

# Netflix Clone Work Guide



## Preview

1. SSR 적용하여 로딩시간 축소를 위한 Next.js 사용
2. typescript를 이용한 타입 지정
3. tailwind css를 이용한 스타일 선언
4. tmdb api를 이용한 data fetching
5. firebase로 로그인 기능 구현
6. recoil을 사용하여 양방향 데이터 관리 및 상태 관리

# Directory

atoms	Recoil에서의 상태 단위 포함
components	App에서 사용할 component
constants	App에서 공통으로 활용해야 되는 정보값
hooks	custom hook 포함
pages	페이지를 담당하는 component(폴더구조로 url 결정)
public	App에서 사용되는 정적 파일들
styles	스타일 관련 파일 모음
utils	데이터(API) 관리 폴더

- > atoms
- > components
- > constants
- > hooks
- > node\_modules
- > pages
- > public
- > styles
- > utils
- .env.local
- ! .gitignore
- TS firebase.ts
- TS next-env.d.ts
- N next.config.js
- npm package-lock.json
- npm package.json
- ⚙️ postcss.config.js
- M README.md
- ⚡ tailwind.config.js
- TS tsconfig.json
- TS typings.d.ts

☰ .env.local	모든 환경에서 최우선순위로 적용할 환경변수를 정의
☰ firebase.ts	firebase config,sdk 정보 저장
☰ next.env.d.ts	타입스크립트 컴파일러가 타입을 가져옴
☰ next.config.js	next.js의 환경설정, 커스텀 설정 관리
☰ tailwind.config.js	tailwind css 설정 파일
☰ tsconfig.json	typescript 설정 파일
☰ typings.d.ts	type 정리 파일
☰ package.json	프로젝트 설정 파일
☰ .gitignore	git에 업로드 되면 안되는 파일 정리

# Next.js

## pre- rendering

### pre-rendering이란?

JavaScript로 모든 작업을 수행하는 대신 미리 각 페이지에 대해서 HTML을 만들어두는 것, SEO에서 더 나은 퍼포먼스를 보여줄 수 있다

pre rendering이 되지 않는다면 data가 불러질 때 까지 페이지를 볼 수 없다 때문에 data가 UI의 전부인 넷플릭스 사이트에서는 적격.

## Why Next.js ?

## Server side Rendering

### Server Side Rendering(SSR)이란?

pre-rendering의 모드로 서버에서 해당 페이지 파일을 불러 올때 pre-render된다. 재이동 측면에서 느리나, 항상 최신 데이터를 유지한다는 특징이 있다.

항상 최신상태를 유지해야하거나 요청마다 다른 내용을 보여줘야하는 넷플릭스 사이트에는 최신 데이터를 유지할 수 있는 ssr 모드가 적격.

## Routing system

### Routing?

Next.js는 페이지 기반 라우팅 시스템이 제공됨. 프로젝트의 가장 바깥 폴더인 /pages 폴더에서 컴포넌트를 export하면 폴더명이 페이지 route가 된다. dynamic router로 중첩 라우팅도 [](대괄호) 하나로 가능하다

dynamic router로 중첩라우팅이 가능하기 때문에 추후 서브페이지를 만들 시 row마다 api에 따라 연결되는 페이지가 많기 때문에 next.js를 선택

# TypeScript

## Why TypeScript?

TypeScript는 기본적으로 변수에 type을 지정해주는데 이는 코드를 추적하는데 도움을 준다  
자료형이 지정되어 있지 않아 유동적인 javascript에서 타입을 고정시켜 속성 변경을 막고 버그 및 에러를 유추할 수 있기 때문에 사용한다.

## Apply to Project

변수, 함수, 클래스에 사용할 수 있도록 data object에 interface를 지정하여 사용

```
export interface Movie {  
    adult: boolean;  
    backdrop_path: string;  
    genre_ids: number[];  
    release_date: string;  
    id: number;  
    title: string;  
    name: string;  
    original_name: string;  
    ...생략  
}
```

**typings.d.ts**

개발과정에서만 실행되고 빌드시  
에 일반 자바스크립트 파일로 컴파  
일 되지 않는 d.ts 파일을 만들어  
데이터 속 오브젝트에 interface로  
타입 export

```
{  
    "compilerOptions": {  
        ... 기본 설정 생략  
    },  
    "baseUrl": "",  
    "paths": {  
        "@/*": [".//*"]  
    },  
    ... 기본 설정 생략  
}
```

