

'use Server': 서버액션생성 필요지식을
 비동기함수여야

NextJS 학습일

① form action → ② BackGround → ③ Trigger
 양식제작 스스로 요청 send function

⑤ Property key name과 데이터 보유 변수 이용

동일시 생략 title: title ⇒ title

요 사용자에게 피드백 제공

양식 → 피드백 ↗ ⚡ Redirection
 제작, 제공 ↗ .storePost: DB 적재 함수 await
 (리다이렉트) .redirect('/feed')

(제작 버튼이 로딩... ↗ ⚡ useFormStatus from react-dom
 으로 뷰까지...) - const status = useFormStatus()
 양식제작상태에 대한 개념화 되었
 ↗ 해당 향후 사용하는 카카오네트워크
 양식 내부에서 사용
 if (status.pending)

useFormState from 'react'

유용하지 않은 데이터 입력 시 퍼드백 제공

△ input tag에 required props 추가

↳ 개발자도 구하고 해당 props 사용하면 어쩔래이

△ 서버에서 제출된 데이터 유효성 검사

↳ 데이터 적재전 검사로직 추가

cf) error message 제공 방법

```
let errors = []
```

```
if (!isValidContent(content)) {
```

```
    errors.push('This content is required!')
```

```
}
```

```
if (errors.length) {
```

return errors } → 서버액션에서 프로퍼티로 반환 가능

```
}
```

= 여러 개체 반환해 콜백이언트 처리 OK

↳

useFormState

사용자 양식 제출 시 상태기반

UI 업데이트

form Action에서

반환하는 모든 데이터 출력

useActionState ≡ use Form State

- use Form Status처럼 중첩 컴포넌트에서 호출하지 않아도 됨
→ 컴포넌트 내에서 직접 호출해도 양의 출역

(cf) 인자로 함수를 넘길 때 '포인터'를 전달한다고 함 00°

use Action State (cbfn, {})

Initial State

formAction

formAction 설정 X 사용할 초기값

const [state, formAction]

Current Form State

Updated form Action

react가 받은 formAction

∴ formAction이 받았던 상태

<form action={formAction} > {...} </form >

Restructuring

상황: action 함수에 'use Server'
use Action State 흑 사용으로 'use client') 충돌!

- 새 컴포넌트 생성 'use client' 블록
서비스액션 props로 받아오기

Callback) function createPost (prevstate, formData) {
 'use Server'
 const title = formData.get ('title')
 :
 }

서비스액션의 첫번째 인자로 받지 X

→ use Form State 흑으로 전달되기 때문

<post form action = {createPost} />

Const [state, form Action]

= use Form State (action, & 3)

→ 양식에 설정할 수 있는 새로운
form Action return !

<form action = {formAction} />

react는 양식 제작을 위한 해 양식 상태를 업데이트 +
use Form State에 전달하는 서비스액션 호출 방법 변경

* Next.js는 이미지 바로 저장X

외부 라이브러리 사용 (Cloudinary, AWS ..)

좋아요 기능 구현하기

① 좋아요 table 구현

② 좋아요 버튼 클릭시 작동 함수

 New Server Action

수신 URL (URL)

새 게시글 / 사용자 ID 생성

전송 URL (작업)

삭제하기

③ 좋아요 버튼의 action에 전달

I. props drilling 해서 상위 컴포넌트에서 함수 import 후
action을 전달 → `<Button action={togglePost} />`

II. 해당 좋아요 버튼 컴포넌트를 `<form action={togglePost}>`
`<Button /> </form>` 감싸다.

(JS)

`bind(null,`

param 1

함수 내부의 키워드가 참조할 대상 정의

) param 2

최종적으로 함수 실행시 새로운 첫번째 인수

함수 객체 호출해 그 함수 미리 구성 가능!

나중에 함수 실행시 전달될 값 설정

개성문제 방지 위한 데이터의 재검증

```
import { revalidatePath } from 'next/cache'
```

Next.js가 페이지 데이터를 재식하여 데이터의 변경사항을 감지X

⇒ revalidatePath() 를 데이터 변경시 호출

- Next.js에 일부페이지의 데이터가 변경되었는지를 알려야 어떤페이지의 데이터가 변경되었는지도!

revalidate('/routes',)

Optimistic Updates in NextJS.

