

웹 페이지 성능 분석 프로그램

사용자 경험을 위한 시각화 도구



인식의 격차: 개발 환경 vs. 실제 사용자 환경

대부분의 개발자는 고성능 장비와 최적화된 네트워크 환경에서 웹 페이지를 테스트합니다. 이로 인해 실제 사용자가 겪는 느린 로딩 경험을 과소평가하기 쉽습니다. 우리의 프로그램은 이러한 인식의 격차를 해소하는 것에서 시작합니다.

개발자 환경

최적의 네트워크와 고성능 CPU/GPU. 로딩 속도 **매우 빠름**.
병목 현상 감지 어려움.

실제 사용자 환경

다양한 사양의 장비와 불안정한 네트워크.
상황에 따라 로딩 속도의 편차가 큼.
특히 모바일 환경에서 성능 저하 심각.

Fast Loading vs Load vs. Slow Day



프로그램 핵심 기능: 다차원적 분석

본 프로그램은 개발자, UX 디자이너, 비기술 기획자 모두에게 웹 페이지 로딩의 모든 과정을 투명하게 이해할 수 있도록 세 가지 핵심 분석 기능을 제공합니다.



1. 프레임 별 로딩 분석

사용자에게 실제로 화면이 그려지는 과정을 프레임 단위로 캡처하고 재생하여, 렌더링 시점의 시각적 변화를 명확하게 보여줍니다.



2. 네트워크 요청 타임라인 시각화

스크립트, 스타일시트, 이미지 등 모든 리소스의 요청 및 응답 시간을 상세하게 기록합니다. 이를 통해 병목 현상을 유발하는 리소스를 쉽게 식별할 수 있습니다.



3. 로딩 분포 테스트

다양한 네트워크 속도(3G, 4G, 와이파이) 및 장비 사양을 시뮬레이션하여, 사용자 그룹별 로딩 시간 분포를 예측하고 테스트 합니다.

작동 원리: 데이터 수집부터 시각화까지

웹 페이지 로딩 과정은 요청(Request), 응답(Response), 렌더링(Rendering)의 복잡한 순서로 이루어집니다. 우리의 프로그램은 이 모든 과정을 단계별로 포착하여 직관적인 흐름도로 보여줍니다.



요청 (Request)

응답 (Response)