**tsConfig.json**

기본 설정 + 커스텀 설정으로 baseUrl이라는 속성에 기본 경로  
를 설정해 주고 그 바로 아래에 paths속성에 절대 경로 지정

```
interface IndexProps {  
    original: TV[];  
    topRated: Movie[];  
    sf: Movie[];  
    drama: Movie[];  
    fantasy: Movie[];  
    thriller: Movie[];  
    animation: Movie[];  
}
```

**index.ts**

IndexProps라는 interface 지정 후  
typings.d.ts에 담아둔 interface[] 지정  
한 후 페이지 return값의 타입, list의 타  
입을 IndexProps로 지정

```
const Home:  
NextPage<IndexProps> = ({  
    original, topRated, sf,  
    drama, fantasy, thriller,  
    animation }: IndexProps) =>  
{}
```

# Tailwind css

## Why Tailwind ?

class명을 더이상 생각하지 않아도 되며 다양한 variant를 사용가능함.

일관성 있는 디자인 시스템을 적용하는데 유용하여 같은 디자인의 컴포넌트가 반복되는 netflix page layout에 적용

### tailwind.config.js

```
/** @type {import('tailwindcss').Config} */
module.exports = {
  content: ['./pages/**/*.{js,ts,jsx,tsx}', './components/**/*.{js,ts,jsx,tsx}', './app/**/*.{js,ts,jsx,tsx}'],
  //content: 프로젝트의 콘텐츠 소스를 구성
  theme: {
    // theme: 프로젝트의 기본 테마를 사용자 지정
    extend : {
      ...생략
    },
    plugins: [require('tailwind-scrollbar')({ nocompatible: true }), require('tailwind-scrollbar-hide')],
    //plugins: 재사용 가능한 타사 플러그인, Tailwind 확장
    variants: {
      // variants : 핵심 플러그인에 대해 활성화해야 하는 변수 제어
      scrollbar: ['rounded'],
    },
  };
}
```

## Apply to Project

### 적용예시 Row.tsx

```
<article className='h-40 space-y-0.5 mb-[40px] md:space-y-2 relative z-[4]>
  <h2 className='w-50 text-base font-semibold'>
    {text}
  </h2>
  ...생략
</article>
```

### global.css

```
@tailwind base;
@tailwind components;
@tailwind utilities;
// @tailwind : pre-existing utility class 추가

@layer base {
  // @layer : 어떤 "유ти리티 레벨"에 커스텀 CSS를 담을지 지정
  body {
    @apply bg-[#141414] w-full text-white ...etc ;
    // @apply : 기존 유ти리티 클래스를 사용자 커스텀 클래스에 추가
  }
}
```

# Data fetching

1. tmdb open api를 사용하여 movie를 category 별로 정리하여 가져옴
2. recoil로 데이터 상태관리

## utils > request.ts

데이터 url 호출

```
const API_KEY = process.env.NEXT_PUBLIC_API_KEY;
const BASE_URL = 'https://api.themoviedb.org/3';

const requests = {
  top: `${BASE_URL}/movie/top_rated?api_key=${API_KEY}&language=en-US`,
  sf: `${BASE_URL}/discover/movie?api_key=${API_KEY}&language=en-US&with_genres=878`,
  drama: `${BASE_URL}/discover/movie?api_key=${API_KEY}&language=en-US&with_genres=18`,
  ...
};

export default requests;
```

## .env.local

이 때 API key는 본인 인증키 이기 때문에 남들에게 노출되지 않도록 env.local 파일에 담아 환경변수로 사용

```
NEXT_PUBLIC_API_KEY = {api key}
```

## atoms > globalAtom.js

데이터를 recoil로 상태관리해주기 위해 recoil의 상태 단위 atom을 설정

```
import { atom } from 'recoil';
import { Movie, TV } from '@typings';

export const modalState = atom({
  key: 'modalState',
  default: false,
});

export const movieState = atom<TV | Movie | null>({
  key: 'movieState',
  default: null,
});
```

## index.tsx

데이터가 호출될 index.tsx 파일에서 SSR 방식을 사용하기 위하여 getServerSideProps로 영화정보를 props로 전달함

```
export const getServerSideProps = async () => {
  const [top, sf, drama, fantasy, thriller, animation] = await Promise.all([
    fetch(requests.top).then((res) => res.json()),
    fetch(requests.sf).then((res) => res.json()),
    fetch(requests.drama).then((res) => res.json()),
    fetch(requests.fantasy).then((res) => res.json()),
    fetch(requests.thriller).then((res) => res.json()),
    fetch(requests.animation).then((res) => res.json()),
  ]);

  return {
    props: {
      topRated: top.results,
      sf: sf.results,
      drama: drama.results,
      fantasy: fantasy.results,
      thriller: thriller.results,
      animation: animation.results,
    },
  };
};
```

