# What’s ‘Next’ in JSS?

**Nick Allen**

June 5th, 2021

**#sugcon**

SUGCON
GLOBAL 2021

# Introduction

Independent Sitecore architect based in the UK

1st Sitecore project in 2007

MVP 2021 Technology

SITECORE

thinkfresh

## Agenda

- Why Next?
- Workflow
- Getting started
- Pre-rendering options (SSG, ISG, SSR)
- Routing
- Demo solution
  - Layout & components
  - Routing
  - Vercel deployments

# Why Next?

- Next is a framework built on top of React
- Next is an evolution of React born out of Jamstack lessons learned in the enterprise
- Hybrid static and server side rendering
- Incremental static regeneration
- API routes to serverless functions
- Simplified implementation
- Developer experience

## Workflow

SUGCON GLOBAL 2021

Full-stack development

- Windows
- Docker
- Sitecore Content Serialization (SCS)
- Sitecore-first
- Connected to Sitecore

NEXT.JS +

Front-end development only

- Any OS supported by Node
- Code-first
- Disconnected from Sitecore

NEXT.JS

## Getting started

NEXT.JS

dotnet new -i Sitecore.DevEx.Templates --nuget-source https://sitecore.myget.org/F/sc-packages/api/v3/index.json

# PowerShell

SUGCON GLOBAL 2021

Open PowerShell as an administrator and install the .NET starter template

```
> dotnet new -i Sitecore.DevEx.Templates --nuget-source https://sitecore.myget.org/F/sc-packages/api/v3/index.json
```

Create the solution

```
> cd <solutions directory>
> dotnet new sitecore.nextjs.gettingstarted -n <project name>
```

Prepare the container environment (certificate generation, update .env and hosts file)

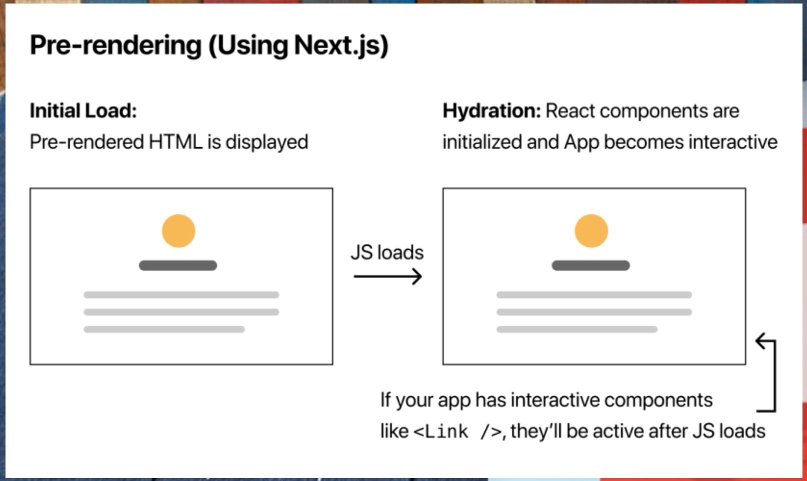
```
> cd <project directory>
> .\Init.ps1 -InitEnv -LicenseXmlPath <path to license> -AdminPassword <b?>
```

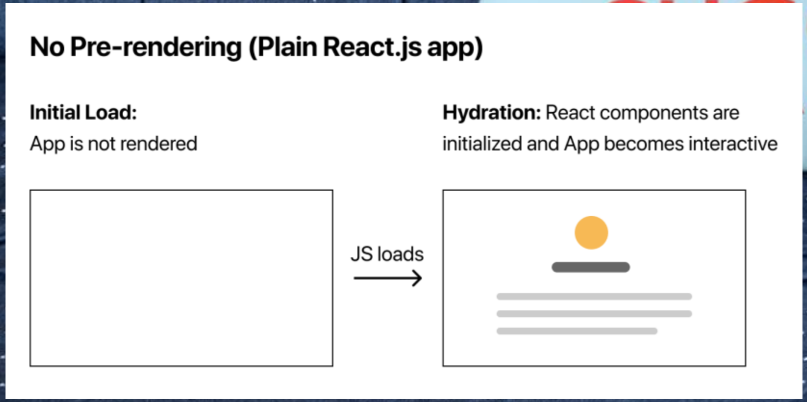
Download images, configure and start containers

```
> .\up.ps1
```