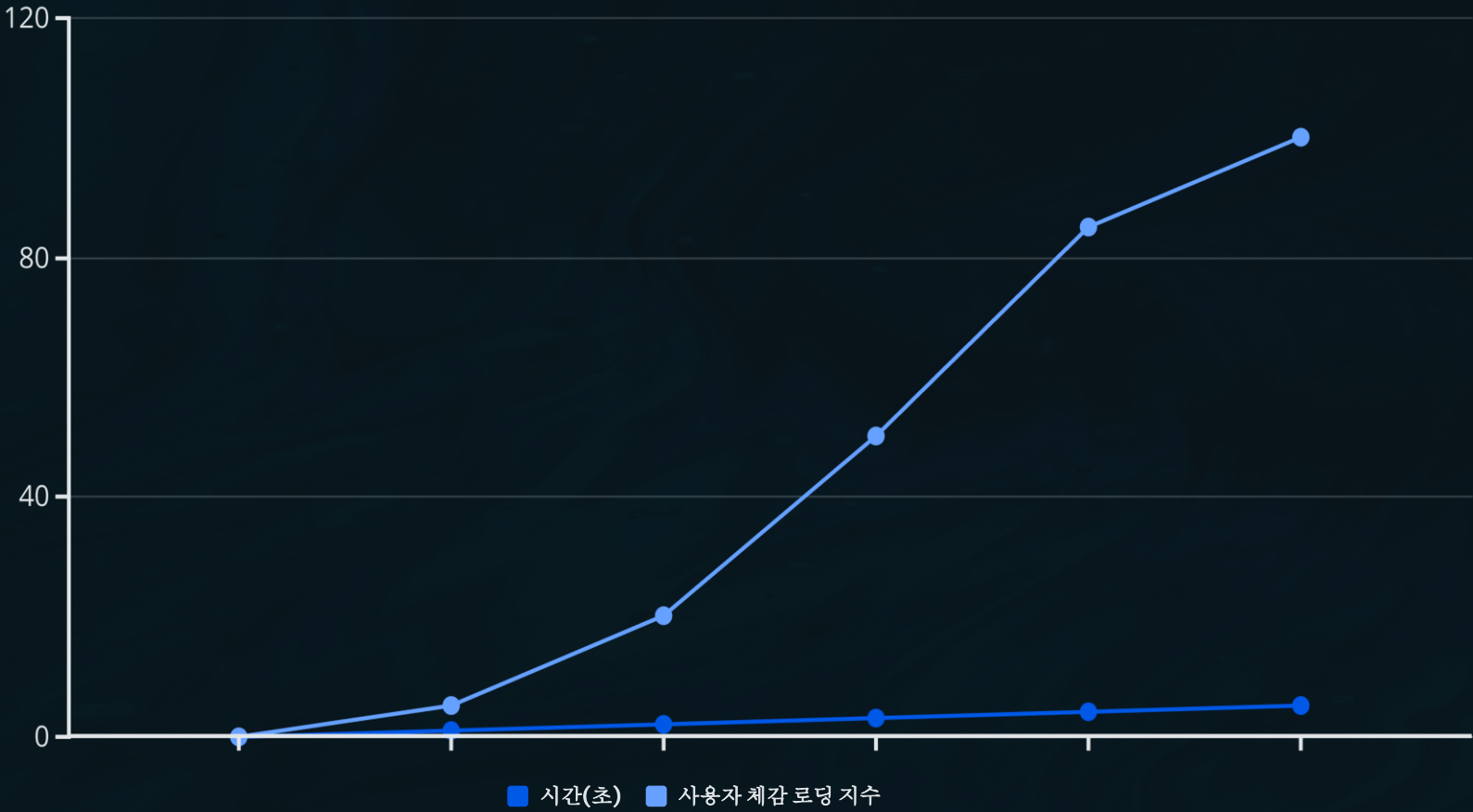
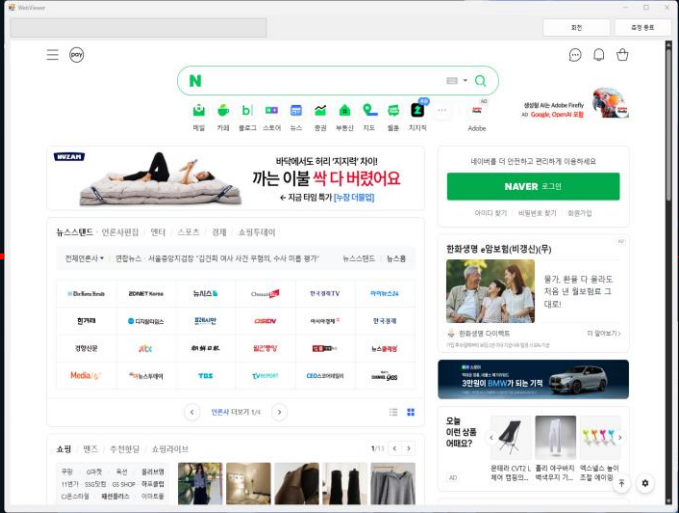
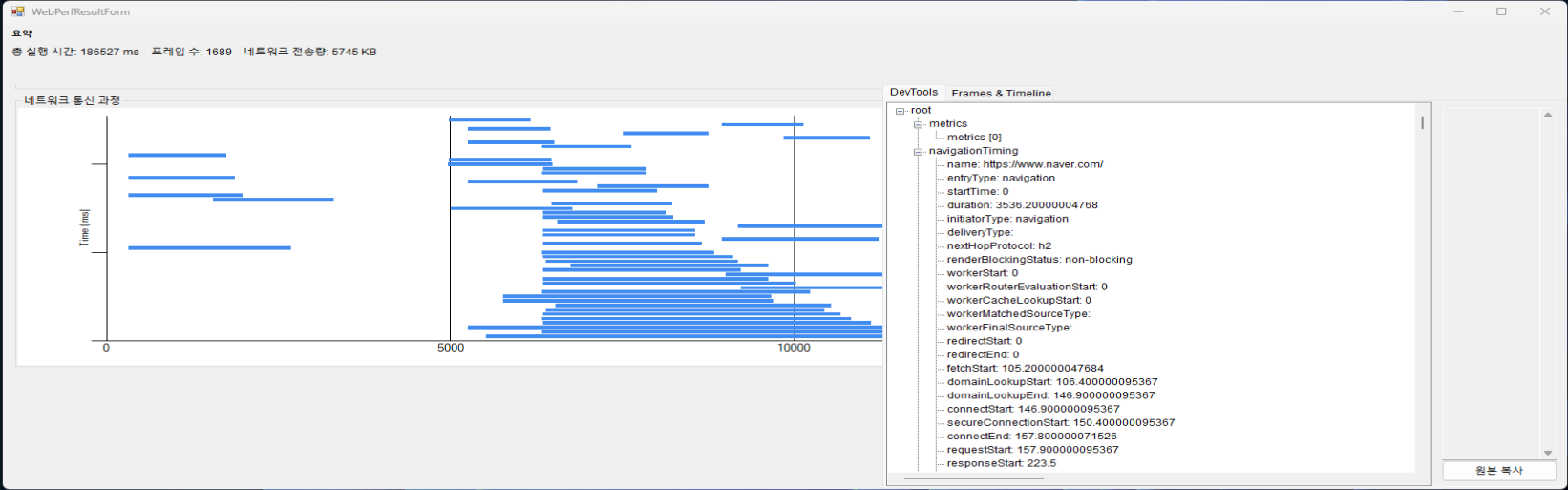
렌더링 (Rendering)