## \_app.tsx

recoil로 프로젝트의 모든 데이터의 상태를 관리하기 위해 recoilRoot로 component 감싸주기

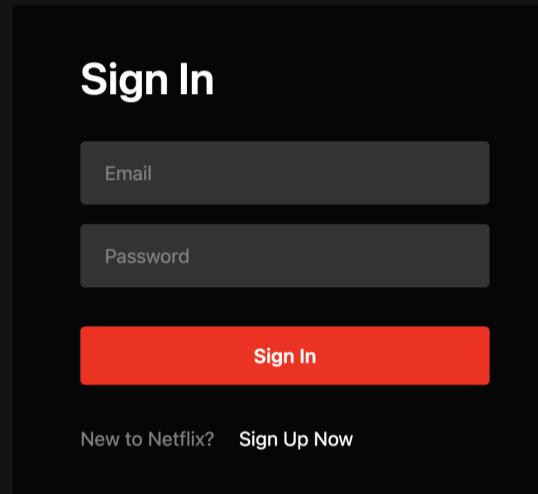
```
<RecoilRoot>
  <Component {...pageProps} />
</RecoilRoot>
```

## Apply

useRecoilState, useRecoilValue 등으로 데이터를 호출, 상태변경

# Login

NETFLIX



## useAuth custom hook 적용

useContext를 이용하여 로그인 정보를 전달해주는 custom hook을 만들어 적용

```
function login() {
  //회원가입, 로그인 값 useAuth 으로 관리
  const { signIn, signUp } = useAuth();
  //전송 이벤트 발생시 handleSubmit함수로 실행할 콜백함수 등록
  const join: SubmitHandler<Inputs> = async ({ email, password }) => {
    if (Login) {
      //만약 클릭한게 로그인 버튼이면 firebase에 로그인처리를 하는 함수 호출
      await signIn(email, password);
    } else {
      //클릭한게 로그인 버튼이 아니면(회원가입 버튼이면) 로그인이 아닌 회원정보 등록함수 호출
      await signUp(email, password);
    }
  };
}
```

## pages > login.tsx

### React-hook-form

React-hook-form을 사용하게 되면 input의 모든 값이 state로 연결되어 하나의 값이 변할때마다 여러개의 자식 컴포넌트들에서 리랜더링되는 것을 방지

```
interface Inputs {
  email: string;
  password: string;
}
//login 시 필요한 input들의 interface로 type 지정
function login() {
  const [Login, setLogin] = useState(false);
  const {
    register, //컨트롤 할 인풋객체에 전개연산자로 등록해서 값을 관리
    handleSubmit, //해당후 전용 전송 이벤트 핸들러
    formState: { errors }, //register로 등록된 값이 올바르지 않으면 에러 반환
  } = useForm<Inputs>(); //useForm 리턴값의 Type은 Inputs에서 가져옴
  ...생략
}
```

## hooks > useAuth.tsx

전역으로 담을 초기 값 정보를 createContext로 생성

```
const AuthContext = createContext<IAuth>({
  user: null,
  signUp: async () => {},
  signIn: async () => {},
  logout: async () => {},
  errors: null,
  loading: false,
});
```

자식 타입을 리액트 노드로 지정

```
interface AuthProviderProps {
  children: React.ReactNode;
}
```

## ! Login.tsx

1. react-hook-form을 이용한 인풋요소 관리
2. useAuth custom hook 적용
3. AuthProvider로 인증값 모든 컴포넌트에서 사용할 수 있도록 활성화
4. 로그인 한 User만 특정 페이지에 접근할 수 있도록 구현

## firebase.ts

firebase 사용을 위한 config, sdk 값, 그 외 필수 설정 요소를 담아줌

```
import { initializeApp, getApp, getApps } from 'firebase/app';
import { getAuth } from 'firebase/auth';

const firebaseConfig = {
  // firebase config, sdk 값
};

const app = !getApps().length ? initializeApp(firebaseConfig) : getApp();
//firebase로 구동한 app이 없으면 아직 인증처리가 되지 않은 상태에서만 초기화
const auth = getAuth();

export default app;
export { auth };
```