# Pre-rendering options (SSG, ISG, SSR)

Pre-rendering is one of the most important concepts in Next.js


Pre-rendering (Using Next.js)
Initial Load:
Pre-rendered HTML is displayed
JS loads
Hydration: React components are initialized and App becomes interactive
If your app has interactive components like <Link />, they'll be active after JS loads

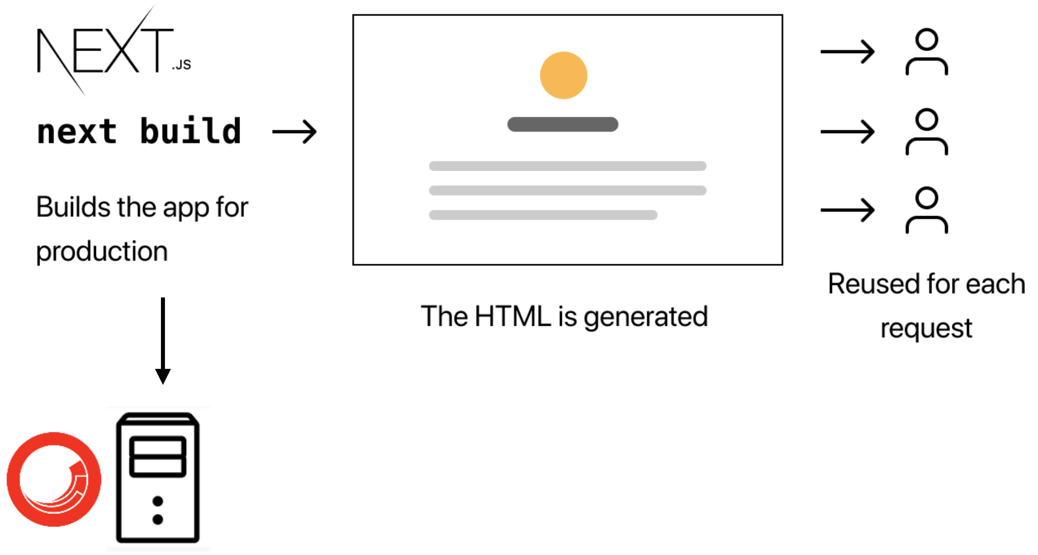

No Pre-rendering (Plain React.js app)
Initial Load:
App is not rendered
JS loads
Hydration: React components are initialized and App becomes interactive