- 요청 단계: 사용자 상호작용 발생 시점과 서버에 데이터 요청이 전달되는 시간을 측정합니다.
- 응답 단계: 서버의 처리 시간과 리소스 다운로드 속도를 분석하여 초기 응답 시간을 파악합니다.
- 렌더링 단계: 브라우저가 화면 요소를 그리는 시점(First Contentful Paint, Largest Contentful Paint)을 정밀하게 기록합니다.

실제 시연: 로딩 과정의 시각적 해부

아래 시연 캡처는 일반적인 웹 페이지가 저속 네트워크 환경에서 로딩될 때 발생하는 시각적 변화와 리소스 로딩 순서를 보여줍니다. 개발 환경에서는 볼 수 없었던, 사용자 관점의 미세한 지연과 끊김을 확인하세요.

로딩 속도 시각화 차트는 리소스 로딩의 우선순위가 올바르게 설정되었는지, 불필요한 지연 요소는 없는지 판단하는 데 결정적인 정보를 제공합니다.



Web UI Performance Tester

주소

https://naver.com

기기 환경

네트워크 환경

기기 성능

네트워크 Cache 사용

측정 시작

Release

runtimes

Microsoft.Web.WebView2

Microsoft.Web.WebView2

Microsoft.Web.WebView2

Microsoft.Web.WebView2

Microsoft.Web.WebView2

Microsoft.Web.WebView2

Newtonsoft.Json.dll

Newtonsoft.Json.xml

WebPerf.exe

WebPerf.exe.config

WebPerf.pdb

2025-09-29 오전 6:36

2025-09-29 오전 6:17

2025-09-29 오전 6:36

2025-09-29 오전 6:17

2025-09-16 오후 5:04

2025-09-16 오후 5:00

2025-10-22 오전 11:16

2025-10-20 오전 9:03

2025-10-22 오전 11:16

응용 프로그램 확장

Microsoft Edge H...

응용 프로그램 확장

Microsoft Edge H...

응용 프로그램 확장

Microsoft Edge H...

