Super — to dokładnie ten przypadek CSP. Zrób tak:

**Co robimy**

* Generujemy **nonce per-request** w Express.
* Ustawiamy nagłówek **Content-Security-Policy** z tym nonce.
* Dla **index.html** w fallbacku SPA wstrzykujemy nonce do wszystkich <script> (i opcjonalnie <style>).
* Statyczne assety serwujemy normalnie przez express.static (żeby nie wrócił błąd „module script … text/html”).

**Minimalny server.js (pod Angular 15 + iisnode)**

// server.js

const path = require('path');

const fs = require('fs');

const crypto = require('crypto');

const express = require('express');

const app = express();

const dist = path.join(\_\_dirname, 'dist', 'your-app-name'); // <- dopasuj

// 1) Nonce per request

app.use((req, res, next) => {

res.locals.cspNonce = crypto.randomBytes(16).toString('base64');

next();

});

// 2) Statyki ZANIM fallbacku (naprawia błąd MIME)

app.use(express.static(dist, {

setHeaders: (res, filePath) => {

if (filePath.endsWith('.mjs') || filePath.endsWith('.js')) {

res.setHeader('Content-Type', 'application/javascript; charset=utf-8');

}

if (filePath.endsWith('.wasm')) {

res.setHeader('Content-Type', 'application/wasm');

}

}

}));

// 3) Fallback SPA tylko dla ścieżek bez kropki

app.get(/^\/(?!.\*\.\w+$).\*/, (req, res) => {

const indexPath = path.join(dist, 'index.html');

let html = fs.readFileSync(indexPath, 'utf8');

// Wstrzyknięcie nonce do <script> bez nonce

html = html.replace(

/<script\b(?![^>]\*\bnonce=)([^>]\*)>/gi,

`<script nonce="${res.locals.cspNonce}"$1>`

);

// (opcjonalnie) nonce także do <style>

// html = html.replace(

// /<style\b(?![^>]\*\bnonce=)([^>]\*)>/gi,

// `<style nonce="${res.locals.cspNonce}"$1>`

// );

// 4) CSP z nonce (dostosuj connect-src itd. pod swoje API)

const csp = [

`default-src 'self'`,

// nonce MA znaczenie dla skryptów inline; dla zewnętrznych nie jest wymagany.

`script-src 'self' 'nonce-${res.locals.cspNonce}' 'strict-dynamic' https: http:`,

`style-src 'self' 'unsafe-inline'`, // lub użyj nonce jak wyżej i usuń 'unsafe-inline'

`img-src 'self' data: blob:`,

`font-src 'self' data:`,

`connect-src 'self' https: http:`,

`frame-ancestors 'self'`,

`object-src 'none'`,

`base-uri 'self'`

].join('; ');

res.setHeader('Content-Security-Policy', csp);

// 5) HTML dynamiczny nie powinien być keszowany (nonce zmienia się co request)

res.setHeader('Cache-Control', 'no-store');

res.send(html);

});

// Start

const port = process.env.PORT || 3000;

app.listen(port, () => console.log(`Express on ${port}`));

**web.config (iisnode → wszystko do server.js, Express rozdaje)**

<?xml version="1.0" encoding="utf-8"?>

<configuration>

<system.webServer>

<handlers>

<add name="iisnode" path="server.js" verb="\*" modules="iisnode" />

</handlers>

<rewrite>

<rules>

<rule name="all-to-node" stopProcessing="true">

<match url=".\*" />

<action type="Rewrite" url="server.js" />

</rule>

</rules>

</rewrite>

<staticContent>

<remove fileExtension=".mjs" />

<mimeMap fileExtension=".mjs" mimeType="application/javascript" />

<remove fileExtension=".wasm" />

<mimeMap fileExtension=".wasm" mimeType="application/wasm" />

<remove fileExtension=".map" />

<mimeMap fileExtension=".map" mimeType="application/json" />

</staticContent>

<iisnode loggingEnabled="true" devErrorsEnabled="true" />

</system.webServer>

</configuration>

**Ważne uwagi**

* **Kolejność**: express.static(...) musi być **przed** fallbackiem SPA, inaczej JS dostanie index.html ⇒ błąd MIME.
* **Nonce a Angular**: nonce ma znaczenie głównie dla **inline** <script>. Możesz dodać nonce do wszystkich <script>, to nie szkodzi, ale CSP i tak musi pozwalać na ładowanie plików z self (albo użyj 'strict-dynamic' jak wyżej).
* Gdy hostujesz pod pod-ścieżką (/myapp/), **zbuduj** Angulara z --base-href /myapp/ --deploy-url /myapp/.

