## All Modules y Custom Strategy 17/25

RECURSOS APUNTES

Al haber activado la técnica de *Lazy Loading* , puedes **personalizar el envío de estos módulos** al cliente con diferentes estrategias.

### Cómo hacer precarga de módulos bajo demanda

Por defecto, la aplicación enviará al cliente solo el módulo que necesita. Si ingresas al módulo website, solo se cargará su respectivo archivo JS.

| Status | Method | Domain | File |
|---|---|---|---|
| 200 | GET | localhost:4200 | runtime.js |
| 200 | GET | localhost:4200 | polyfills.js |
| 200 | GET | localhost:4200 | vendor.js |
| 200 | GET | localhost:4200 | main.js |
| 200 | GET | localhost:4200 | src_app_modules_website_website_module_ts.js |

Si el usuario solicita otro módulo, este se cargará solo cuando sea necesario.

| Status | Method | Domain | File |
|---|---|---|---|
| 200 | GET | localhost:4200 | runtime.js |
| 200 | GET | localhost:4200 | polyfills.js |
| 200 | GET | localhost:4200 | vendor.js |
| 200 | GET | localhost:4200 | main.js |
| 200 | GET | localhost:4200 | src_app_modules_website_website_module_ts.js |
| 200 | GET | localhost:4200 | src_app_modules_cms_cms_module_ts.js |

Esto puede causarte problemas, ya que si el módulo solicitado es algo pesado o la conexión es lenta, tardará varios segundos en estar listo y no será buena la experiencia de usuario.

## Cómo hacer precarga de todos los módulos

Puedes decirle a tu aplicación que, por defecto, precargue todos los módulos con la siguiente configuración.

```
// app-routing.module.ts
import { NgModule } from '@angular/core';
import { RouterModule, Routes, PreloadAllModules } from '@angular/router';

const routes: Routes = [
  {
    path: '',
    loadChildren: () => import('./modules/website/website.module').then(m =>
m.WebsiteModule),
  },
  {
    path: 'cms',
    loadChildren: () => import('./modules/cms/cms.module').then(m => m.CmsModule),
  }
];
```

```
@NgModule({
  imports: [RouterModule.forRoot(routes, {
    preloadingStrategy: PreloadAllModules
  })],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Importando **PreloadAllModules** desde `@angular/router`, lo pasas como parámetro al `import` en el decorador `@NgModule()`. De esta manera, se cargarán en el primer render TODOS los módulos que tu aplicación tenga, pudiendo ver por consola algo como lo siguiente.

| Status | Method | Domain | File |
|---|---|---|---|
| 200 | GET | localhost:4200 | runtime.js |
| 200 | GET | localhost:4200 | polyfills.js |
| 200 | GET | localhost:4200 | vendor.js |
| 200 | GET | localhost:4200 | main.js |
| 200 | GET | localhost:4200 | src_app_modules_website_website_module_ts.js |
| 200 | GET | localhost:4200 | src_app_modules_cms_cms_module_ts.js |

## Pasos para una estrategia personalizada de precarga

Precargar todos los módulos a la vez, puede ser contraproducente. Imagina que tu aplicación posea 50 o 100 módulos. Sería lo mismo que tener todo en un mismo archivo `main.js`.

Para solucionar esto, puedes personalizar la estrategia de descarga de módulos indicando qué módulos si se deben precargar y cuáles no.

### 1. Agrega metadata a cada ruta

Agrégale a cada regla en el routing de tu aplicación, metadata para indicarle a cada módulo si debe ser precargado, o no.

```
// app-routing.module.ts
const routes: Routes = [
  {
    path: '',
    loadChildren: () => import('./modules/website/website.module').then(m =>
m.WebsiteModule),
    data: { preload: true },
  },
  {
    path: 'cms',
    loadChildren: () => import('./modules/cms/cms.module').then(m => m.CmsModule),
    data: { preload: true },
  }
];
```

Con la propiedad data: `{ preload: true }` , le indicas al servicio **CustomPreloadingStrategy** si el módulo debe ser precargado en el primer render de tu app.

### 2. Crea un servicio con estrategia personalizada

Crea un servicio al cual llamaremos **CustomPreloadingStrategy** con la siguiente lógica.

```
// modules/shared/services/custom-preloading-strategy.service.ts
import { Injectable } from '@angular/core';
import { Route, PreloadingStrategy } from '@angular/router';
```

```
import { Observable, of } from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class CustomPreloadingStrategyService implements PreloadingStrategy {

  preload(route: Route, load: () => Observable<any>): Observable<any> {
    if (route.data && route.data.preload)
      return load();
    else
      return of(null);
  }

}
```

El servicio implementa **PreloadingStrategy** y sobreescribiendo el método `preload()`, hace uso de la metadata para desarrollar tu propia lógica de renderizado de módulos.

### 3. Importa tu estrategia

Finalmente, importa tu estrategia personalizada en el routing.

