To zachowanie (“module script” dostał text/html) prawie zawsze oznacza, że żądania do plików JS (np. main.xxxx.js, polyfills.xxxx.js) nie trafiają w statyczny plik, tylko są przekierowywane na index.html (fallback SPA). Najczęstsze przyczyny:

1. Kolejność middleware w Express – wildcard app.get('\*', ...) stoi **przed** express.static(...).
2. Zły base href/deployUrl dla aplikacji hostowanej pod pod-ścieżką w IIS.
3. (rzadziej) błędne reguły URL Rewrite w web.config, które przepychają **wszystko** do Node’a, a Express i tak oddaje index.html.
4. Brak/niepoprawne MIME dla nowszych rozszerzeń (np. .mjs, .wasm) – wtedy zwykle widać application/octet-stream, ale sprawdzić warto.

Poniżej minimalny, poprawny układ.

**1) Express – poprawna kolejność i fallback**

// server.js

const express = require('express');

const path = require('path');

const app = express();

// Katalog buildu Angulara (dopasuj nazwę projektu)

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

// 1) Najpierw statyki

app.use(express.static(dist, {

// opcjonalnie twardsze typy dla modułów

setHeaders: (res, filePath) => {

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

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

}

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

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

}

}

}));

// 2) Potem fallback TYLKO dla ścieżek bez kropki (czyli nie pliki)

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

res.sendFile(path.join(dist, 'index.html'));

});

// Start

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

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

Najczęstszy błąd: app.get('\*', send index.html) przed app.use(express.static(...)) – wtedy każdy asset dostaje index.html (MIME text/html) i przeglądarka krzyczy.

**2) web.config dla iisnode (IIS → Node → Express)**

Dla hostowania wszystkiego przez Express (a nie przez IIS Static Content) prosta wersja:

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

<configuration>

<system.webServer>

<handlers>

<!-- iisnode obsługuje server.js -->

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

</handlers>

<rewrite>

<rules>

<!-- Przepuść wszystko do server.js; Express dalej rozdzieli -->

<rule name="NodeRoute" patternSyntax="ECMAScript" stopProcessing="true">

<match url=".\*" />

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

</rule>

</rules>

</rewrite>

<!-- Dla pewności typy MIME używane w nowych bundlach -->

<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" />

<remove fileExtension=".json" />

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

<!-- .js zwykle już jest, ale jeśli nie: -->

<!-- <remove fileExtension=".js" />

<mimeMap fileExtension=".js" mimeType="application/javascript" /> -->

</staticContent>

<security>

<requestFiltering>

<!-- pozwól na dłuższe nazwy plików z hashami -->

<requestLimits maxQueryString="32768" maxUrl="16384" />

</requestFiltering>

</security>

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

</system.webServer>

</configuration>

Uwaga: tu **wszystkie** żądania trafiają do server.js, a dalej rozstrzyga Express. Dzięki temu reguły SPA (fallback) i serwowanie statyk mamy w jednym miejscu (Express), a nie “pół na pół” (IIS + Express), co często rodzi konflikty.

**3) Jeśli hostujesz pod pod-ścieżką (IIS “Application”)**

Jeżeli aplikacja działa np. pod https://example.com/myapp/, musisz:

* Zbudować Angulara z właściwą bazą:
* ng build --base-href /myapp/ --deploy-url /myapp/
* Upewnić się, że w index.html masz <base href="/myapp/">.
* W Express **nie zmieniasz** nic (statyki i fallback działają względem rzeczywistych ścieżek).

Objaw źle ustawionego base href: żądania idą na /main.xxxx.js zamiast /myapp/main.xxxx.js, Express zwraca fallback index.html → MIME text/html.

**4) Szybka checklista diagnostyczna**

1. W DevTools → Network kliknij jeden z błędnych \*.js/\*.mjs.
   * Jeśli **Response** zaczyna się od <!doctype html> lub <html>... → to jest index.html → problem z kolejnością middleware / base-href / regułami.
2. Sprawdź server.js:
   * app.use(express.static(...)) **musi** być **przed** fallbackiem.
   * Fallback niech **ignoruje** żądania z kropką: ^/(?!.\*\.\w+$).\*
3. Jeśli używasz .wasm/.mjs: dodaj MIME (jak wyżej).
4. Gdy aplikacja jest pod wirtualnym katalogiem: przebuduj z --base-href i --deploy-url.

**5) Minimalny przykład „działa od razu”**

1. Build:

ng build --configuration production

1. Struktura:

site\

web.config

server.js

dist\your-app-name\index.html, main.x.js, ...

1. IIS: Site wskazuje na site\. iisnode 0.2.5 zainstalowany. Otwierasz URL – powinno działać bez błędów MIME.

Jeśli po tych zmianach nadal dostajesz ten sam błąd, podeślij proszę (tu wklej):

* fragment z DevTools → Network → Response jednego z problematycznych \*.js,
* Twój server.js,
* web.config,
* adres ścieżki aplikacji w IIS (czy to root, czy np. /myapp).

Na tej podstawie wypunktuję dokładną linię, która psuje routing.