ChatGPT

Analiza problema i preporuke za rješenje problema Thesara platforme

Sažetak izvještaja

U izvještaju se opisuje kako **Thesara** platforma trenutno ima dva načina posluživanja mini-aplikacija:

1. **Stari projekt** – statičke datoteke se poslužuju preko `@fastify/static`. Jedan handler automatski prepoznaje MIME tipove, poziva `setHeaders` callback za CSP/CORS i koristi relativne putanje. Mini-aplikacije se u `iframe` učitavaju direktno sa `/builds/:buildId/build/` i sve radi.
2. **Novi projekt** – uveden je kompleksan routing (`/:listingId/build/*` → `buildAlias`) koji iz `listingId` izvlači `buildId` i zatim „injectira" zahtjev na `/builds/:buildId/build/*`. Isti server registrira više hook-ova (`onRequest`, `setHeaders`), pokušava prebrisati `Content-Type` za `.js` datoteke i postavlja CSP/CORS u više mjesta.

U novoj arhitekturi pojavljuju se sljedeći problemi:

- **Pogrešan MIME tip** za `app.js` – browser očekuje `application/javascript`, ali Fastify vraća `application/json`, zbog čega ES-moduli ne prolaze strogu provjeru MIME tipa [1] i rezultiraju greškom **NS_ERROR_CORRUPTED_CONTENT**. Pogrešno određivanje extensiona nastaje jer `@fastify/static` koristi `mime-types` biblioteku te backslash (`\`) u Windows putanjama i upitni parametri (`?v=123`) zbunjuju detekciju.
- **CORS greške** – za statičke assete server ne vraća `Access-Control-Allow-Origin`. Module skripte se obrađuju kao CORS zahtjev; ako odgovor ne sadrži `Access-Control-Allow-Origin`, browser blokira učitavanje modula [2]. U staroj verziji uvijek se postavlja CORP i CORS header, dok nova uvjetno postavlja CORS samo kad je prisutan `Origin` header.
- **CSP blokiranje** – generirani Content-Security-Policy ne uključuje CDN domene (`esm.sh`, `cdn.jsdelivr.net`) pa se moduli React i React-DOM blokiraju. Prestroga pravila networkPolicy u manifestu (`network: "isolated"`) još više ograničavaju mrežne zahtjeve.
- **Kompleksni hook-ovi** – `onRequest` hook čita `.js` datoteke, ali `fastify-static` može prebrisati zaglavlja. `setHeaders` callback se poziva nakon što je Fastify već postavio `Content-Type`, pa ga nije moguće nadjačati.

Analiza uzroka i iskustva iz literature

Ispravni MIME tipovi za module

MDN u članku o JavaScript modulima naglašava da server mora poslužiti module s MIME tipom koji sadrži **JavaScript**; u suprotnom browser izbacuje strogu grešku MIME-tipa [1]. U drugom članku MDN upozorava da pogrešna konfiguracija MIME tipova može dovesti do toga da browser pogrešno tumači sadržaj, te

preporučuje da se za skripte koristi `text/javascript` ili `application/javascript` [3] [4]. Trenutni server vraća `application/json`, što narušava ove smjernice.

CORS kao preduvjet za module

Skripte tipa `module` podliježu CORS pravilima – ako zahtjev dođe s drugog origin-a ili iz `iframe`-a, server mora vratiti zaglavlje **Access-Control-Allow-Origin**, inače browser blokira skriptu [2]. Najjednostavniji način za dopuštanje pristupa javnim statičkim datotekama je `Access-Control-Allow-Origin: *`, što prema MDN-u znači da resurs može biti dohvaćen s bilo kojeg origin-a [5]. Za API pozive koji zahtijevaju autentikaciju treba navesti točan origin.

Cross-Origin-Resource-Policy

MDN objašnjava da zaglavlje **Cross-Origin-Resource-Policy** definira tko smije učitavati resurs: opcija `cross-origin` dopušta da resursi budu učitani s bilo koje stranice, dok `same-origin` i `same-site` ograničavaju pristup [6]. Za statičke assete koje treba učitavati unutar `iframe`-a preporučuje se `cross-origin`.