# SSG vs. SSR

SUGCON GLOBAL 2021

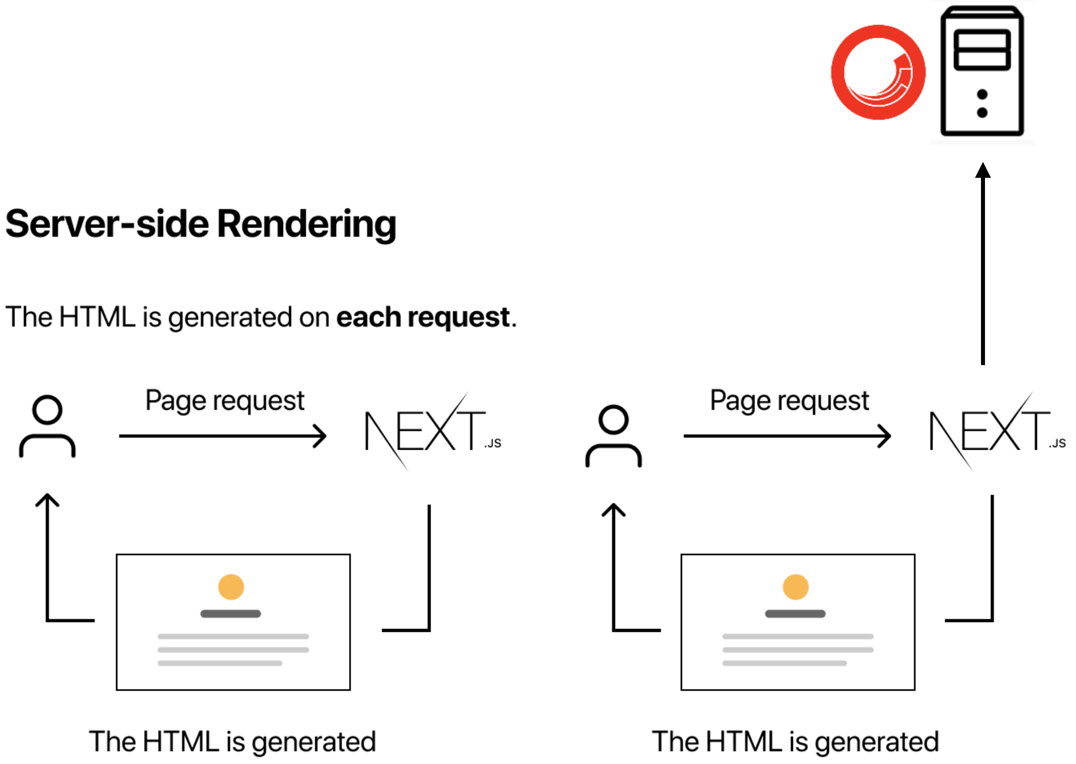
## Static Generation

The HTML is generated at **build-time** and is reused for each request.



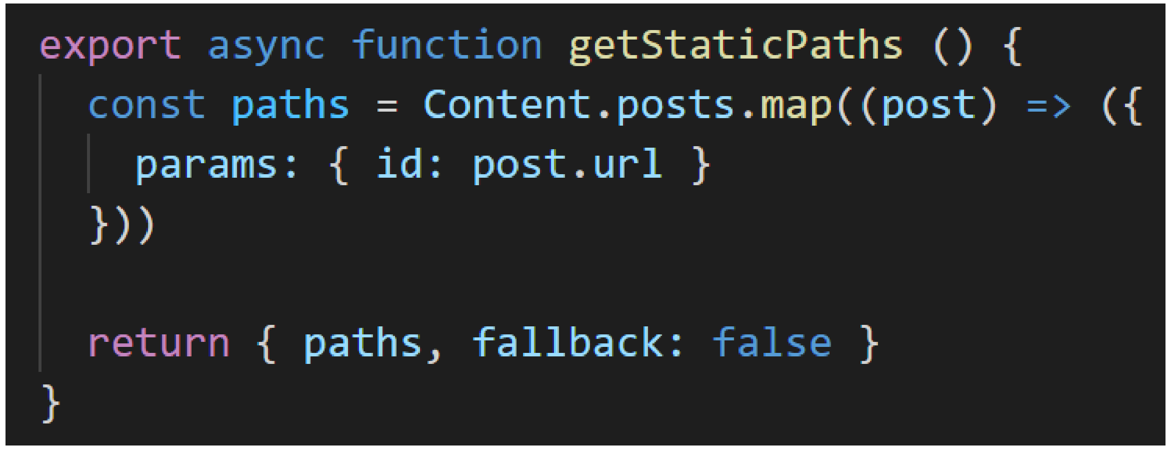
## Server-side Rendering

The HTML is generated on **each request**.



