**Next.js**

Contents

[Head component 2](#_Toc96507096)

[Image component 2](#_Toc96507097)

[Pre-rendering problems that we face 2](#_Toc96507098)

[Page Pre-Rendering 2](#_Toc96507099)

[Static Site Generation (SSG) 2](#_Toc96507100)

[export async function getStaticProps(context) 3](#_Toc96507101)

[Incremental static regeneration 3](#_Toc96507102)

[export async function getStaticPaths(context) 3](#_Toc96507103)

[Server-side rendering (SSR) 4](#_Toc96507104)

[export async function getServerSideProps(context) 4](#_Toc96507105)

# Head component

Anytime we need to pass information into

      <Head>

        <title>Add a New Meetup</title>

        <meta

          name="description"

          content="Add your own meetups and create amazing networking opportunities."

        />

      </Head>

Or dynamic value

      <Head>

        <title>{props.meetupData.title}</title>

        <meta name="description" content={props.meetupData.description} />

      </Head>

# Image component

With next we have like a middle service between the client and the server, for the client request, and it can do very smart stuff, like page rendering, and also optimization for the images, that’s why we use the Image tag, because it transforms in webp format that Is smaller but doesn’t lose quality, so the pages load a lot faster than normal React that sends the whole bundle to the client when opening the page.

# Pre-rendering problems that we face

When we use backend data with useEffect, the initial state is gonna be empty, for the initial render of the component we are gonna have an empty array, only at the second render we will get the data, this means that the html page provided to the client will not contain that data, so SEO crawlers will not see it.

The pre-rendered html page rendered by Next.js takes the first cycle, it doesn’t wait for the useEffect data to finish.

We want to pre-render a page with data for which we have to wait, we have to tell Next.js.

# Page Pre-Rendering

Next.js has built in pre rendering, but the content is the first render cycle. (missing data)

Hydrate with React code once loaded. React takes control, useEffect, useState, page updated on the browser, not the server.

Two forms of pre-rendering:

Static generation and server side generation

# Static Site Generation (SSG)

In order to wait for data, we need to export getStaticPages from the pages components:

## export async function getStaticProps(context)

export async function getStaticProps() {

  return {

    props: {

      meetups: DUMMY,

    },

  };

}

It will be called before calling the page component.

It can execute server side code, like accessing files and connecting to the db, because it will never get to the client, it is executed during the build process.

## Incremental static regeneration

export async function getStaticProps() {

  return {

    props: {

      meetups: DUMMY,

    },

    revalidate: 10

  };

}

The page gets re-pre-generated every 10 seconds on the server.

## export async function getStaticPaths(context)

Needs to be used when we use getStaticProps inside a dynamic page, but not when we use getServerSideProps. We need to tell next all the possible urls that we can have for this dynamic page, so that I can generate the pages at the build time.

export async function getStaticPaths() {

  return {

    fallback: false,

    paths: [

      {

        params: {

          meetupId: "m1",

        },

      },

      {

        params: {

          meetupId: "m2",

        },

      },

    ],

  };

}

The fallback param set to false means that we have provided ALL the possible paths, and if a user tries to access a path that wasn’t generated, he will get a 404. If true, then next will try to generate that path when requested by the user. IT can also be set to ‘blocking’ to generate that page on demand and cache it. Difference between true and ‘blocking’ is that the user won’t see anything until the page has been generated with blocking, with true the page will be empty until the page is ready to be served.

# Server-side rendering (SSR)

## export async function getServerSideProps(context)

Page build for every request.

Runs always on the server after deployment.

export async function getServerSideProps(context) {

  const req = context.req;

  const res = context.res;

  return {

    props: {

      meetups: DUMMY,

    },

  };

}

Tailwind

Breakpoints. By default styles are for mobile, mobile first. If we want behavior for larger screens we use breakpoints, that are media queries behind, lg:

Firebase v9 uses treeshaking, with a modular approach, instead of using the whole package, you can use just the things that you need, so you have a light package.

Recoil

Global store, atoms,