```
// app-routing.module.ts
import { CustomPreloadingStrategyService } from './modules/shared/services/custom-preloading-strategy.service';

// ..

@NgModule({
```

```
  imports: [RouterModule.forRoot(routes, {
    preloadingStrategy: CustomPreloadingStrategyService,
  })],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

De esta manera, ya puedes personalizar qué módulos serán enviados al cliente y cuáles no, mejorando así el rendimiento de tu aplicación.

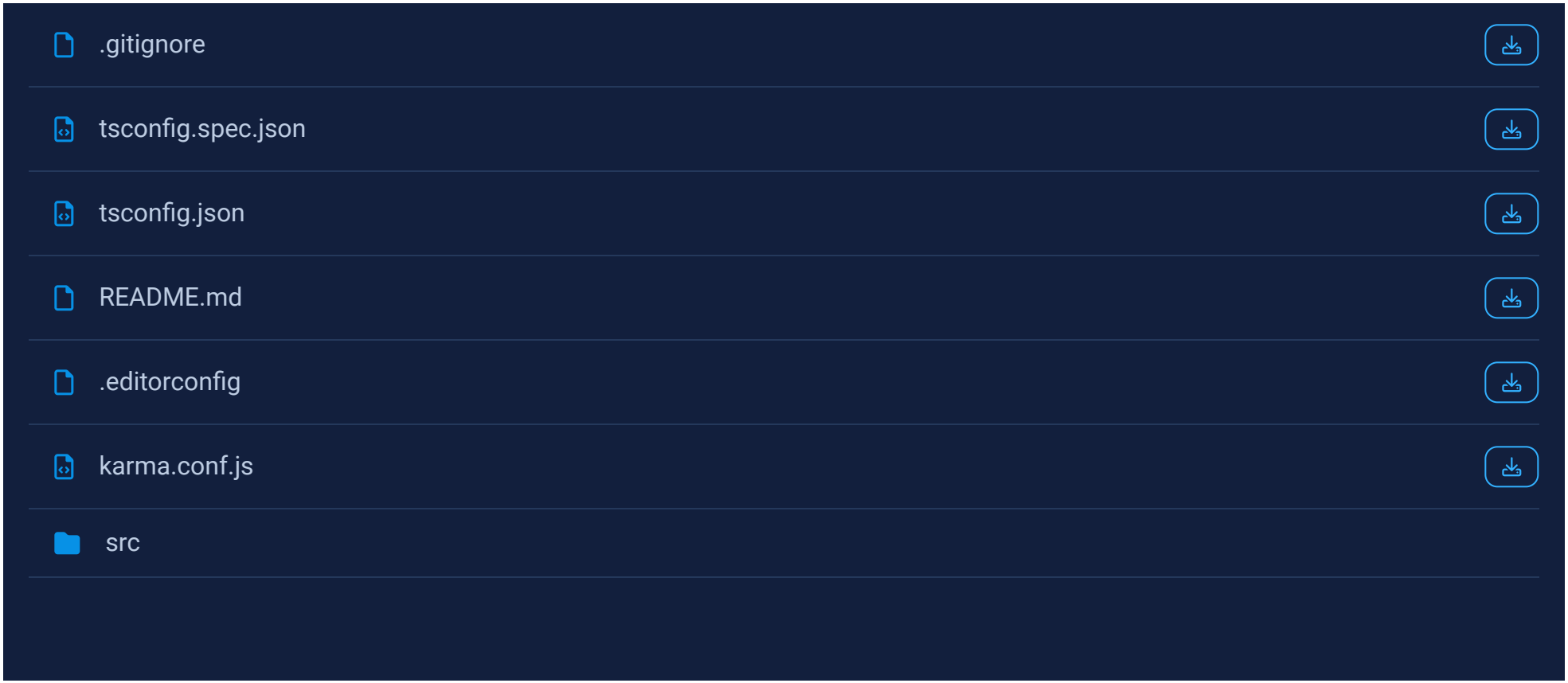
---

***Contribución creada por:*** *Kevin Fiorentino.*

## Archivos de la clase

/

- tsconfig.app.json
- package.json
- angular.json
- .eslintrc.json
- package-lock.json
- .browserslistrc


.gitignore
tsconfig.spec.json
tsconfig.json
README.md
.editorconfig
karma.conf.js
src