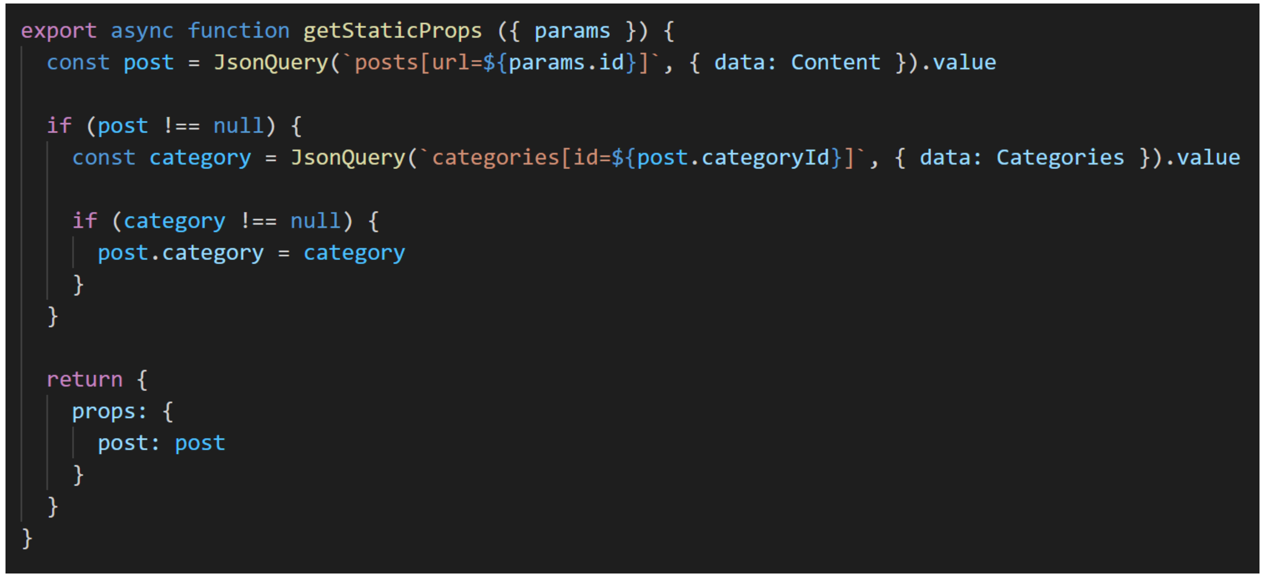
# SSG Functions

SUGCON GLOBAL 2021



```
export async function getStaticPaths () {
  const paths = Content.posts.map((post) => ({
    params: { id: post.url }
  }))

  return { paths, fallback: false }
}
```

At build time an array of blog post ids (the URL in this case) is generated from a Json file containing an index of all available posts.



```
export async function getStaticProps ({ params }) {
  const post = JsonQuery(`posts[url=${params.id}]`, { data: Content }).value

  if (post !== null) {
    const category = JsonQuery(`categories[id=${post.categoryId}]`, { data: Categories }).value

    if (category !== null) {
      post.category = category
    }
  }

  return {
    props: {
      post: post
    }
  }
}
```

At build time data is fetched from Json files on the file system containing the contents of each blog post using the ids generated by getStaticPaths().

# ISG Functions

```
export async function getStaticPaths () {
  const paths = Content.posts.map((post) => ({
    params: { id: post.url }
  }))

  return { paths, fallback: false }
}
```

At build time an array of blog post ids (the URL in this case) is generated from a Json file containing an index of all available posts.

```
export async function getStaticProps() {
  return {
    props: await getDataFromCMS(),
    // we will attempt to re-generate the page:
    // - when a request comes in
    // - at most once every second
    revalidate: 1
  }
}
```