## Apply to Input

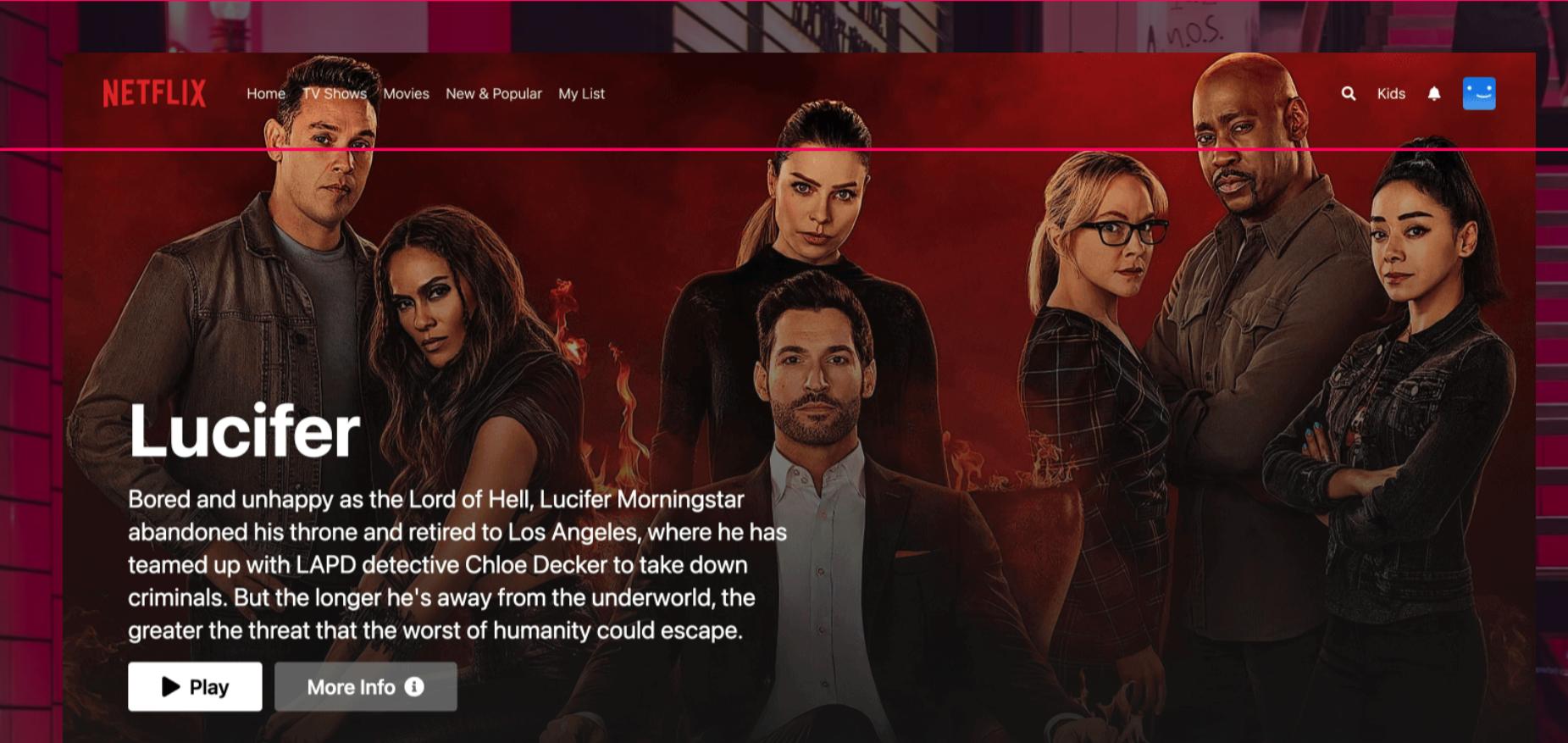
```
<input type='email' placeholder='Email' className='input'
  {...register('email', { required: true })} />
```

1. useEffect 안에서 onAuthStateChanged 메소드를 사용, auth, user 정보를 넣어 firebase가 계속 감시할 수 있도록 함
2. user, loading값이 바뀔때마다 각 함수 및 정보를 객체로 묶어서 내보낼 수 있도록 메모제이션
3. firebase로 인증정보가 아직 받아오지 않은 상태에서는 컴포넌트가 보여지지 않도록 처리
4. useContext hook을 사용하여 내게 필요한 props를 전역으로 사용할 수 있도록 처리

```
export const AuthProvider = ({ children }: AuthProviderProps) => {
  useEffect(() => onAuthStateChanged(auth, (user) => {...생략});
  ...firebase를 통한 로그인, 회원가입, 그아웃 함수 생략
  const memoedValue = useMemo(() => ({ user, loading, error, signUp, signIn, logout }), [user, loading]);
  return <AuthContext.Provider value={memoedValue}>{!initialLoading && children}</
  AuthContext.Provider>;
}
```

