

Minimizando o tamanho do pacote

Saiba mais sobre as ferramentas que você pode usar para reduzir o tamanho do pacote.

Tamanho do pacote importa

O tamanho do pacote do Material-UI é levado muito a sério. Fotos contendo o tamanho do pacote são feitas em cada commit e partes críticas dos pacotes([veja a última foto](#)). Combinado com [dangerJS](#) podemos inspecionar [alterações detalhadas no tamanho do pacote](#) em cada solicitação de Pull Request.

Quando e como usar tree-shaking?

Tree-shaking no Material-UI funciona de uma forma moderna. MUI exposes its full API on the top-level `@mui` imports. If you're using ES6 modules and a bundler that supports tree-shaking ([webpack >= 2.x](#), [parcel](#) [with a flag](#)) you can safely use named imports and still get an optimized bundle size automatically:

```
import { Button, TextField } from '@material-ui/core';
```

⚠ As instruções a seguir são somente necessárias se você deseja otimizar o tempo de startup em desenvolvimento ou se você esta utilizando um bundler antigo que não suporte tree-shaking.

Ambiente de desenvolvimento

Os pacotes de desenvolvimento podem conter a biblioteca completa que pode deixar **o tempo de inicialização mais lento**. Isso é especialmente perceptível se você importar de `@material-ui/icons`. Os tempos de inicialização podem ser aproximadamente 6 vezes mais lentos do que sem utilizar importações nomeadas da API de nível superior.

Se isso é um problema para você, tem várias opções:

Opção 1

Você pode usar as importações de caminho para evitar puxar módulos não utilizados. Por exemplo, use:

```
// 🚀 Rápida
import Button from '@material-ui/core/Button';
import TextField from '@material-ui/core/TextField';
```


em vez de importações de nível superior (sem um plugin do Babel):

```
import { Button, TextField } from '@material-ui/core';
```

Esta é a opção que apresentamos em todas as demonstrações, pois não exige qualquer configuração. É o mais recomendável para autores de biblioteca que estendem os componentes. Vá até [Opção 2](#) para uma abordagem que produz uma melhor DX e UX.

Ao importar diretamente desta maneira, não usa as exportações do [arquivo principal do @material-ui/core](#), este arquivo pode servir como uma referência útil para quais módulos são públicos.

Esteja ciente de que apenas damos suporte para as importações de primeiro e segundo nível. Qualquer coisa em níveis mais profundos é considerado privado e pode causar problemas, como a duplicação de módulos em seu pacote.

```
//  OK
import { Add as AddIcon } from '@material-ui/icons';
import { Tabs } from '@material-ui/core';
//          ^^^ 1º ou nível superior

//  OK
import AddIcon from '@material-ui/icons/Add';
import Tabs from '@material-ui/core/Tabs';
//          ^^^ 2º nível

//  NÃO OK
import TabIndicator from '@material-ui/core/Tabs/TabIndicator';
//          ^^^^^^^^^^^ 3º nível
```

Se você estiver usando `eslint` você pode capturar esta problemática de importações com a regra [no-restricted-imports](#). A configuração `.eslintrc` a seguir irá capturar as problemáticas das importações dos pacotes `@material-ui`:

```
{
  "rules": {
    "no-restricted-imports": [
      "error",
      {
        "patterns": ["@material-ui/**/*", "!@material-ui/core/test-utils/*"]
      }
    ]
  }
}
```

Opção 2

Esta opção fornece a melhor Experiência do Usuário e Experiência do Desenvolvedor:

- UX: O plugin Babel permite tree-shaking de nível superior, mesmo se o seu bundler não suporte.
- DX: O plugin Babel torna o tempo de inicialização no modo de desenvolvimento tão rápido quanto a opção 1.
- DX: Essa sintaxe reduz a duplicação de código, exigindo apenas uma única importação para vários módulos. Em geral, o código é mais fácil de ser lido, e é menos provável que você cometa um erro ao importar um novo módulo.

```
import { Button, TextField } from '@material-ui/core';
```

No entanto, você precisa aplicar as duas etapas seguintes corretamente.

