaysStrict`

**Рекомендован: всегда / сложность: легко.**

Флаг `alwaysStrict` включает добавление строки `"use strict"` в каждый скомпилированный файл. Другими словами, `alwaysStrict` включает строгий режим JavaScript и никак не связан с проверкой типов TypeScript.

## `strict`

**Рекомендован: всегда / сложность: сложно / может быть заменён набором других флагов.**

Флаг `strict` напрямую связан с проверкой типов. Его включение автоматически активирует абсолютно все флаги секции `Strict Checks`, включая и `alwaysStrict`. Это именно то, о чём я говорил в самом начале.

У такого подхода есть как минимум один недостаток – неочевидность.

Устанавливая `strict: true`, нет наглядного представления, какие именно проверки включены и какие опции вообще существуют. Для проектов, которые с самого начала пишутся на TypeScript это не так принципиально, как для проектов, которые поэтапно портируются с JavaScript.

В процессе портирования существующего приложения нет возможности сразу включить все проверки. Приходится активировать их по одной шаг за шагом.

Иногда даже случаются сложности, из-за которых приходится откатывать ранее установленные флаги обратно в `false`.

Есть небольшая особенность работы флага `strict` – список подконтрольных ему флагов может пополняться по мере выхода новых версий TypeScript. Подобные моменты если случаются, то редко и всегда освещаются в `release notes` если, конечно, вы их читаете перед обновлением версии.

# New page / Content



35H / 415 USD

`options.detached#`

Added in: v0.7.10

On Windows, setting `options.detached` to `true` makes it possible for the child process to continue running after the parent exits. The child will have its own console window.

Once enabled for a child process, it cannot be disabled.

On non-Windows platforms, if `options.detached` is set to `true`, the child process will be made the leader of a new process group and session. Child processes may continue running after the parent exits regardless of whether they are detached or not.

See `setsid(2)` for more information.

By default, the parent will wait for the detached child to exit. To prevent the parent from waiting for a given subprocess to exit, use the `subprocess.unref()` method. Doing so will cause the parent's event loop to not include the child in its reference count, allowing the parent to exit independently of the child, unless there is an established IPC channel between the child and the parent.

When using the detached option to start a long-running process, the process will not stay running in the background after the parent exits unless it is provided with a `stdio` configuration that is not connected to the parent. If the parent's `stdio` is inherited, the child will remain attached to the controlling terminal.

Example of a long-running process, by detaching and also ignoring its parent `stdio` file descriptors, in order to ignore the parent's termination:



## Compiling an npm package

npm packages are distinguished from applications by defining package entry-points in `package.json`.

`pkgroll` is the recommended bundler for projects using `tsx`. It's developed by the same author and used to compile `tsx`.

Given your source files are in the `src` directory, it automatically infers how to build your package based on the entry points defined in `package.json` by outputting them to the `dist` directory.

## Second page / Content



0H / 0 USD



NEW LABEL

IN PROGRESS

LABEL

`options.detached#`

Added in: v0.7.10

On Windows, setting `options.detached` to `true` makes it possible for the child process to continue running after the parent exits. The child will have its own console window.

Once enabled for a child process, it cannot be disabled.

On non-Windows platforms, if `options.detached` is set to `true`, the child process will be made the leader of a new process group and session. Child processes may continue running after the parent exits regardless of whether they are detached or not.

See `setuid(2)` for more information.

By default, the parent will wait for the detached child to exit. To prevent the parent from waiting for a given subprocess to exit, use the `subprocess.unref()` method. Doing so will cause the parent's event loop to not include the child in its reference count, allowing the parent to exit independently of the child, unless there is an established IPC channel between the child and the parent.

When using the detached option to start a long-running process, the process will not stay running in the background after the parent exits unless it is provided with a `stdio` configuration that is not connected to the parent. If the parent's `stdio` is inherited, the child will remain attached to the controlling terminal.

Example of a long-running process, by detaching and also ignoring its parent `stdio` file descriptors, in order to ignore the parent's termination:



## Compiling an npm package

npm packages are distinguished from applications by defining package entry-points in `package.json`.

`pkgroll` is the recommended bundler for projects using `tsx`. It's developed by the same author and used to compile `tsx`.

Given your source files are in the `src` directory, it automatically infers how to build your package based on the entry points defined in `package.json` by outputting them to the `dist` directory.



# Second page / SEO

\$

0H / 0 USD

NEW LABEL

IN PROGRESS

LABEL

Meta Description

Main

## Blog page 2 / Content



0H / 0 USD

`options.detached#`

Added in: v0.7.10

On Windows, setting `options.detached` to `true` makes it possible for the child process to continue running after the parent exits. The child will have its own console window.

Once enabled for a child process, it cannot be disabled.

On non-Windows platforms, if `options.detached` is set to `true`, the child process will be made the leader of a new process group and session. Child processes may continue running after the parent exits regardless of whether they are detached or not.

See `setsid(2)` for more information.

By default, the parent will wait for the detached child to exit. To prevent the parent from waiting for a given subprocess to exit, use the `subprocess.unref()` method. Doing so will cause the parent's event loop to not include the child in its reference count, allowing the parent to exit independently of the child, unless there is an established IPC channel between the child and the parent.

When using the detached option to start a long-running process, the process will not stay running in the background after the parent exits unless it is provided with a `stdio` configuration that is not connected to the parent. If the parent's `stdio` is inherited, the child will remain attached to the controlling terminal.

Example of a long-running process, by detaching and also ignoring its parent `stdio` file descriptors, in order to ignore the parent's termination:



## Compiling an npm package

npm packages are distinguished from applications by defining package entry-points in `package.json`.

`pkgroll` is the recommended bundler for projects using `tsx`. It's developed by the same author and used to compile `tsx`.

Given your source files are in the `src` directory, it automatically infers how to build your package based on the entry points defined in `package.json` by outputting them to the `dist` directory.

# Blog page 2 / SEO



0H / 0 USD

Title

Seo title, bleat

Meta Description

Some description here

Slug

/blog

**new**

\$ OH / 0 USD

# Blog page / Content



403H / 5,636 USD



IN PROGRESS

`options.detached#`

Added in: v0.7.10

On Windows, setting `options.detached` to `true` makes it possible for the child process to continue running after the parent exits. The child will have its own console window.

Once enabled for a child process, it cannot be disabled.

On non-Windows platforms, if `options.detached` is set to `true`, the child process will be made the leader of a new process group and session. Child processes may continue running after the parent exits regardless of whether they are detached or not.

See `setsid(2)` for more information.

By default, the parent will wait for the detached child to exit. To prevent the parent from waiting for a given subprocess to exit, use the `subprocess.unref()` method. Doing so will cause the parent's event loop to not include the child in its reference count, allowing the parent to exit independently of the child, unless there is an established IPC channel between the child and the parent.

When using the detached option to start a long-running process, the process will not stay running in the background after the parent exits unless it is provided with a `stdio` configuration that is not connected to the parent. If the parent's `stdio` is inherited, the child will remain attached to the controlling terminal.

Example of a long-running process, by detaching and also ignoring its parent `stdio` file descriptors, in order to ignore the parent's termination:



## Compiling an npm package

npm packages are distinguished from applications by defining package entry-points in `package.json`.

`pkgroll` is the recommended bundler for projects using `tsx`. It's developed by the same author and used to compile `tsx`.

Given your source files are in the `src` directory, it automatically infers how to build your package based on the entry points defined in `package.json` by outputting them to the `dist` directory.

rrr



0H / 0 USD



NEW LABEL

# Separate



0H / 0 USD



LABEL