응용 프로그램

CONFIG 파일

PDB 파일

635KB

567KB

39KB

41KB

81KB

140KB

705KB

717KB

56KB

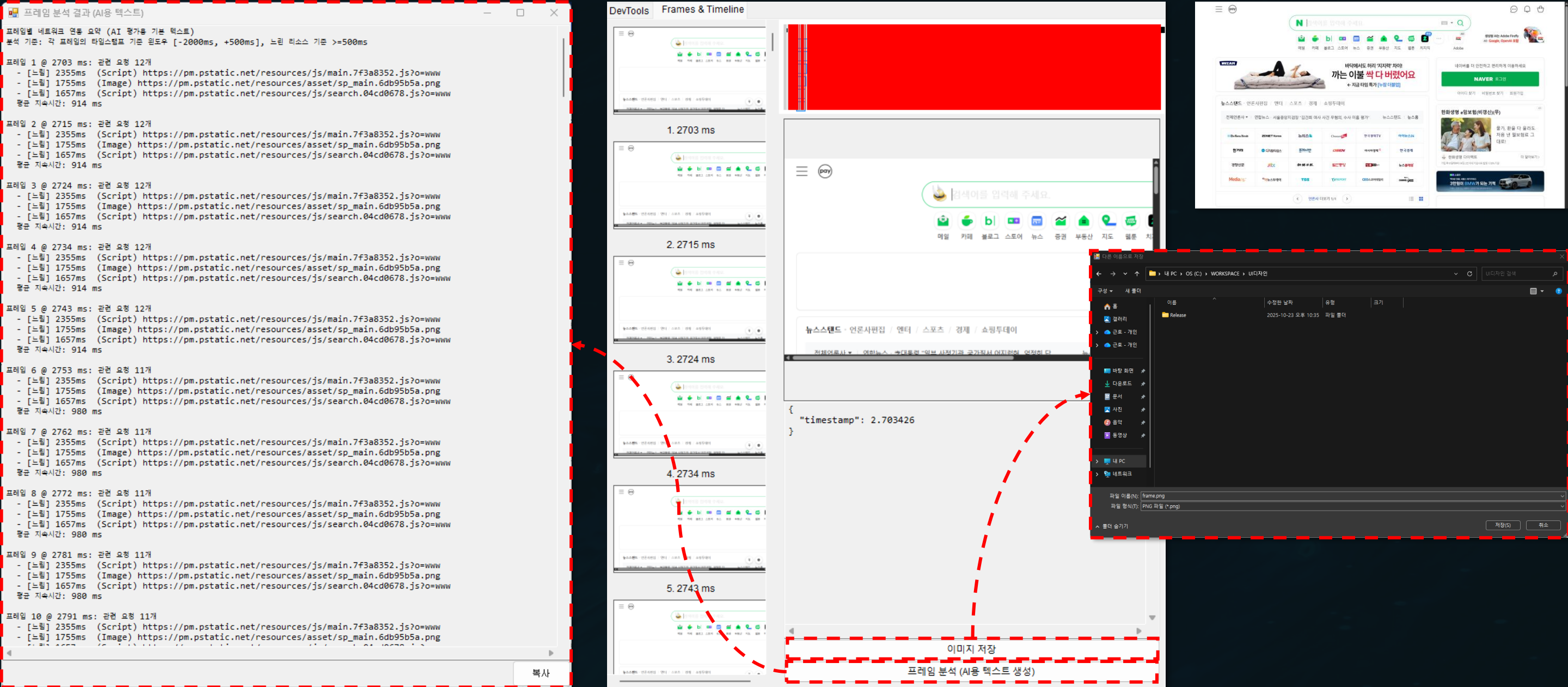
1KB

110KB

실제 시연: 로딩 과정의 시각적 해부

아래 시연 캡처는 일반적인 웹 페이지가 저속 네트워크 환경에서 로딩될 때 발생하는 시각적 변화와 리소스 로딩 순서를 보여줍니다. 개발 환경에서는 볼 수 없었던, 사용자 관점의 미세한 지연과 끊김을 확인하세요.