Web-socketi i sobe (real-time sesije)

Za implementaciju soba/sesija prikladno je koristiti **WebSocket**. LogRocket tutorijal pokazuje kako `fastify-websocket` omogućuje rukovanje WebSocket događajima unutar Fastify routera te kombiniranje s `fastify-static` za posluživanje klijentske aplikacije [7]. U primjeru autor definira rutu `fastify.get('/chat', { websocket: true }, … )` koja: provjerava parametre, prihvaća vezu, čuva klijente u `fastify.websocketServer.clients` i emitira poruke svim povezanim korisnicima preko funkcije `broadcast()` [7]. Taj pristup se može prilagoditi za sobe: svaki `listingId` može predstavljati room, a poruke se usmjeravaju samo korisnicima u toj sobi.

Preporučena rješenja

1. Osvježavanje logike posluživanja statičkih datoteka

1. **Uklanjanje nepotrebnog routinga** – izbjegavajte `buildAlias` i `inject` jer komplicira tok i onemogućuje `fastify-static` da ispravno detektira MIME tipove. Klijentska aplikacija može sama iz `listingId` dohvatiti `buildId` preko API-ja (`/api/listings/:listingId`) i zatim učitati `iframe` sa `/builds/:buildId/build/`.
2. **Custom route prije** `fastify-static` – implementirajte rutu `GET /builds/:buildId/build/:file` koja ručno određuje MIME tip pomoću `mime-types.lookup()` i postavlja zaglavlja:

```
app.get('/builds/:buildId/build/:filename', async (req, reply) => {
  const { buildId, filename } = req.params;
  const ext = path.extname(filename);
  const mimeType = mime.lookup(ext) || 'application/octet-stream';
  const filePath = path.join(config.BUNDLE_STORAGE_PATH, 'builds', buildId,
'build', filename);
```

```
  // Postavi ispravne zaglavlja
  reply.header('Content-Type', mimeType);
  reply.header('Access-Control-Allow-Origin', '*');
  reply.header('Cross-Origin-Resource-Policy', 'cross-origin');
  // CSP se dodaje samo za index.html
  return reply.sendFile(filename, path.dirname(filePath));
});
```

Ova ruta osigurava da `.js` datoteke imaju MIME tip `application/javascript` i da su dostupne svugdje. Za `index.html` i dalje koristite `fastify-static` uz callback `setHeaders` koji generira CSP na temelju `manifest.networkPolicy`. 3. **Normalizacija putanja** – prije poziva `mime.lookup()` koristite `path.posix.join()` ili zamijenite `\` s `/` (Windows support), te uklonite query parametre iz imena datoteke. Time izbjegavate `mime-types` bugove.

2. Uskladiti sigurnosna zaglavlja prema svjetskim standardima

1. **Content-Security-Policy** – iz manifesta izvucite dopuštene domene (`allowedDomains`, `allowCdn`, `allowInlineStyles`) i generirajte CSP npr.: `default-src 'self'; script-src 'self' https://esm.sh https://cdn.jsdelivr.net; connect-src 'self' https://api.example.com; style-src 'self' 'unsafe-inline';` tako da se dopušta uvoz Reacta/ReactDOM-a preko CDN-ova. Prestroge postavke (npr. `network: isolated`) treba izbjeći ako aplikacija treba vanjske module.
2. **CORS** – za javne statičke datoteke (`/builds/*` i `/shims/*`) postaviti `Access-Control-Allow-Origin: *`, jer su moduli dostupni svima [5]. Za API pozive koji zahtijevaju autentikaciju specificirati dopuštene origin-e i dodati `Access-Control-Allow-Credentials: true` prema potrebi.
3. **Cross-Origin-Resource-Policy** – postaviti `Cross-Origin-Resource-Policy: cross-origin` na sve statičke assete kako bi ih se moglo učitavati unutar `iframe`-a [6].
4. **Opcijski:** koristiti **OWASP Secure Headers** preporuke – implementirati `Referrer-Policy`, `X-Content-Type-Options: nosniff`, `Permissions-Policy` itd. kako bi se dodatno poboljšala sigurnost.

3. Arhitektura za sobe / sesije

1. **WebSocket server** – koristite `fastify-websocket` ili `socket.io` za real-time komunikaciju. U primjeru LogRocket-a server registrira WebSocket rutu i čuva klijente u `fastify.websocketServer.clients` te emitira poruke svima [7]. Slično možete kreirati rutu `/play/:roomId` gdje `roomId` predstavlja sobu. Kod povezivanja, klijent šalje `roomId` i `userId`; server dodaje WebSocket vezu u skup klijenata te prilikom događaja (npr. izmjena rezultata ili state aplikacije) šalje poruku samo korisnicima u toj sobi.
2. **Pohrana stanja** – iskoristite postojeći **storage API** za perzistenciju; WebSocket poruke mogu biti samo notifikacije da se podatak promijenio, a klijent preko API-ja dohvaća novi state. To smanjuje opterećenje na websocket serveru i omogućuje horizontalno skaliranje.
3. **Autentikacija i autorizacija** – prilikom uspostave WebSocket veze provjerite token korisnika i dopuštenja (hook `preValidation` kao u primjeru [7]). To sprječava neautorizirane korisnike da pristupe sobama.