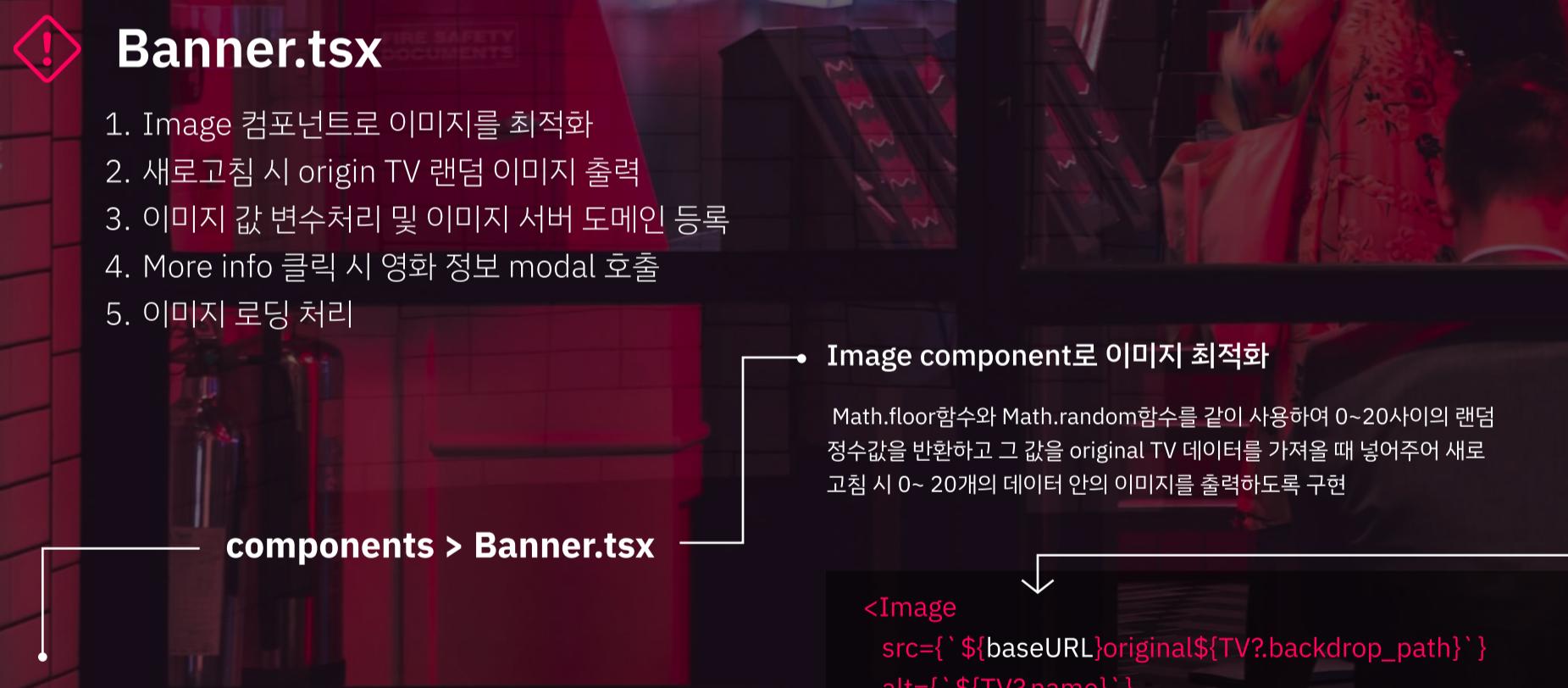
```
export default function useAuth() {
  return useContext(AuthContext);
}
```

# Page guide



## Header.tsx

```
function Header() {
  const [Scrolled, setScrolled] = useState(false);
  // useState로 상태 지정 scroll top = 0일 경우 스타일 변경 X
  useEffect(() => {
    const handleScroll = () => (window.scrollY > 0 ? setScrolled(true) : setScrolled(false));
    // scrollY의 값이 0보다 클경우 setScroll이 True로 변경, 아닐 시 false 처리
    window.addEventListener('scroll', handleScroll);
    // window가 스크롤 될 때 handScroll 함수 실행
    return () => window.removeEventListener('scroll', handleScroll);
    // return으로 이벤트 종료하여 재이벤트 방지
  }, []);
  return(
    <header className={` transition-colors duration-[.5s] ${Scrolled && 'bg-[#141414]!'}`>
      // scroll 이벤트 발생시 스타일 변경
      ...생략
    </header>
  )
}
```



## Banner.tsx

1. Image 컴포넌트로 이미지를 최적화
2. 새로고침 시 origin TV 랜덤 이미지 출력
3. 이미지 값 변수처리 및 이미지 서버 도메인 등록
4. More info 클릭 시 영화 정보 modal 호출
5. 이미지 로딩 처리