로딩 속도 시각화 차트는 리소스 로딩의 우선순위가 올바르게 설정되었는지, 불필요한 지연 요소는 없는지 판단하는 데 결정적인 정보를 제공합니다.



실제 시연: ChatGPT 프롬프트 질의 결과

ChatGPT 프롬프트를 활용한 분석 질의를 통해 정량적 데이터 기반의 성능 진단 결과를 도출하였으며, 단순 관찰이 아닌 AI 분석 모델을 통한 지연 원인 추론과 개선 방향을 체계적으로 제시하였습니다.

https://pm.pstatic.net/resources/js/main.7f3a8352.js?o=www

- [논팀] 4269ms (Image)
https://pm.pstatic.net/resources/asset/sp_main.6db95b5a.png
- [논팀] 3934ms (Script)
https://pm.pstatic.net/resources/js/search.04cd0678.js?o=www
평균 지속시간: 2814 ms

프레임 41 @ 4704 ms: 관련 요청 10개

- [논팀] 5561ms (Script)
https://pm.pstatic.net/resources/js/main.7f3a8352.js?o=www
- [논팀] 4269ms (Image)
https://pm.pstatic.net/resources/asset/sp_main.6db95b5a.png
- [논팀] 3934ms (Script)
https://pm.pstatic.net/resources/js/search.04cd0678.js?o=www
평균 지속시간: 2814 ms

전체 요약:

검사된 프레임 수: 41
검사된 관련 요청 총합: 469
전체 평균 요청 지속시간: 2456 ms (대략)

관망: 이 텍스트를 AI에 입력하여 UI/UX 평가(지연 원인 가설, 리소스 우선순위 제한 등)를 요청하십시오.

16s 동안 생각함 >

요청 없이도 핵심까지 바로 갑니다. 아래는 주어진 로드를 통한 사실 기반 진단·가설·우선순위 제언입니다. 과정이나 추측은 배제했고, 불확실한 부분은 명시했습니다.

Ask ChatGPT

웹사이트 성능을 공유할까요?

타인이 채팅할 수 있는 시론을 생성합니다

요청 없이도 핵심까지 바로 갑니다. 아래는 주어진 로그만으로 도출한 사실 기반 진단·가설·우선순위 제안입니다. 과장이나 추측은 배제했고, 불확실한 부분은 명시했습니다.

핵심 요약

- **병목 자원 3개 및 일관되게 지연:** `main.7f3a8352.js` (~5.6s), `search.04cd0678.js` (~3.9s), `polyfill.de9c70b5.js` (~3.8s).
- **이미지 합성의 추가 병목:** `sp_main.6db95b5a.png` (~4.3s)가 3.1s 이후 프레임부터 지속 노출.
- **41 프레임/초 460 요청 기준 평균 요청 지속시간 ≈ 2.46s**로 높음. 동일 자원들이 모든 프레임의 윈도우에서 반복적으로 “느림” 기준(<500ms)을 만족.

기술적 해석(검증 필요)

1. **중복 관측:** 동일 리소스가 다수 프레임 윈도우에서 반복 포함됨. 이는 “동일 요청이 여러 번 발송”이라 단정할 수 없고, 한 번의 느린 요청이 여러 프레임 연속창에 걸쳐 보이는 것일 가능성이 큼(확실히 않음).
2. **대형 JS 번들:** `main/search/polyfill` 3종이 모두 3.8~5.6s. 네트워크 전송 지연 + 파싱/컴파일 지연이 겹친되었을 가설(확실히 않음).
3. **캐시-우선순위 문제:** 파일명 해시(`7f3a...04cd...de9c...`)로 보아 배포 커밋 무효화 체계는 있는 것으로 추정. 그러나 캐시 미스/재검증 과다 혹은 우선순위(HTTP/2/3 priority) 부적절 가능성.
4. **이미지 최적화 부족 가설:** `sp_main.6db95b5a.png` 4.3s. 스프라이트/대형 PNG일 확률. AVIF/WebP 전환-서빙 경량화로 개선 여지 큼.
5. **초기 경로 자질 파단:** 프레임 2.3~4.7s 구간에서 ... 수 밀집(14→11→10개). 초기 렌더 크리티컬 경로가 비대하다는 신호.