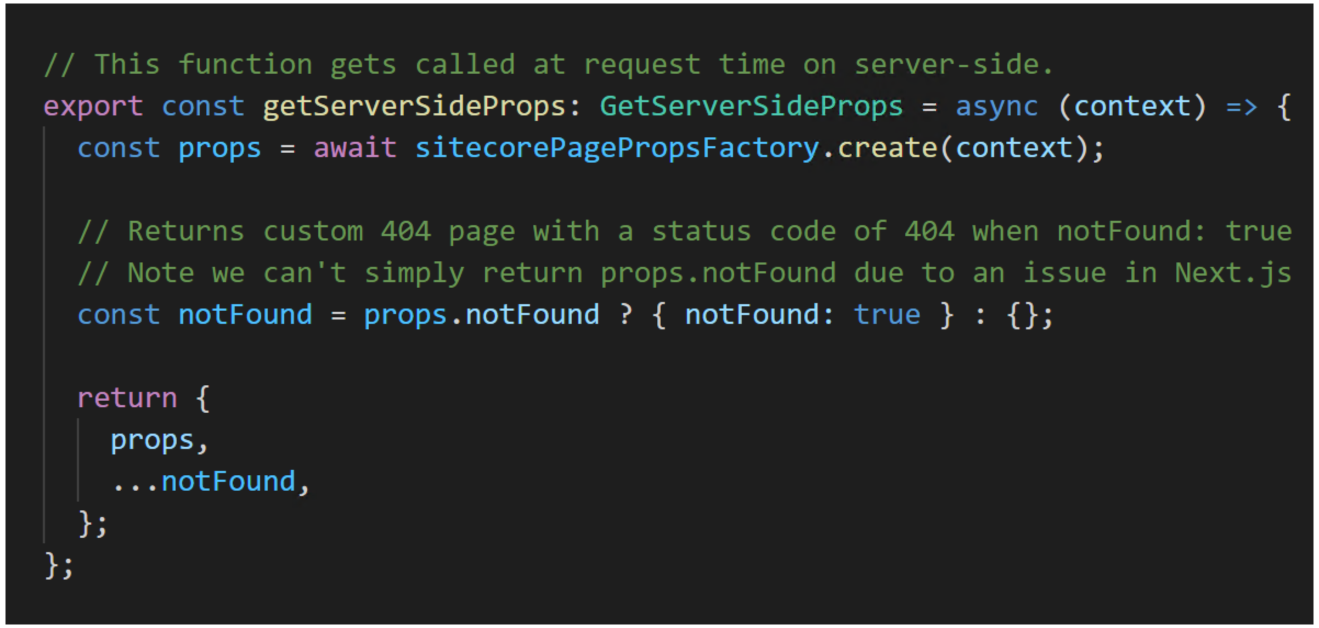
Background regeneration ensures traffic is served uninterruptedly, always from static storage, and the newly built page (or component) is pushed only *after it's done regenerating*.

Now we have static content that is also dynamic!

## SSR Functions

Because getServerSideProps is called at request time, its parameter “context” contains request specific parameters.

The results cannot be cached by a CDN without extra configuration.



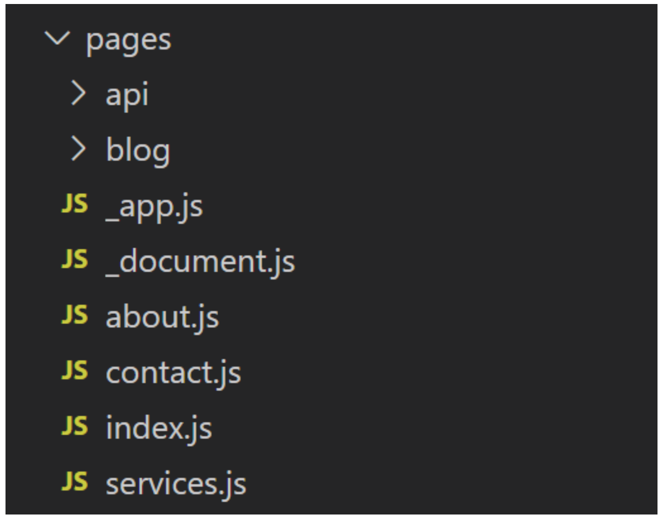
```
// This function gets called at request time on server-side.
export const getServerSideProps: GetServerSideProps = async (context) => {
  const props = await sitecorePagePropsFactory.create(context);

  // Returns custom 404 page with a status code of 404 when notFound: true
  // Note we can't simply return props.notFound due to an issue in Next.js
  const notFound = props.notFound ? { notFound: true } : {};

  return {
    props,
    ...notFound,
  };
};
```

## Routing

Data fetching & dynamic routes

# Dynamic Routes

For simple scenarios we can create pre-defined routes in the pages directory of our application such as:



```
pages
  api
  blog
  _app.js
  _document.js
  about.js
  contact.js
  index.js
  services.js
```