### • Image component로 이미지 최적화

Math.floor함수와 Math.random함수를 같이 사용하여 0~20사이의 랜덤 정수값을 반환하고 그 값을 original TV 데이터를 가져올 때 넣어주어 새로 고침 시 0~ 20개의 데이터 안의 이미지를 출력하도록 구현

### components > Banner.tsx

#### Random image 출력

Math.floor함수와 Math.random함수를 같이 사용하여 0~20사이의 랜덤 정수값을 반환하고 그 값을 original TV 데이터를 가져올 때 넣어주어 새로 고침 시 0~ 20개의 데이터 안의 이미지를 출력하도록 구현

```
useEffect(() => {
  const num = Math.floor(Math.random() * 19);
  setTV(original[num]);
}, [original]);
```

```
<Image
  src={`${baseURL}original${TV?.backdrop_path}`}
  alt={`${TV?.name}`}
  priority //이미지 로딩 우선시
  quality={70} // 퀄리티 70
  fill // 부모 element 크기와 맞추기
  sizes='100vw'
  className='object-cover'
  onLoadingComplete={() => ref.current.remove()}
  // 이미지의 로딩이 끝났을 때 로딩 애니메이션 제거
/>
```

## constants > movie.ts

컴포넌트에서 공통으로 활용해야 되는 정보값을 변수로 등록

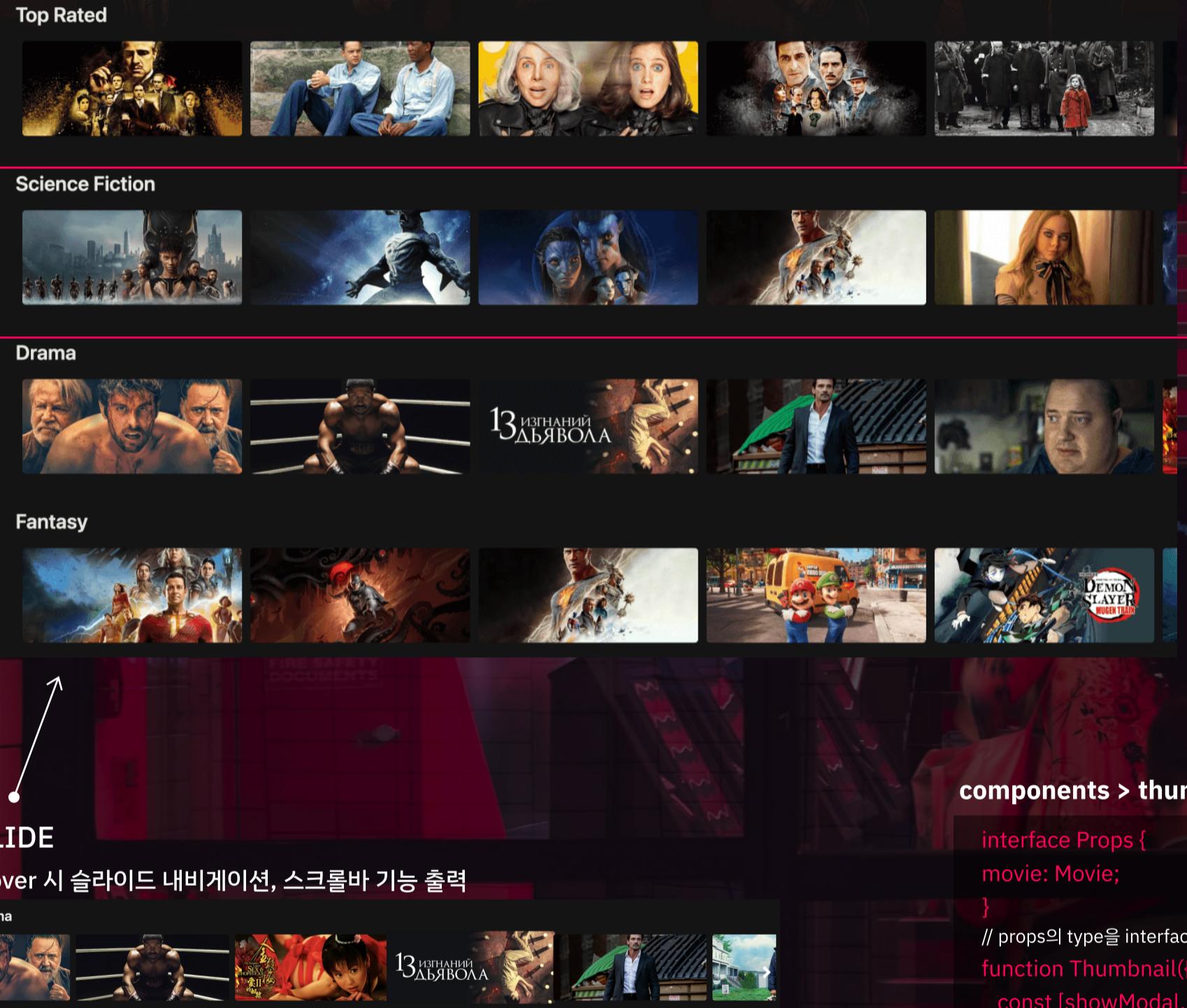
```
export const baseURL = 'https://image.tmdb.org/t/p/';
```

