NEXT.js → Static side generation
→ Server side rendering
↳ framework for React

Use of Server-side Rendering ?

↳ SEO crawlers can read site content easily

React → only shows

↓

Why?

↙

cause it renders everthing on client side

while for Next.js you get all of the content because it is server-side rendered.

→ simply exporting as static websites like HTML pages

Why to use ?

① Pre-rendering - Server (entire page is made on server & then sent on browser (client)

② NO-setup for server-side-rendering of React

③ Easy concepts of page creation

④ No configuration for routing

⑤ built in CSS support

File system routing → make pages in pages

⑥ Automatic code splitting

↓

Dynamic routing
(page specific bundles are brought) → when you go to a page then specific content comes.

Install

Quick setup → npx create-next-app app-name or yarn

Manual →

```
cd app-name
yarn dev
```

✡. To change port

```
next dev -p 4000
```

Static / Dynamic Routing Page

↓ /About, /posts

↓ /posts/1 = dynamic

<Link> from next/link

<Link href={`/posts/${post.id}`}>

for dynamic we use [id].js

↓

changing parameter

```
<Link href='/posts/[id]' as={`/posts/${post.id}`}
```

```
const posts = [
    {
    },
    {
    },
]
```

```
function
const getPostDataById (id)
{
  for (int i=0; i < posts.length;
                          i++)
{
   if (posts[i].id ===
       parseInt(id))
  { return posts[i];
  }
}
}
```

→ Don't put string directly in Link
+
use a tag
<Link> <a>... </Link>

Static & Server Rendering of page using API

① Pre-rendering is common in both i.e generates HTML page in advance (instead of client side JS) (better performance & SEO)

Static generation is recommended (HTML is generated at build time (used same on each req.)

server-side → HTML is generated on each request

imp Algorithms

[5, 14, 1, 1]

Mayority element?

↗ length of array

→ element present > $\frac{N}{2}$ times

.....d 3 times

Static → performance + cached

client side is also an option

fetching external data or Api dependent

dynamic content → getStaticProps

dynamic paths → getStaticPaths
(unknown)

```
export async function getStaticProps () {
  const res = await fetch (' ');
  const posts = await res.json();
  return {
    props = {
      posts,
    },
  }
```

```
export default function
Posts ({posts}) {
```

on build it hits the api once and creates HTML for it no loading or hitting again

element present > $\frac{N}{2}$ times

.... 3 times

Static → performance + cached

client side is also an option

fetching external data or api dependent

dynamic content → getStaticProps

dynamic paths → getStaticPaths
(unknown)

```
export async function getStaticProps () {
  const res = await fetch ('_____');
  const parts = await res.json();
  return {
    props = {
      parts,
    },
  }
```

```
export default function
Parts ({parts}) {
```

on build it hits the api once and creates HTML for it no loading or hitting again

Static → performance + cached

client side is also an option

fetching external data or Api dependent

dynamic content → getStaticProps

dynamic paths (unknown) → getStaticPaths

```
export async function getStaticProps (){
  const res = await fetch('_____');
  const posts = await res.json();
  return {
    props = {
      posts,
    },
  }
```

```
export default function
Posts ({posts}) => {
```

on build it hits the api once and creates HTML for it no loading or hitting again

Server-side

```
export async function getServerSideProps () {
  const res = await fetch ('_____').
  const posts = res.json ();
  return {
  props : {
    posts},
  }
},
```

Note: fetch on requests ~~on~~ not build like static

For dynamic routes

```
const async function getServerSideProps ({query})
{
    const {id} = query;
  const res = await fetch (`_____${id}`)
  cons posts = res.json();
  return {
  props : {
    posts,
  }
}
```

Static side (starting 5 to 10) you can convert while others for server side.

to save some dynamic paths for build as static we use .

```
export async getStaticPaths(){
  const paths = ["/posts/1","/posts/2"];
  return {paths, fallback: false}
}
```

false → no new fetch
true → new page requests

add one line in getStaticProps

```
export async function getStaticProps({query, params}) {
  const {id} = query || params.
```

```
import {useRouter} from 'next/router'

const router = useRouter()

if(router.isFallback){
  return <div>Loading...</div>
```

on true it will show loading

after one request it caches and renders it.

CSS in NEXT.js

pages → _app.js (to overwrite app.js)

↑ this will be preferred.

```
import "../styles/global.css"
export default function customApp ({Component, pageProps})
{ return <Component {...pageProps} /> }
```

// to pages folder make styles folder → global.css

css in this will be applied globally

Component level css

pages → posts → components

↓

~~Page~~ Post

↓ ↘

Post.js Post.module.css

className={styles.class}

.class {

}

Sass or Scss Support

npm install sass or yarn add sass

global - scss

nesting

just change css to scss

Bootstrap or other css framework

↳ install → node_modules

↳ imported globally in _app.js

```
import "bootstrap/dist/css/bootstrap.min.css";
```

Styled components

install
import
use like react

serve static files

public folder → for static files

public → images → posts

```
<img src={`/images/posts/${id}.jpeg`} />
```

Environment variables

(passwords, api-tokens, secrets)

root → .env.local

```
API_BASE_URL = ~~ ~~ ;
```

to use API_BASE_URL

```
fetch(`${process.env.}/posts`)
```

.gitignore → · .env.local
· env.development.local
· env.test.local
· env.production.local

· env.development

POST_URL = /posts

fetch (`${process.env.API_BASE_URL}${process.env.POST_URL}`)

if not in local then development

Exposing env. variables to Browser

use NEXT_PUBLIC_ as prefix
to make it available to frontend

Imports (Dynamic)

Static → import Header from ' '

Dynamic → import dynamic from 'next/dynamic';

```
const Header = dynamic(() =>
  import("./com/__"), {
  loading: () => <p>loading...</p>,
}, ssr: false,
```

Routing

① import {useRouter} from 'next/router';

② const router = useRouter();

withRouter (higher order component)

Router API

import Router from 'next/router'

for static onClick = (()=> Router.push('/about'));

for dynamic onClick = (()=> Router.push('/post/[pid]', pattern
'/post/abc')}>
value

Replace

Router.replace('/home');

Prefetch faster client side transitions

Router.prefetch(url, as)
path to page in pages directory
optional used for dynamic routes

Breadcrumb or back

Router.back()

for reload

Router.reload();

Link

import Link from 'next/link'

Static
```
<Link href="___">
  <a className> ____ </a>
</Link>
```

Dynamic `<Link href="/post/[pid]" as="/post/abc">`

Head

import Head from 'next/head'

- document.js in pages folder to manipulate HTML head tags.

```
<Html lang='en'>
<HTML>
<Head>
  <title> this is nextJS </title>  - different for every page.
  <meta>
```

next.config.js

syntax

```
module.exports = {
  basePath: '/docs',
}
```

for subpath folder

Rewrite to show different URL while your actual are different.
Redirects