However for more complex applications such as Sitecore we need to take advantage of Next.JS dynamic routes, which behave a little like wildcard items in Sitecore.

| Type | Example | Matches |
|---|---|---|
| Param | blog/[path].tsx | blog/1<br>blog/2<br>blog/n |
| Catch all | blog/[...path].tsx | blog/1<br>blog/1/edit<br>blog/foo/bar |
| Catch all optional | blog/[[...path]].tsx | **blog**<br>blog/1<br>blog/1/edit |

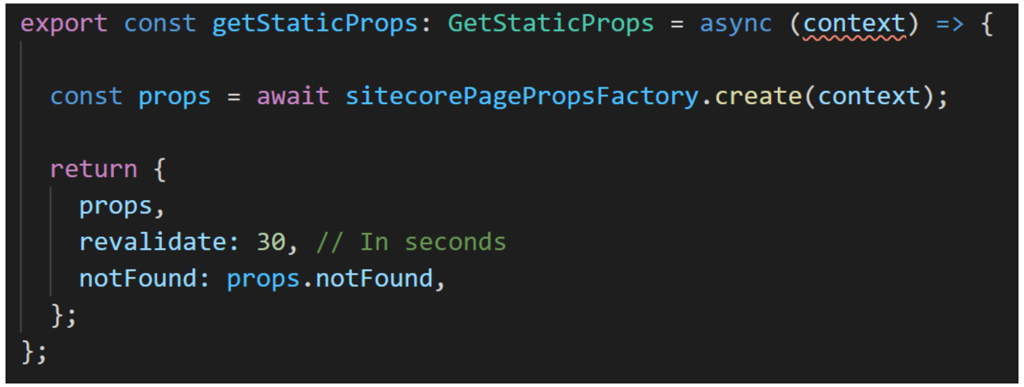
# Dynamic Routes Boilerplate

SUGCON GLOBAL 2021

By default the boilerplate solution created for you during the getting started step contains a single "Catch all optional" route:

[[...path]].tsx

And this route uses ISG. So effectively by default ISG is enabled for all routes.



```
export const getStaticProps: GetStaticProps = async (context) => {

  const props = await sitecorePagePropsFactory.create(context);

  return {
    props,
    revalidate: 30, // In seconds
    notFound: props.notFound,
  };
};
```

It's important to note that the context passed into the:

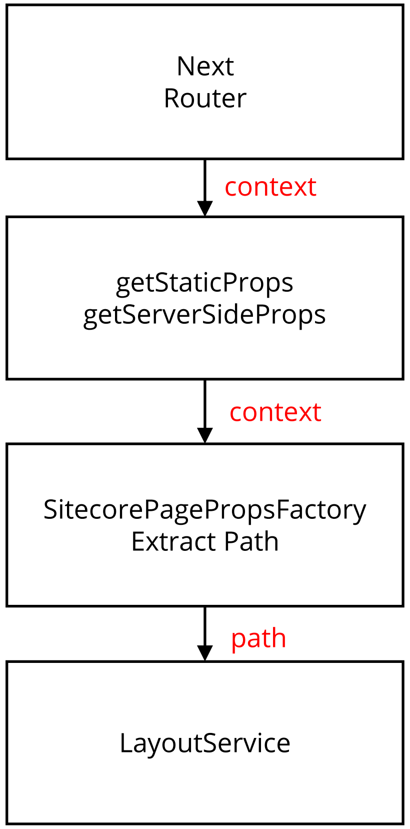
- getStaticProps
- getServerSideProps

functions contains information about the parameters in the current route but **only when using dynamic routes**. This is Next.JS behaviour by design.

| Route | Request | Context |
|---|---|---|
| blog/[**path**].tsx | blog/1 | { params: { **path**: ["1"] } } |
| blog/[...**id**].tsx | blog/1/edit | { params: { **id**: ["1", "edit"] } } |

# Dynamic Routes Page Props Factory

Page props factory is responsible for (among other things) mapping the dynamic route information passed in the context, to a Sitecore path in order to retrieve the correct data from the layout service.