1. Configure o Babel

Escolha um dos seguintes plugins:

- [babel-plugin-import](#) with the following configuration:

```
yarn add -D babel-plugin-import
```

Crie um arquivo `.babelrc.js` no diretório raiz do seu projeto:

```
const plugins = [
  [
    'babel-plugin-import',
    {
      libraryName: '@material-ui/core',
      libraryDirectory: '',
      camel2DashComponentName: false,
    },
    'core',
  ],
  [
    'babel-plugin-import',
    {
      libraryName: '@material-ui/icons',
      libraryDirectory: '',
      camel2DashComponentName: false,
    },
    'icons',
  ],
];

module.exports = { plugins };
```

- [babel-plugin-transform-imports](#) com a seguinte configuração:

```
yarn add -D babel-plugin-transform-imports
```

Crie um arquivo `.babelrc.js` no diretório raiz do seu projeto:

```
const plugins = [
  [
    'babel-plugin-transform-imports',
    {
      '@material-ui/core': {
        transform: '@material-ui/core/${member}',
        preventFullImport: true,
      },
      '@material-ui/icons': {
        transform: '@material-ui/icons/${member}',
        preventFullImport: true,
      },
    },
  ],
];
```

```
module.exports = { plugins };
```

Se você estiver usando Create React App, você precisará usar alguns projetos que permitem a configuração por `.babelrc`, sem ejetar.

```
yarn add -D react-app-rewired customize-cra
```

Crie um arquivo `config-overrides.js` na pasta raiz:

```
/* config-overrides.js */
/* eslint-disable react-hooks/rules-of-hooks */
const { useBabelRc, override } = require('customize-cra');

module.exports = override([useBabelRc()]);
```

Se você desejar, `babel-plugin-import` pode ser configurado através de `config-overrides.js` ao invés de `.babelrc` usando esta [configuração](#).

Modifique seu comando start no `package.json`:

```
"scripts": {
  - "start": "react-scripts start",
  + "start": "react-app-rewired start",
  - "build": "react-scripts build",
  + "build": "react-app-rewired build",
  - "test": "react-scripts test",
  + "test": "react-app-rewired test",
  "eject": "react-scripts eject"
}
```

Desfrute do tempo de inicialização significativamente mais rápido.

2. Converta todas as suas importações

Finalmente, você pode converter sua base de código existente com esse [codemod top-level-imports](#). Ele executará as seguintes alterações:

```
-import Button from '@material-ui/core/Button';
-import TextField from '@material-ui/core/TextField';
+import { Button, TextField } from '@material-ui/core';
```

Pacotes disponíveis

O pacote publicado no npm é **transpilado** com [Babel](#), para levar em consideração as [plataformas suportadas](#).

⚠ Para minimizar a duplicação de código nos pacotes de usuários, autores de biblioteca são **fortemente desencorajados** a importar de qualquer um dos outros pacotes. Otherwise it's not guaranteed that dependencies used also use legacy or modern bundles. Instead, use these bundles at the bundler level with e.g [Webpack's resolve.alias](#):

```
{
  resolve: {
    alias: {
      '@mui/base': '@mui/base/legacy',
      '@mui/lab': '@mui/lab/legacy',
      '@mui/material': '@mui/material/legacy',
      '@mui/styled-engine': '@mui/styled-engine/legacy',
      '@mui/system': '@mui/system/legacy',
    }
  }
}
```

Pacote moderno

O pacote moderno pode ser encontrado sob a [pasta /modern](#) . Ela tem como alvo as versões mais recentes de navegadores evergreen (Chrome, Firefox, Safari, Edge). Isso pode ser usado para criar pacotes separados visando diferentes navegadores.

Pacote legado

Se você precisar suportar o IE 11, você não pode usar o pacote padrão ou moderno sem transpilação. No entanto, você pode usar o pacote legado encontrado sob [pasta /legacy](#) . Você não precisa de nenhum polyfill adicional.