## next.config.js

외부 사이트에서 가져오는 이미지를 최적화하기 위해 해당 이미지의 서버 도메인을 등록

```
module.exports = {
  images: {
    domains: ['image.tmdb.org'],
  },
};
```

# Page guide



```
const handleClick = (direction: string) => {
  if (rowRef.current) {
    const { scrollLeft, clientWidth } = rowRef.current;
    const targetPos = direction === 'left' ? scrollLeft - clientWidth : scrollLeft + clientWidth;
    //targetPos : 좌우버튼 클릭시 클릭한 방향에 따라 가로축으로 이동할 타겟위치값을 구해서 scrollTo로 이동
    rowRef.current.scrollTo({ left: targetPos, behavior: 'smooth' });
  }
};
```

Apply

```
<FaAngleLeft onClick={() => handleClick('left')} />
<FaAngleRight onClick={() => handleClick('right')} />
```

## Row.tsx

1. thumbnail component로 이미지 출력
2. 슬라이드 내비게이션 적용
3. 좌우 스크롤 슬라이드 적용
4. row마다 다른 영화 리스트 적용
5. 이미지 클릭 시 영화 정보 모달 출력

### components > Row.tsx

props로 title과 movies를 던져줌

```
function Row({ title, movies }: Props) {}
```

props type interface로 적용

```
interface Props {
  title: string;
  movies: Movie[];
}
```

### list thumbnail

각 장르 별 movie를 map으로 반복문 처리 후  
thumbnail components를 만들어 리스트 처리

### components > Row.tsx

```
<div ref={rowRef}>
  {movies.map((movie) => (
    <Thumbnail key={movie.id} movie={movie} />
  ))}
</div>
```

### components > thumbnail.tsx

```
interface Props {
  movie: Movie;
}

// props의 type을 interface로 지정
function Thumbnail({ movie }: Props) {
  const [showModal, setShowModal] = useRecoilState(modalState);
  // useRecoilState로 modal의 상태 값을 가져옴
  const [CurrentMovie, setCurrentMovie] = useRecoilState<any>(movieState);
  // useRecoilState로 무비 카테고리 상태 값을 가져옴
  return (
    <div onClick={() => {setShowModal(true); setCurrentMovie(movie)}}>
      // thumbnail 클릭시 모달 도출
    </div>
    <Image ...생략 />
  )
}
```