Next Router
context
getStaticProps getServerSideProps
context
SitecorePagePropsFactory Extract Path
path
LayoutService

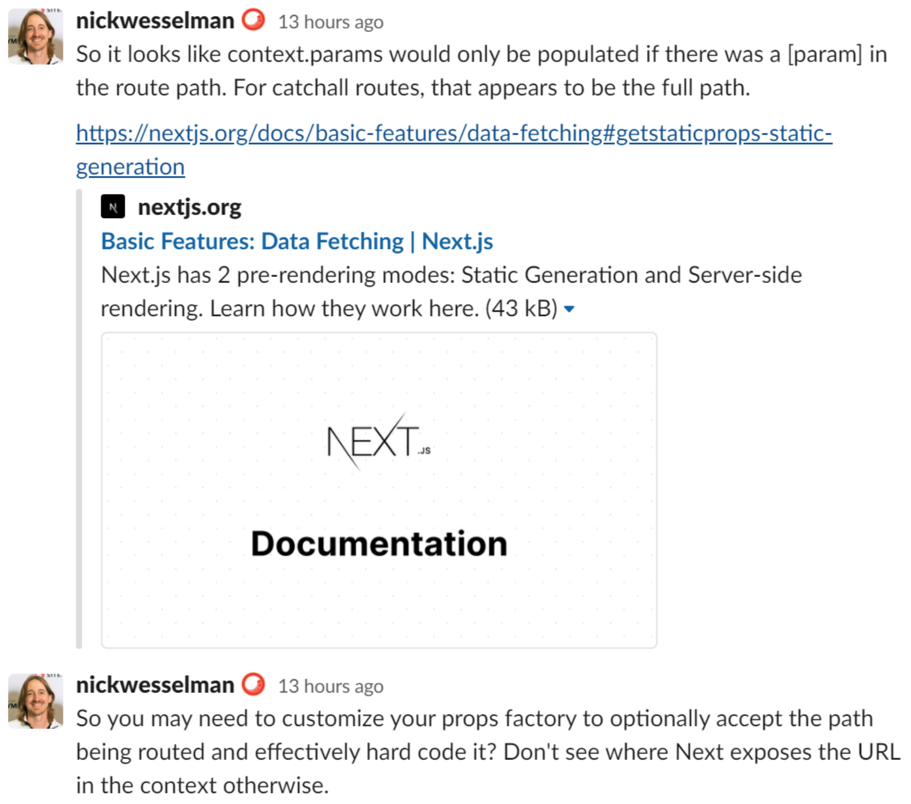
**Scenario A**

- I want to "takeover" a page using a pre-defined route
- foo.tsx

**Scenario B**

- I want to "takeover" a section of pages using a dynamic route
- blog/[[...slug]].tsx

Both of these scenarios are possible but you need to do a little work in the **SitecorePagePropsFactory**.



**nickwesselman** 13 hours ago
So it looks like context.params would only be populated if there was a [param] in the route path. For catchall routes, that appears to be the full path.

https://nextjs.org/docs/basic-features/data-fetching#getstaticprops-static-generation

**nextjs.org**
Basic Features: Data Fetching | Next.js
Next.js has 2 pre-rendering modes: Static Generation and Server-side rendering. Learn how they work here. (43 kB)

NEXT.js
**Documentation**

**nickwesselman** 13 hours ago
So you may need to customize your props factory to optionally accept the path being routed and effectively hard code it? Don't see where Next exposes the URL in the context otherwise.

# Let’s see it in action

- Solution walkthrough
- Layout & components
- Vercel deployments

*I am a proud community member! Please contact me on the following handles:*

**nick.allen**
sitecorechat.slack.com

**@thinkfreshnick**
twitter

**nick-allen**
sitecore.stackexchange.com

# Thank you!

hello@thinkfreshfreelance.co.uk

https://thinkfreshfreelance.co.uk

thinkfresh