(79)

import {useOptimistic} from 'react'

• 사용법

useOptimistic ()

param1. Initial Data

param2.

리액트가 자동으로 호출할 함수

▷ Returns

const [optimistic, optimistic] = useOptimistic (data, update, triggered)
|
| updated : data
| update : function
| triggered : function
(prevstate, (update 함수시 필요한 data))
{ }
업데이트 실행
함수

△ 서버 측에서 변경 처리하기 전에 클라이언트 측에서

Param 1 을 변경
(data)

△ data 즉시 변경되고 서버 측 업데이트 완료된 후
서버 측 상태와 다시 동기화

aggressive Caching

→ 중복요청 방지 & 단일요청 동안만
Stores data requests with the same configuration

1. Request Memoization

2. Data Cache

데이터 불경 X 요청 자체 X 추가 요청 방지
Stores & reuses fetched data until its
revalidated → 사용자가 수정해놓은 내용 or
설정한 특정 시간이 지나면

3. Full Route Cache

→ 기존 페이지 재사용, 업데이트 데이터로 re-render

Server Side

전부 route cache가 있어도
NEXTJS 서버에 요청을 보내야

4. Router Cache

Stores the RSC Payload in memory
In the browser

Client Side

Ex) 페이지 이동을 해도 새로그가 쌓이지 않는 이유 → data cache

① BE에서 fetch 함수로 데이터 가져옴

② 응답 data 서버 측 캐시에 저장 후 re-use

185

(사용자가 재사용 말라고 지시할 때까지)

- 1. revalidatePath()
- 2. Cache Configuration

쿠리어링 재검증하는 방법

- revalidate Path ('routes')
- Cache Configuration
- request configuration

Cache Configuration Object

```
const response = await fetch('http://localhost:8080', {
  cache: 'force-cache' // default
```

3) ↳ nextJS의 built-in 헬퍼 override
(캐시 설정 찾아 봄 등..)

① force-cache : 가능한 캐시되거나 사용!

② no-store : 캐시되지X (동일 요청이 보내지는
이후의 데이터)

↳ 세요성을 항상 전송

Request Configuration in Next setting

```
const response = await fetch('http://localhost:8080', {
```

next: { revalidate: 5 } })

↳ NextJS가 캐시데이터를
재사용해야 할 흐름

(tonstack or staleTime 같은..)
query

파일 전체에 대한 캐싱 제어 설정

- export const revalidate : number = 5 → 해당 숫자의 초 동안 데이터 캐시 & 재사용
↳ Next.js 가 해당 함수 인지 유함
- export const dynamic : "auto" | "force-dynamic" | "force-static"
 - default
 - cache: no-store
 - force-dynamic : 캐싱방지 (항상 새로운 값)
 - force-static : 캐싱강제
 - ↳ 새 데이터 전역
갖고오지 않을 수도 있음

→ force-dynamic 설정한 것과 같은 효과 but 천장!

Import {unstable_noStore} from 'next/cache'

: 데이터가 재사용지 않도록 확실하고자 하는 경우 네트워크 출력

ex) function Component() {
 unstable_noStore()
}

full-route Cache

- : 빌드시 자동 렌더링 & 캐싱 But 동적 routes 가능 X
데이터가 변경될 때 강제 유통하고 업데이트 & 렌더링 X

revalidatePath

- : 재시도 데이터 제거! (변경되지도 않은 데이터는 뒤늦게 refetching 보다 명시적으로 refetching 하는게..)

↳ 중첩 경로 / 중첩된 페이지는 데이터 & 라우트 캐시가 삭제 / 재검증 X

별도로 명시 필요 ('/mess', **'layout'**) → 사이트의 모든 페이지, 중첩 page 경로 캐시 삭제 O

< 'page': default - 특정 페이지 재시도만 재검증 & 삭제
→ fnstack query의 query key 같을..

revalidate Tag from 'next/cache'

- : 데이터 가져올 때 재시도 요청을 보낼 때 태그 할당 O

String
array

① fetch('http://localhost:8080', { next: { tags: ['msg'] } })

② revalidate Tag (**'msg'**) → 해당 태그가 있는 모든 재시도

데이터 재검증 & 폐기 (다른 사이트, 동일 tag ⇒ 전세계 캐시 삭제)

• 개발서버 사용시 full-route cache 업음!