### index.tsx

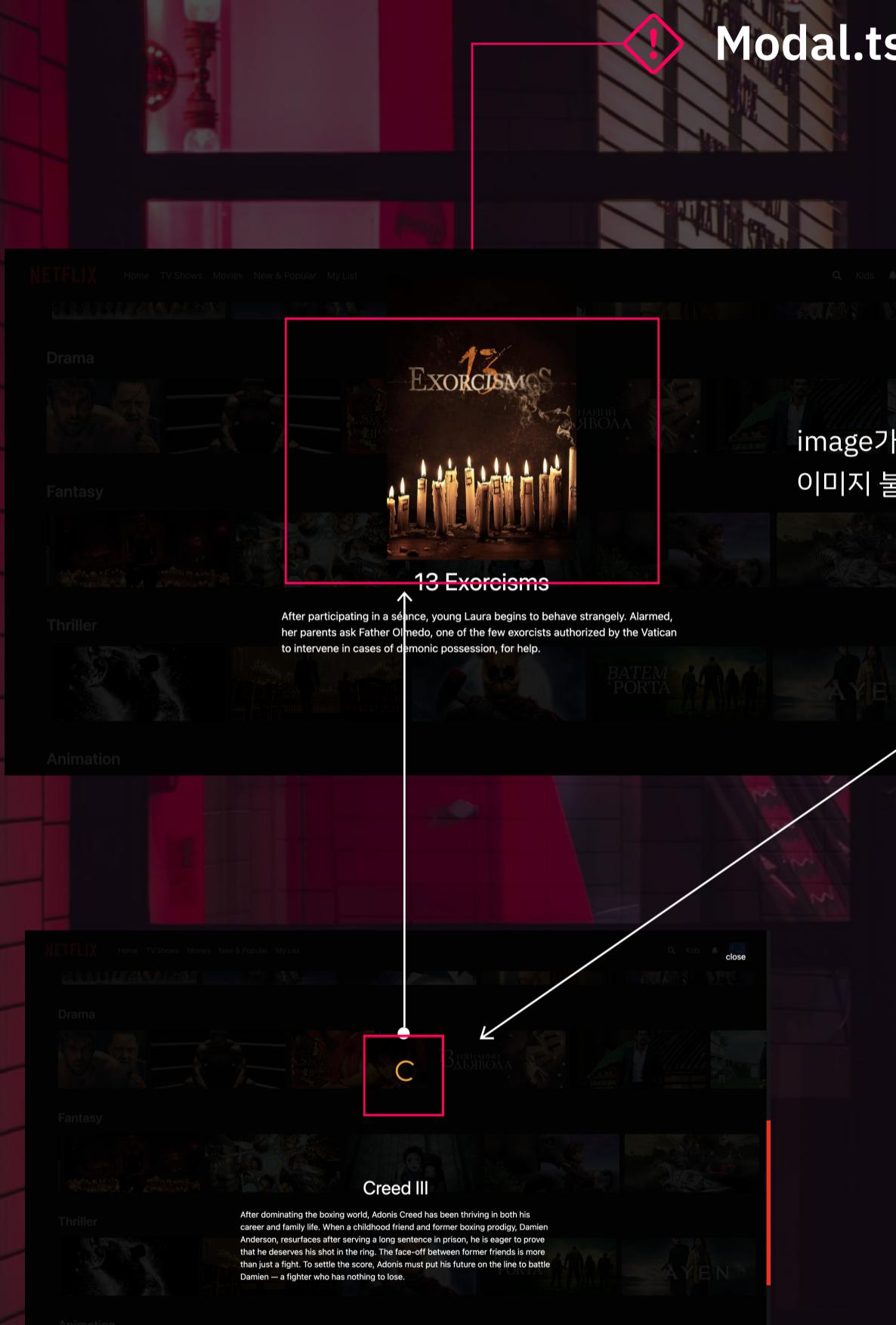
부모 컨포넌트에서 적용

```
<section>
  <Row title='Top Rated' movies={topRated} />
  <Row title='Science Fiction' movies={sf} />
  <Row title='Drama' movies={drama} />
  <Row title='Fantasy' movies={fantasy} />
  <Row title='Thriller' movies={thriller} />
  <Row title='Animation' movies={animation} />
</section>
```

장르별 데이터 type을 정해주기 위한 interface

```
interface IndexProps {
  original: TV[];
  topRated: Movie[];
  sf: Movie[];
  drama: Movie[];
  fantasy: Movie[];
  thriller: Movie[];
  animation: Movie[];
}
```

# Page guide



## Modal.tsx

banner, thumbnail component에서 버튼, 이미지 클릭 시 나오는 공통 모달

1. 로딩 바 구현
2. useRecoilState, useRecoilValue로 상태값 가져옴

image가 불러와질 때까지 loading bar 출력,  
이미지 불러와지면 loading bar 제거

```
<Image  
src={`${baseUrl}w780${Movie?.poster_path}`}  
fill  
className='object-contain'  
alt={`${Movie?.name || Movie?.title}`}  
onLoadingComplete={() => ref.current.remove()}/>
```

loading bar

```
<div ref={ref} className='...생략 animate-ani-rotation'></div>  
  
tailwind.config.js의 theme -> extend 안에 keyframes로 rotation 애니  
메이션을 만들어준 후 ani-rotation라는 이름으로 애니메이션 적용
```

```
keyframes: {  
  rotation: {  
    '0%': { transform: 'rotate(0deg)' },  
    '100%': { transform: 'rotate(360deg)' }  
  },  
},  
animation: {  
  'ani-rotation': 'rotation 1s linear infinite',  
},
```

## useRecoilState, useRecoilValue

Recoil 상태의 값을 반환하는 useRecoilValue를 사용하여 movie의 현재 카테고리가  
어디인지 읽고 useRecoilState로 모달이 열리고 닫히는 상태값 변경 감지

```
const [ShowModal, setShowModal] = useRecoilState(modalState);  
const Movie = useRecoilValue(movieState);
```