

포트폴리오 작성을 위한
리액트 프로젝트 고도화
4회차

Recap

1. 성능 측정 툴 소개 및 사용법
 - a. Lighthouse
 - b. Performance
 - c. Profiler
2. 최적화
 - a. Code Splitting
 - b. Lazy Loading

Code Splitting

1. 말 그대로 코드 분할

- a. 전체 코드를 작은 조각들로 나누는 것
- b. 처음에 불러올 코드가 줄어드니 사용자가 빈 화면을 보는 시간이 줄어든다

2. 특징 **다이나믹 로딩**

- a. 지연 로딩(Lazy Loading)
 - i. 애플리케이션의 특정 부분이 사용자에게 의해 필요할 때 **load**하는 것
 - ii. 어떤 페이지나 컴포넌트가 사용자의 특정 액션을 통해 나타난다면, 그 액션이 발생하기 전까지는 그 페이지나 컴포넌트를 로드하지 않음
- b. 캐싱 최적화
 - i. 코드를 작은 단위로 나누면, 변경이 필요한 부분만 업데이트 가능.
 - ii. 전체 애플리케이션을 새로고침하지 않고도, 특정 부분만 쉽게 업데이트할 수 있음

Code Splitting

1. index.js 사이즈 감소

dist/index.css	13.63 kB	gzip: 4.07 kB
dist/views/vendor-f4ea1fe1.js	191.61 kB	gzip: 61.71 kB
dist/index.js	592.83 kB	gzip: 186.32 kB

(!) Some chunks are larger than 500 kBs after minification. Consider using dynamic import() to code-split the application.

Use build.rollupOptions.output.manualChunks to improve chunking.

dist/views/Feed.view-28559a3d.js	122.58 kB	gzip: 39.95 kB
dist/index.js	150.89 kB	gzip: 46.09 kB
dist/views/vendor-b27f5108.js	191.61 kB	gzip: 61.71 kB
✓ built in 4.13s		

Code Splitting

1. FCP, LCP 비교

■ First Contentful Paint 2.5 s	■ Largest Contentful Paint 3.5 s
● Total Blocking Time 0 ms	● Cumulative Layout Shift 0
● Speed Index 2.5 s	

■ First Contentful Paint 2.0 s	■ Largest Contentful Paint 3.2 s
● Total Blocking Time 0 ms	● Cumulative Layout Shift 0
● Speed Index 2.0 s	

Agenda

1. 포트폴리오 관리 시 주의사항
2. 각종 최적화 기법

포트폴리오 관리 시 주의사항

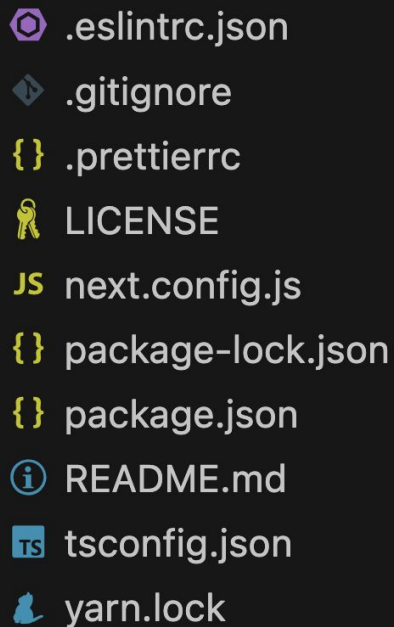
1. Default branch에서 빌드확인 dependency error











a. npm install 이 안되는 경우

```
npm ERR! Found: react@17.0.2
npm ERR! node_modules/react
npm ERR!   peer react@>=16.0.0 from @ant-design/cssinjs@1.18.2
npm ERR!   node_modules/@ant-design/cssinjs
npm ERR!     @ant-design/cssinjs@"^1.18.2" from antd@5.12.6
npm ERR!     node_modules/antd
npm ERR!       antd@"^5.12.4" from the root project
npm ERR!   peer react@>=16.0.0 from @ant-design/icons@5.2.6
npm ERR!   node_modules/@ant-design/icons
npm ERR!     @ant-design/icons@"^5.2.6" from antd@5.12.6
npm ERR!     node_modules/antd
npm ERR!       antd@"^5.12.4" from the root project
npm ERR! 51 more (@ant-design/react-slick, @rc-component/color-picker, ...)
npm ERR!
npm ERR! Could not resolve dependency:
npm ERR! @firebase/auth@"*" from the root project
npm ERR!
npm ERR! Conflicting peer dependency: react@18.2.0
npm ERR! node_modules/react
npm ERR!   peer react@"18.2.0" from react-native@0.73.1
npm ERR!   node_modules/react-native
npm ERR!     peer react-native@"^0.0.0-0 || >=0.60 <1.0" from @react-native-async-storage/async-storage@1.21.0
npm ERR!     node_modules/@react-native-async-storage/async-storage
npm ERR!       peerOptional @react-native-async-storage/async-storage@"^1.18.1" from @firebase/auth@1.5.1
npm ERR!       node_modules/@firebase/auth
npm ERR!         @firebase/auth@"*" from the root project
npm ERR!
npm ERR! Fix the upstream dependency conflict, or retry
npm ERR! this command with --force or --legacy-peer-deps
npm ERR! to accept an incorrect (and potentially broken) dependency resolution.
```

