



Step 1 - NextJS Intro, Pre-requisites

Pre-requisites

You need to understand basic Frontend before proceeding to this track.

You need to know what **React** is and how you can create a simple application in it

NextJS Intro

NextJS was a framework that was introduced because of some **minor inconveniences** in React

1. In a React project, you have to maintain a separate Backend project for your API routes
2. React does not provide out of the box routing (you have to use react-router-dom)
3. React is not SEO Optimised
 1. not exactly true today because of React Server components
 2. we'll discuss soon why
4. Waterfalling problem

Let's discuss some of these problems in the next slides



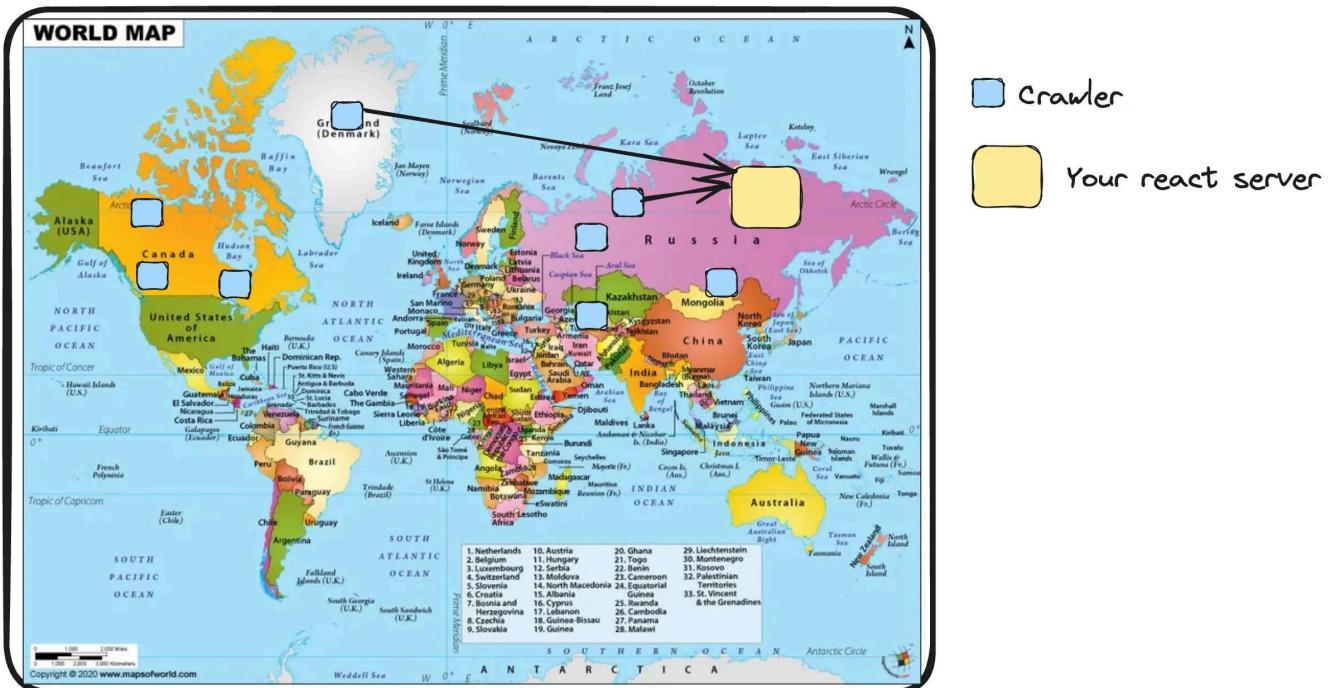


Step 2 – SEO Optimisation

Google/Bing has a bunch of **crawlers** that hit websites and figure out what the website does.

It ranks it on **Google** based on the HTML it gets back

The crawlers **DONT** usually run your JS and render your page to see the final output.



While Googlebot can run JavaScript, **dynamically generated** content is

Try visiting a react website



NextJS (Client side) 1 of 14

back when they visit a website written in react?

Try visiting <https://blog-six-tan-47.vercel.app/signup>

Name	X	Headers	Preview	Response	Initiator	Timing
signup	2			<html lang="en">		
index-C5hDLA4v.js	3			<head>		
index-BvT5dujR.css	4			<meta charset="UTF-8" />		
inpage.js	5			<link rel="icon" type="image/svg+xml" href="/vite.svg" />		
vite.svg	6			<meta name="viewport" content="width=device-width, initial-scale=1" />		
vite.svg	7			<title>Vite + React + TS</title>		
	8			<script type="module" crossorigin src="/assets/index-C5hDLA4v.js" />		
	9			<link rel="stylesheet" crossorigin href="/assets/index-BvT5dujR.css" />		
	10			</head>		
	11			<body>		
	12			<div id="root"></div>		
	13			</body>		
	14			</html>		

Googlebot has no idea on what the project is. It only sees **Vite + React + TS** in the original HTML response.

Ofcourse when the JS file loads eventually, things get rendered but the **Googlebot** doesn't discover this content very well.

