Next.js 프로젝트 구성

#01. 프로젝트 생성

```
$ yarn create next-app 프로젝트이름
```

프로젝트 생성시 아래의 항목을 설정한다.

```
√ Would you like to use TypeScript? ... No
√ Would you like to use ESLint? ... Yes
√ Would you like to use Tailwind CSS? ... No
√ Would you like to use `src/` directory? ... Yes
√ Would you like to use App Router? (recommended) ... No
√ Would you like to customize the default import alias (@/*)? ... No
```

#02. Yarn Berry

> React.js와 동일

#03. styledComponents를 위한 구성

[1] 필수 패키지 설치

react와의 차이점은 styledComponents의 사용을 위해서 `babel-plugin-styled-components`를 추가로 설치해야 한다는 부분이다.

기타 패키지도 React.js와 동일

```
$ yarn add styled-components babel-plugin-styled-components ...기타 패키지 설치...
```

[2] 설정파일 수정

`/next.config.js` 파일을 아래와 같이 수정한다.

```
/** @type {import('next').NextConfig} */
const nextConfig = {
    //reactStrictMode: true,

    /** ServerSide에서 StyledComponent가 작동하게 함 */
    compiler: { styledComponents: true }
};

module.exports = nextConfig;
```

#04. 기본 코드 구조

폴더	설명
`/src/pages`	이 폴더 하위의 파일 구조에 따라 URL이 자동 맵핑된다. 첫 페이지의 이름은 `index.js`

> `pages` 폴더 이외에는 `src` 폴더 안에 자유롭게 구성 가능함.

#05. 예약된 파일

[1] `/src/pages/_document.js` (중요)

styledComponents의 정상적인 작동을 위해서 아래 코드 적용

```
import Document, { Html, Head, Main, NextScript } from 'next/document'

// styledComponent를 사용하기 위한 참조
import { ServerStyleSheet } from 'styled-components';

class MyClass extends Document {

    /**
     * 초기화 함수(고정코드)
     * 컴포넌트 전역에서 styledComponent를 사용할 수 있게 한다.
     */
    static async getInitialProps(ctx) {
        const sheet = new ServerStyleSheet();
        const originalRenderPage = ctx.renderPage;

        ctx.renderPage = () => originalRenderPage({
            enhanceApp: (App) => (props) => sheet.collectStyles(<App {...props}
/>),
            enhanceComponent: (Component) => Component,
        });

        const initialProps = await Document.getInitialProps(ctx);

        return {
            ...initialProps,
            styles: [initialProps.styles, sheet.getStyleElement()]
        }
    }

    /**
     * 화면 렌더링 함수 -> Html, Head, Main 첫 글자가 대문자임에 주의
     */
    render() {
        return (
            <Html>
```

```
                <Head>
                    {/*
                        <head>는 순수 HTML태그.
                        <Head>는 next.js의 컴포넌트.
                        이 안에서 charset과 viewport 지정은 자동으로 이루어진다.
                        그 외에 개발자가 적용하고자 하는 외부 CSS나 JS리소스 참조, SEO
구현등을 처리할 수 있다.
                    */}
                    {/*  getInitialProps에서 리턴한 styleTags를 출력한다. */}
                    {this.props.styleTags}
                </Head>
                <body>
                    {/*
                        이 구조를 기본으로 적용한 상태에서 일반 페이지용 js들이 이 위
치에 출력된다.
                        만약 _app.js가 정의되어 있다면 _app.js의 구조를 먼저 적용한
후에 페이지가 표시된다.
                    */}
                    <Main />
                    <NextScript />
                </body>
            </Html>
        )
    }

}

export default MyClass;
```

[2] /src/pages/_error.js

404 에러가 발생한 경우 노출될 페이지.

함수형 컴포넌트로 작성되며 함수의 이름은 페이지 구성에 영향을 주지 않는다.

[3] /src/pages/_app.js

프로젝트 생성시 기본 제공됨 (내용 추가가 필요함)

모든 페이지들에게 적용되는 공통 컴포넌트.

헤더, 푸터를 구현하는 용도로 사용된다.

이 페이지에서 return하는 JSX 내용이 _document.js의 <body>안에 표시된다고 이해할 수 있다.

개발자가 작성하는 일반 페이지에서 return 하는 내용은 이 파일에 포함된다.

```
[개발자코드(ex: hello.js)] -----> [_app.js] -----> [_document.js]
```

- 개발자코드 : 실제 페이지 내용을 구현
- _app.js : 개발자 코드의 주변부에 header, footer를 덧붙임

- _document.js : _app.js가 구성한 내용을 `<html><head>...</head><body>...</body></html>` 구조에 포함시킴