포트폴리오 관리 시 주의사항

1. lock 파일은 하나만 모노레포를 쓸때는 npm 안쓰는게 좋음
=> yarn, pnpm
- a. 디펜던시 파악이 어려움
 - b. 패키지는 하나만 사용하세요!
 - i. 팀프로젝트라면 맞출것
 - ii. 솔직히 말하면 패키지 매니저 별 차이 없음
 - iii. 체감할만큼의 프로젝트를 개인적으로 하기 어려울 것



-  .eslintrc.json
-  .gitignore
-  .prettierrc
-  LICENSE
-  next.config.js
-  package-lock.json
-  package.json
-  README.md
-  tsconfig.json
-  yarn.lock

최적화 주의사항

1. 꼭 필요한 것인지 확인 사용자의 사용빈도를 따져봐야한다 !
2. Lighthouse 점수는 올라갔는데, 화면은 오히려 느릴수도 있음

이미지 최적화

1. 용량이 너무 크면 오래걸림

- a. 로딩도 오래걸리고 preload 별 도움안됨
- b. 브라우저에 과부하 걸림
 - i. 체감되는 정도는 아닌데, 최적화 관점에서 볼때는 안 좋음
 - ii. 만약에 화면에 이미지가 10000개 보이는데 모두 오버사이즈라면?

2. 권장하는 크기는 2x 2x 해야 안깨짐

- a. 윈도우는 굳이 2x 필요 없음
- b. 맥에서 retina display에서 이미지가 깔끔하게 보이려면 2x를 넣어줘야함

3. 크기를 줄이는 방법

- a. 디자이너가 있다면 이미지를 새로 받아라
- b. 서버에서 스토리지에 이미지를 저장할 때 화면 보고 크기에 맞는 사이즈로 저장해야함
- c. `.webp`

이미지 최적화 제대로 하려면

1. 스토리지에 화면에서 필요한 사이즈별로 이미지를 들고있어야함
 - a. 같은 이미지가 여러 화면에 사용되면 해당 영역에 맞는 크기의 **presigned-url**을 넘겨줘야함

Calit!



Calendar

Kanban

Todo

< 2024. 01 >



SUN

MON

TUE

WED

THU

FRI

SAT

31

1

2

3

4

5

6

이미지 최적화 - 파일포맷

1. 상황에 따라 다른 포맷을 사용해야함.

- a. **svg** - 아이콘. 원가 작은 이미지
 - i. 개인적으로는 **64*64**보다 작으면 **svg**
 - ii. 임의의 기준일 뿐 꼭 따를 필요는 없음
- b. **png** - 스크린샷 처럼 세부사항이 잘 보여야 하는 경우
 - i. **jpeg**과 **png**의 차이점도 이해해야함
 - ii. **jpeg**은 용량이 작은 대신 품질이 손상됨
 - 1. 가끔 백엔드에서 비용타령하는 경우가 있는데
 - 2. 스토리지 비용 계산해보라고 하세요
 - 3. 별로 차이 안남
- c. **webp** - 요즘 구글에서 미는 파일포맷 **압축이 굉장히 잘되는 포맷이라고 보면됨**
 - i. 확실히 용량 측면에서 유리함
 - ii. 로드가 빠리됨