Jeśli chcesz, mogę dopasować powyższe pod Twoją dokładną ścieżkę dist\... i aktualne dyrektywy CSP (np. dodać domeny API do connect-src).

Jasne — tu chodzi o to, **pod jaką ścieżką URL działa aplikacja** (np. /myapp/) i jak Angular generuje/rozwiązuje adresy do bundli, assetów i routingu.

**Co robi baseHref vs deployUrl**

* **baseHref** → wpisuje do index.html tag <base href="...">. Wpływa na:
  + jak **przeglądarka** rozwiązuje **wszystkie adresy względne** w dokumencie (np. main.js, styles.css, /assets/... jeśli nie są absolutne),
  + jak działa **Angular Router** (nawigacja w SPA).
* **deployUrl** → prefix dodawany przez builder do **adresów skryptów/CSS i assetów** wstrzykiwanych do index.html oraz ładowanych dynamicznie (lazy chunks). Nie wpływa na router.

W praktyce, przy hostowaniu pod pod-ścieżką **najbezpieczniej ustawić oba** na tę samą wartość, np. /myapp/.

**Typowy objaw złego baseHref/deployUrl**

Budujesz dla root ('/'), a hostujesz pod /myapp/. W efekcie:

* index.html ma <base href="/"> **albo** skrypty wstrzyknięte są z src="/main.xxxx.js".
* Przeglądarka próbuje pobrać /main.xxxx.js (z **roota**, nie z /myapp/) → na IIS/Express trafia fallback SPA i wraca index.html z Content-Type: text/html → błąd „Expected a JavaScript-or-Wasm module… got text/html”.

**Poprawna konfiguracja builda (Angular 15)**

**Jednorazowo z linii poleceń**

ng build --configuration production --base-href /myapp/ --deploy-url /myapp/

**Na stałe w angular.json**

{

"projects": {

"your-app-name": {

"architect": {

"build": {

"configurations": {

"production": {

"baseHref": "/myapp/",

"deployUrl": "/myapp/"

}

}

}

}

}

}

}

**Uwaga:** zakończ baseHref **ukośnikiem** (/myapp/), inaczej rozwiązywanie ścieżek względnych bywa błędne.

**Checklista po buildzie**

1. Otwórz dist/your-app-name/index.html i sprawdź:

<base href="/myapp/">

1. Sprawdź, czy w <script src=...> i <link href=...> wstrzyknięte ścieżki zaczynają się od /myapp/ **lub** są względne (np. main.xxxx.js) – przy poprawnym <base> i względnych src też będzie OK.
2. W DevTools → Network:
   * żądania do main.\*.js, polyfills.\*.js, styles.\*.css, runtime.\*.js, assets/... idą na **/myapp/...**,
   * **Response** to pliki JS/CSS, a nie <!doctype html>.

**IIS / Express – co jeszcze może „skosić” ścieżkę**

* **Rewrite/fallback** w Express musi być po express.static i **tylko dla ścieżek bez kropki**:
* app.use(express.static(dist));
* app.get(/^\/(?!.\*\.\w+$).\*/, (req, res) => res.sendFile(path.join(dist,'index.html')));

Wtedy assety (\*.js, \*.css, \*.map, \*.png, \*.wasm) nie dostaną index.html.

* Jeśli to **IIS Application** pod /myapp, to Node/Express dostaje już ścieżki względne do aplikacji; po stronie klienta i tak **musisz** mieć /myapp/ w baseHref/deployUrl.
* Dodaj typy MIME (zwł. .mjs, .wasm) w web.config, żeby IIS nie bawił się w domyślne text/plain.

**Kiedy wystarczy samo baseHref**

Gdy Angular CLI w Twojej wersji wstrzykuje **ścieżki względne** (main.js, styles.css), poprawny <base href="/myapp/"> zwykle wystarczy. Jeśli jednak widzisz w index.html ścieżki absolutne od root (/main.js), dodaj **również** --deploy-url /myapp/.

**Alternatywy / obejścia**

* **Hash routing** (useHash: true) – serwer widzi tylko /myapp/, reszta po #.... Upraszcza reguły, ale mniej eleganckie URL-e.
* Jeśli środowiska różnią się ścieżką (dev: /, prod: /myapp/), zdefiniuj **osobne konfiguracje** w angular.json (np. production-myapp) i buduj odpowiednią.

Jeśli chcesz, wklej tu krótki wycinek swojego index.html (sekcja <head> ze <base> + 2–3 pierwsze <script>/<link>), a powiem, czy potrzebujesz tylko baseHref, czy też deployUrl.