5. 초기 프로브 처리: 프레임 2.3~4.7s 구간에서 요청 수 밀집(14~11→10개). 초기 탠더 크리티컬
ChatGPT | 하다는 신호

개선 우선순위 (P0→P2)

P0: 즉시 영향(크리티컬 경로)

1. JS 번들 분해/지연
 - main, search, polyfill 코드 스플리팅: 최초 뷰 필수 모듈만 type="module"로 로드, 나머지는 라우트/상호작용 시 지연(import()).
 - defer(기존) + priority hint(<link rel="preload" as="script">, 필요 시 fetchpriority="high"로 최초 핵심만 고우선).
 - polyfill 조건부 로드: UA 기능 검지로 필요한 폴리필만 서비스 제공("one-big-polyfill" 금지).
2. 캐시-전송 최적화
 - 헤더 파일에 Cache-Control: public, max-age=31536000, immutable.
 - 서버/엣지에서 Brotli(br) 우선, HTTP/2 또는 HTTP/3(QUIC) 강제.
 - Preconnect(DNS+TLS) to https://pm.static.net + Early Hints(103) 지연 여부 점검.
3. 이미지 병목 제거
 - sp_main.6db95b5a.png → AVIF/WebP 파생본 서빙(accept 해더 내고) + 록-필드 적응형 **스스셋**.
 - 위 자원이 Fold 위 필수가 아니면 loading="lazy" + fetchpriority="low".

P1: 구조적 개선 4. 크리티컬 CSS 인라인(Critical CSS inlined), 비핵심 CSS는 media="print"/onload 스왑.

5. Long Task 억제: JS 실행/컴파일/실행 분할, ↓ drafter(Partial/Islands).

6. 3rd-party 스크립트 통제: 태그메니저/추적 스크립트 지연 로드 및 샌드박스.

특정 링크를 공유할까요?

타인이 채팅할 수 있는 사본을 생성합니다

즉시 확인 체크리스트 (명확/측정 가능)

- [] 위 4개 자원에 content-encoding: br, cache-control: immutable 적용 여부.
- [] TTFB vs 다운로드 시간 구분(네트워크 > 파싱?)
- [] HTTP/3 연결 성립률, 0-RTT 이득 유무.
- [] Priority 실제 적용(DevTools "Priority" 열, Resource Timing transferSize/encodedBodySize/responseStart).
- [] polyfill.*,.js 용량/모듈 수, 실제 필요한 폴리필 목록(UA 기준).
- [] sp_main.*,.png 원본 크기/용량/픽셀 차수, FMP/LCP 관련 여부.

검증 스니펫(현장 적용용)

1) 대상 4개 리소스 리소스 타이밍 델트(JS, 콘솔에서 실행)

```
Js 코드 복사  
const targets = [  
  /main\./faas85a.js/,  
  /script\./d6c0d97a.js/,  
  /polyfill\d6c70b5.js/,  
  /sp_main\./5db9585a.png/  
];  
performance.getEntriesByType("resource")  
.filter(e => targets.some(r => r.test(e.name)))  
.map(e => ({  
  name: e.name,  
  start: e.startTime.toFixed(),  
  dur: e.duration.toFixed(),  
}));
```

+ Ask ChatGPT 🔍 🗣️ 🔊

The image displays a web browser interface with a performance analysis tool. The top section shows a list of network requests, with the first one selected. The bottom section shows the 'Long Task' analysis results, indicating a long task of 100ms. The browser's address bar shows the URL 'https://pm.pstat.net/resources/js/main.7f3a8352.js?o=www'.