폰트 최적화

■ Avoid enormous network payloads — Total size was 3,501 KiB

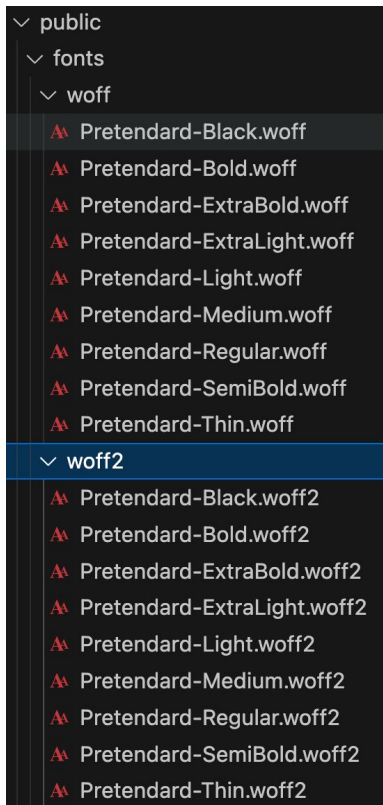
Large network payloads cost users real money and are highly correlated with long load times. [Learn how to reduce payload sizes.](#) `LCP`

URL	Transfer Size
localhost <code>1st Party</code>	3,433.7 KiB
...woff2/Pretendard-Bold.woff2 (localhost)	773.8 KiB
...woff2/Pretendar....woff2 (localhost)	768.0 KiB
...woff2/Pretendard-Medium.woff2 (localhost)	758.2 KiB
...woff2/Pretendard-Regular.woff2 (localhost)	747.2 KiB

폰트 최적화

1. 브라우저 호환성 고려해서 woff, woff2 설정

- a. 둘 다 필요없음
- b. woff2만 설정해도 충분함
 - i. 요즘 웬만하면 다들 크롬이나 엣지 사용
 - ii. 크롬 안쓰는 개발자들 솔직히 문제있음



폰트 최적화

1. 가급적이면 웹폰트 추천

- a. CDN을 직접 관리 안하는것 자체만으로도 일단은 관리포인트가 줄어든다
- b. 웹폰트 호스팅을 도저히 못믿겠으면 직접 하는것도 방법일듯

폰트 최적화

1. 가급적이면 웹폰트 추천

- a. CDN을 직접 관리 안하는것 자체만으로도 일단은 관리포인트가 줄어든다
- b. 웹폰트 호스팅을 도저히 못믿겠으면 직접 하는것도 방법일듯

2. @fontsource 라이브러리 사용추천

- a. <https://www.npmjs.com/package/@fontsource/noto-sans-kr>
 - i. 원하는 폰트를 찾으면 됨
- b. 용량이 훨씬 가볍고
- c. import하면 사용되는 스타일의 폰트만 추가 가능

폰트 최적화

1. 가급적이면 웹폰트 추천

- a. CDN을 직접 관리 안하는것 자체만으로도 일단은 관리포인트가 줄어든다
- b. 웹폰트 호스팅을 도저히 못믿겠으면 직접 하는것도 방법일듯

2. @fontsource 라이브러리 사용추천

- a. <https://www.npmjs.com/package/@fontsource/noto-sans-kr>
 - i. 원하는 폰트를 찾으면 됨
- b. 용량이 훨씬 가볍고
- c. import하면 사용되는 스타일의 폰트만 추가 가능

3. Next.js에서 추천하는 폰트 최적화 방식이 있긴함

- a. 성능상 우위에 있는지 Lighthouse에서 측정은 안됨
- b. 그래도 한번 시도해보는 편이 좋을듯

추상화

1. 복잡한 내용을 단순화하는 과정입니다.
2. 복잡한 코드의 세부 사항을 걱정하지 않고도 프로그램의 핵심적인 부분에 집중할 수 있게 해줍니다.
3. 추상화의 목적은 코드의 복잡성을 관리하고 재사용성을 높이는 것입니다.
 - a. 객체지향에서 나온거기 때문에 원칙상 재사용성도 고려
 - b. 프론트에서 컴포넌트를 쪼갤때는 코드 복잡성 관리 + 가독성 향상

추상화

1. 컴포넌트를 잘 쪼개줘야함

- a. 변수명만 보고도 그 컴포넌트의 역할을 알 수 있도록
- b. 컴포넌트 하나의 **return**구간이 너무 길어지면
 - i. 로직도 많아지고
 - ii. **jsx**도 많아져서
 - iii. 가독성이 떨어짐

2. 추가작업

- a. 불필요한 요청 제거
- b. 함수형 프로그래밍 원칙
- c. 불필요한 **state**제거