중복 요청 제거

상황: 페이지로드시 db 접근을 두 번씩 하는 현상

원인: 같은 메시지를 페이지 내부, 레이아웃에서 모두 가져옴

`import { Cache } from 'react'` : 중복 요청제거 방지해야하는
함수 강제는데 사용

ex)

```
const getMessage = Cache(function getMessage() {
```

...

})

호출은 한번만 실행, 호출에 대한 응답이 저장돼 다음 호출에

재사용됨

불안정데이터 재사용 방지방법

`import { unstableCache } from 'next/cache'`

ex)

```
const getMessage = unstableCache (
```

`cache (function getMessage()`

불안정데이터

제거

))

↳ 중복 요청제거

↳ Return Promise ^{함수 토출시} _{async-await}

```
# import {unstable_cache} from 'next/cache'
```

사용법

const 함수명 = unstable_cache (function 함수명 () {}) ,

[cache-key], { 구성옵션 }

→ 내부적으로 캐시된 데이터 식별에 사용

▶ 구성옵션

191

revalidate: number - 재검증 시간(초) 설정

tags: string[] - 해당 태그가 캐시된 데이터에 추가
 → revalidate Tag 호출 시 해당 데이터
 삭제

새로운 데이터 생성시 나타나지 않음 → aggressive cache

- 캐시된 데이터가 변경되었음을 고지

① revalidate Path

:revalidate Path('routes') 여기서 사용된 데이터 삭제
 새 데이터 불러옴

② revalidate Tag ('msg')

Image Optimization

- import {Image} from 'next/image' : size optimization

1) 노출파일시스템에서 이미지 불러오기

```
import logo from '@/assets/logo.png'
```

```
<Image src={logo} alt="" />
```

→ **가로/세로** 전체를 설정해야됨 (너비, 높이 설정 X)

경로 X 이미지에 대한 전체정보

(이미지 사이즈, 배포 후/개발시 작동하는 경로 ...)

height, width src, blurDataURL

⇒ 표준요소 렌더링하지만 **srcset, loading=lazy**
 (다양한 일도에 맞는 (화면에 실제 보이는
 크기의 이미지 표시) 경우에만 이미지로드)

노출이미지 사용시 사이즈 조절

주의) 사용하는 입력파일의 너비&높이 자동결정해서 입력파일이
 불필요하게 커지지 않도록 함

해결) width & height 속성을 줘서 수동으로 overwide **전장X**

sizes property 사용 **전장O** ex) "10vw" - next.js에
 알아서 재조정

Image 정포넌트 → `src = "loading"`

: modern browser에서 이미지를 드시 화면에 보이는 경우만 표시함 → header component에 적합 x ⇒ **Priority**

Priority : 사진로드 (=fetchPriority)

⊕ 외부사이트에서 이미지를 불러오는 경우 보안상 차단

// next.config

const nextConfig =

image: { remotePatterns: [{ hostname: 호스트주소 }] }

2) 사용자가 생성한 이미지 최적화

= 너비&높이를 모르는 이미지 → **fill**



이미지 요소 `position: absolute`로 설정함 + 부모요소의 너비, 높이까지
→ 부모요소 `position: relative`와 높이&너비 설정 필수

이미지 사이즈 최적화

- 이미지 호스팅 BE에 크기 조정한 이미지 요청

- ① loader 프로퍼티로 요청 : function returning a URL string
- ② fill & sizes 프로퍼티 추가 → view-port에 맞춰 딱 맞는 이미지
- ③ next.config 에 도메인 맵핑

Metadat2

① Static metadata

`export const metadata = { }`
must be export!

`title : 페이지 제목`

`description : <meta name="description"/>`

+ OpenGraph ...

② Dynamic metadata : depend on dynamic contents

`export async function generateMetadata() {`
↳ ex) 경색 매개변수, 가능성이...