Network Requests:

Request	Duration	Start Time	End Time	Transfer Size	Encoded Body Size	Decoded Body Size	Initiator Type	Next Hop Protocol
GET /resources/js/main.7f3a8352.js?o=www	100ms	0.000s	0.100s	100KB	100KB	100KB	Script	HTTP

Long Task Analysis:

Task	Duration	Start Time	End Time
Script	100ms	0.000s	0.100s

Browser Address Bar: `https://pm.pstat.net/resources/js/main.7f3a8352.js?o=www`

The screenshot shows a ChatGPT interface with a dark theme. At the top, there's a header with a plus icon, the text "ChatGPT", and a share icon. Below the header, the main content area displays a conversation. The user's message is in Korean, asking for an analysis of a JavaScript code snippet. The assistant's response is also in Korean, providing a detailed explanation of the code's functionality, including the use of the 'this' keyword and the 'call' method. The code snippet is as follows:

```
const obj = {
  name: 'John',
  age: 30,
  greet: function() {
    console.log(`Hello, my name is ${this.name} and I am ${this.age} years old.`);
  }
};

obj.greet.call({ name: 'Jane', age: 25 });
```

The assistant explains that the code defines an object 'obj' with properties 'name' and 'age', and a method 'greet'. The 'greet' method uses 'this' to refer to the object it is called on. In the final line, 'obj.greet' is called with an object literal as an argument, which is passed to the 'call' method. This results in the 'greet' method being executed with 'this' pointing to the object literal, outputting 'Hello, my name is Jane and I am 25 years old.'

사용 사례: 누가, 어떻게 활용할 것인가?

본 프로그램은 기술 및 비기술 직군 모두에게 최적화된 사용자 인터페이스를 제공하여, 각자의 목표에 맞는 깊이 있는 분석을 가능하게 합니다.



개발자

코드 수준의 성능 최적화가 필요한 부분을 정확히 파악합니다. 특히 **렌더링 병목 현상**과 비효율적인 리소스 로딩 순서를 감지하여 개선 작업을 수행합니다.



디자이너 / PM

UX 관점에서 로딩 타이밍을 분석합니다. 초기 화면에 반드시 필요한 **핵심 UI 요소**가 언제 사용자에게 보이는지 확인하고, 리소스 할당 계획을 수립합니다.



비기술 기획자

복잡한 기술적 지식 없이도 시각적인 자료를 통해 **서비스 품질 저하 요인**을 이해합니다. 개선 방향에 대한 논의 시 명확한 근거를 제시할 수 있습니다.

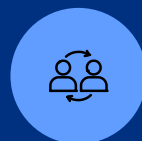
기대 효과: 사용자 만족도와 효율성의 향상

현실적인 로딩 경험 분석은 단순히 속도를 높이는 것을 넘어, 제품의 전반적인 품질과 팀 간의 협업 수준을 향상시킵니다.



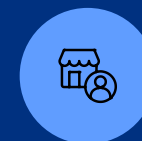
사용자 경험(UX) 개선

느린 로딩으로 인한 사용자 이탈률을 최소화하고, 모든 환경에서 일관되고 만족스러운 로딩 경험을 제공합니다.



팀 간 협업 강화

개발자와 디자이너가 동일한 시각적 데이터를 공유함으로써, 성능 문제에 대한 오해를 줄이고 효율적인 해결책을 공동으로 마련합니다.



운영 효율성 및 성능 튜닝

불필요한 리소스 로드를 제거하고 캐싱 전략을 최적화하여, 서버 비용을 절감하고 서비스의 전반적인 반응 속도를 높입니다.

결론: 기술과 사용자 경험의 연결

우리의 프로그램은 개발자 도구의 기술적 깊이와 UX 디자인의 사용자 중심적 사고를 결합합니다. 목표는 모든 사용자에게 빠르고 즐거운 웹 경험을 제공하는 것입니다.

향후 계획 및 확장 로드맵

모바일 환경 완벽 지원

iOS 및 Android 네이티브 앱 환경의 로딩 분석 기능을 추가하여 테스트 범위를 확장합니다. (Q4 2024)

비 개발자 직군 기대효과

비 개발자를 위한 UI 로드과정을 넣어서 로딩을 적절히 배분하는 작업 가능

자동화된 테스트 통합

CI/CD 파이프라인에 성능 테스트를 자동으로 통합하여 배포 전 성능 회귀를 방지합니다.