4. Dugoročna održiva rješenja

1. **Reverse proxy / CDN** – razmotrite korištenje **Nginx** ili **CDN** (Cloudflare, AWS CloudFront) za posluživanje statičkih asseta. Nginx može pouzdano detektirati MIME tip po ekstenziji, postaviti `Content-Type`, `Access-Control-Allow-Origin` i `Cross-Origin-Resource-Policy` te keširati datoteke. CDN omogućuje globalno keširanje i smanjuje latenciju.
2. **Centralizacija security logike** – kreirajte middleware koji dodaje CSP, CORS i CORP zaglavlja na svim mjestima umjesto dupliciranja logike. To smanjuje rizik od inkonzistentnosti i olakšava održavanje.
3. **Automatsko testiranje** – implementirajte end-to-end testove koji provjeravaju da `app.js` vraća `Content-Type: application/javascript`, da CSP sadrži potrebne domene i da statički asseti imaju ispravna CORS/CORP zaglavlja. Unit testovi mogu simulirati moduli i detektirati pogreške prije produkcije.
4. **Monitoriranje performansi** – dodajte hook `onResponse` koji logira URL, status kod, `Content-Type` i vrijeme odaziva. To omogućuje brzu dijagnostiku u slučaju degradacije performansi.

Projekcija rezultata

Implementacijom predloženih mjera Thesara platforma bi mogla postići:

- **Pouzdano učitavanje mini-aplikacija** – ispravan MIME tip i CORS/CORP zaglavlja eliminiraju greške "disallowed MIME type" i "CORS request did not succeed"; korisnici će moći pokrenuti aplikacije bez ručnog prilagođavanja.
- **Poboljšana sigurnost** – dinamički generirani CSP smanjuje rizik od XSS i ograničava mrežne veze na dopuštene domene; dodatni sigurnosni headeri slijede preporuke OWASP-a.
- **Scalabilnost i performanse** – statički asseti posluženi preko Nginx-a ili CDN-a smanjuju opterećenje API servera i ubrzavaju učitavanje. Modularna arhitektura omogućuje jednostavno horizontalno skaliranje.
- **Korisničke sobe** – uvođenjem WebSocket-a korisnici mogu istovremeno koristiti istu mini-aplikaciju u zasebnim sobama, primjerice zajednički rješavati zadatke ili igrati igru. Broadcast logika iz primjera [7] omogućit će sustav bilježenja rezultata za sve igrače.
- **Jednostavnija održivost** – centralizirana konfiguracija zaglavlja i jasna razdvojenost servisa (API vs. statički sadržaj) olakšavaju buduće nadogradnje. Normalizacijom putanja i jasnim routingom kod će biti portabilniji (nema Windows backslash problema) i pogodniji za više okruženja.

Zaključak

Temeljni uzrok blokada u Thesari je pogrešan `Content-Type` i neadekvatno postavljen CORS/CORP, a ne nužno sam `fastify-static`. Uvođenjem pojednostavljenog routinga, ručnom normalizacijom putanja i ispravnim zaglavljima u skladu s MDN-om i OWASP-om platforma može brzo otkloniti greške. Dugoročno, prelazak na Nginx/CDN za statičke assete te integracija WebSocket-a za sobe pružit će skalabilnu i sigurnu arhitekturu koja zadovoljava moderne web standarde.

---

[1] JavaScript modules - JavaScript | MDN
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Modules

2 Cross-origin module scripts require CORS response headers | Stefan Judis Web Development
https://www.stefanjudis.com/today-i-learned/cross-origin-module-scripts-require-cors-response-headers/

3 4 Properly configuring server MIME types - Learn web development | MDN
https://developer.mozilla.org/en-US/docs/Learn_web_development/Extensions/Server-side/Configuring_server_MIME_types

5 Cross-Origin Resource Sharing (CORS) - HTTP | MDN
https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/CORS

6 Cross-Origin-Resource-Policy (CORP) header - HTTP | MDN
https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers/Cross-Origin-Resource-Policy

7 Using WebSockets with Fastify - LogRocket Blog
https://blog.logrocket.com/using-websockets-with-fastify/