`return {`

`title:, description`

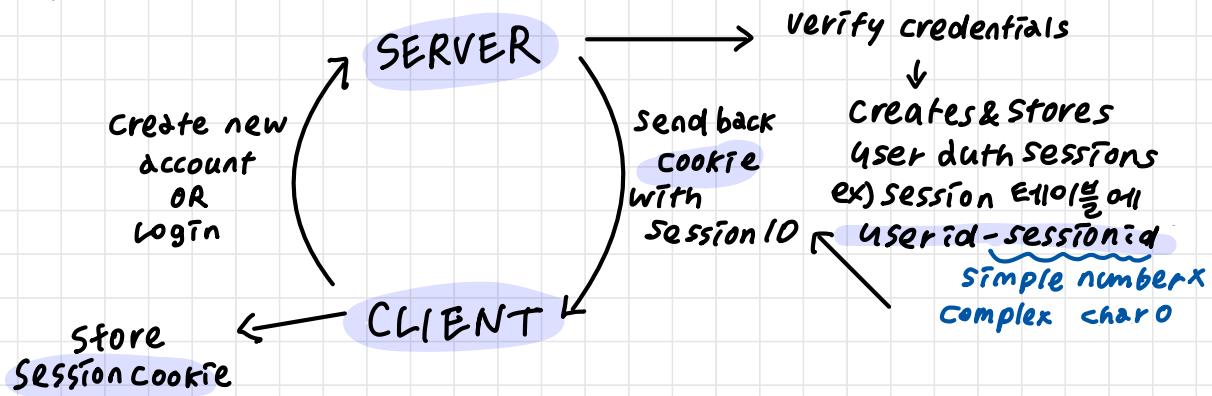
} } 33

How Does Authentication Work?

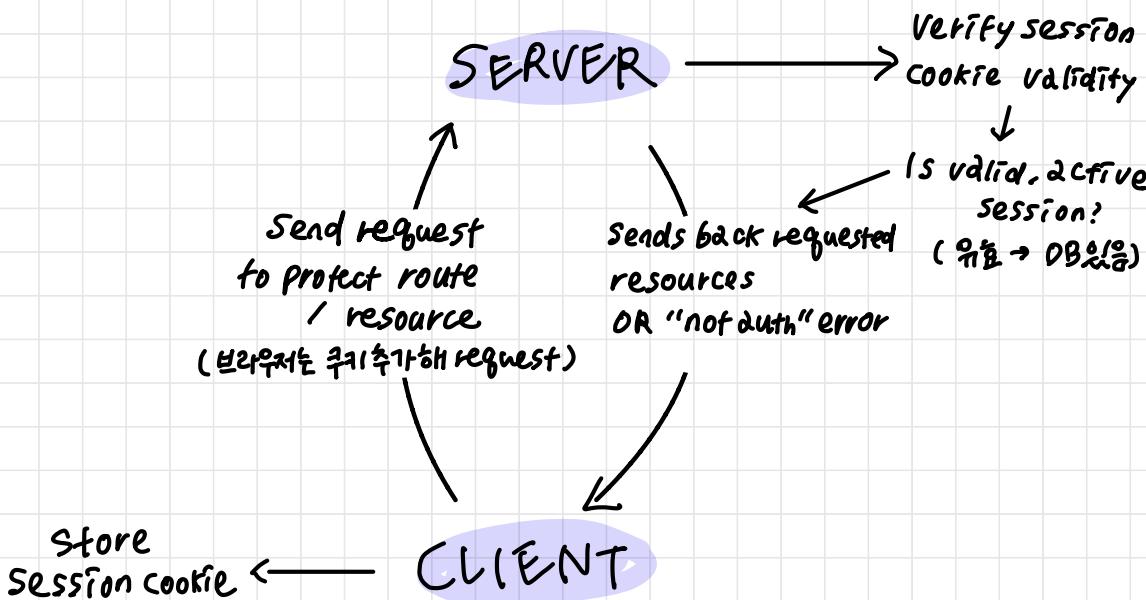
LOGIN: Verify user credentials & "mark" user as authentication
 ↴
 Email & Password

Authorized Access: Grant authenticated user access to protected routes & resources

I. LOGIN



II. Access Protected Resources



쿠키 확인

- 개발자 모드 → Application → Cookies
- 브라우저는 자동으로 쿠키를 요청 Headers에 첨부
(개발자 모드 → Network → Headers → Request Headers → $\xrightarrow{\text{Cookie}}$)