Step 3 – Waterfalling problem

Let's say you built a blogging website in react, what steps do you think the **request cycle** takes?

2 Fetching script.js from CDN



NextJS (Client side) 1 of 14

on (if not, redirect them to /login page)

4. Fetching the actual blogs

There are 4 round trips that happen one after the other (sequentially)



The "waterfalling problem" in React, and more broadly in web development, refers to a scenario where data fetching operations are chained or dependent on each other in a way that leads to inefficient loading behavior.

What does nextjs provide you?

Step 4 – Next.js offerings

Next.js provides you the following **upsides** over React

1. Server side rendering – Gets rid of SEO problems
2. API routes – Single codebase with frontend and backend
3. File based routing (no need for react-router-dom)

generation

5 Maintained by the Vercel team



NextJS (Client side) 1 of 14

Downsides -

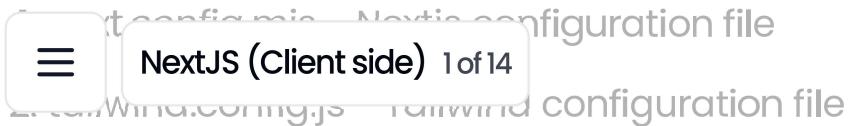
1. Can't be distributed via a CDN, always needs a server running that does **server side rendering** and hence is expensive
2. Very opinionated, very hard to move out of it

Step 5 – Let's bootstrap a simple Next app

npx create-next-app@latest



File structure



3. app - Contains all your code/components/layouts/routes/apis

Bootstrap the project

1. Remove everything from `app/page.tsx` and return an empty div
2. Remove the css bits (not the tailwind headers) from the `global.css` file

Step 6 – Understanding routing in Next

Routing in React

<https://bladecivitan-17.vercel.app/signup>



NextJS (Client side) 1 of 14

Next.js has a **file based router** (<https://nextjs.org/docs/app/building-your-application/routing/defining-routes>)

This means that the way you create your files, describes what renders on a route

1. Let's add a new folder in **app** called **signup**
2. Let's add a file called **page.tsx** inside **app/signup**

▼ page.tsx



```
export default function Signup() {
  return (
    <div>
      hi from the signup page
    </div>
  );
}
```

1. Start the application locally

npm run dev



Final folder structure

Assignment – Can you add a **signin route?**

Step 1: Prettify the signin page

Let's replace the signup page with a prettier one

```
export default function Signin() {
  return <div className="h-screen flex justify-center flex-col">
    <div className="flex justify-center">
      <a href="#" className="block max-w-sm p-6 bg-white border border-gray-200 rounded-lg">
        <div>
          <div className="px-10">
            <div className="text-3xl font-extrabold">
              Sign in
            </div>
          </div>
          <div className="pt-2">
            <LabelledInput label="Username" placeholder="harkirat@gmail.com" type="text" />
            <LabelledInput label="Password" type="password" placeholder="password" />
            <button type="button" className="mt-8 w-full text-white bg-gray-700 py-2 px-4 rounded-lg">Sign in</button>
          </div>
        </div>
        </a>
      </div>
    </div>
  </div>
}
```

```
interface LabelledInputType {
  label: string;
  placeholder: string;
  type?: string;
}
```

```
function LabelledInput({ label, placeholder, type }: LabelledInputType) {
  return <div>
    <label className="block mb-2 text-sm text-black font-semibold pt-4">{label}</label>
    <input type={type} placeholder={placeholder} className="bg-gray-50 border border-gray-300 py-2 px-4 rounded-lg" />
  </div>
```

</div>



NextJS (Client side) 1 of 14

Step 8 – Server side rendering

Let's try exploring the response from the server on the `/signup` route

1. Run `npm run dev`
2. Visit `http://localhost:3000/signup`
3. Notice the response you get back in your HTML file

Now if `GoogleBot` tries to scrape your page, it'll understand that this is a `signup page` without running any Javascript.

The first `index.html` file it gets back will have context about the page since it was `server side rendered`



Step 9 – Layouts

You'll notice a file in your `app` folder called `layout.tsx`

Let's see what this does (Ref <https://nextjs.org/docs/app/building-your-application/routing/pages-and-layouts>)

What are layouts

Layouts let you `wrap` all child `pages` inside some logic.

Let's explore `layout.tsx`

Assignment

Try adding a simple `Appbar`





Step 10 – Layouts in sub routes

What if you want all routes that start with `/signin` to have a `banner` that says `Login now to get 20% off`



Step 11 – Merging routes

What if you want to get the banner in both `signup` and `signin` ?

Approach #1

Move both the `signin` and `signup` folder inside a `auth` folder where we have the layout

You can access the routes at

<http://localhost:3000/auth/signup> and <http://localhost:3000/auth/signin>

Approach #2

You can use create a new folder with `()` around the name.

This folder is `ignored` by the router.

You can access the routes at

<http://localhost:3000/signup> and <http://localhost:3000/signin>



Step 12 – components directory

You should put all your `components` in a `components` directory and use them in the `app routes` rather than shoving everything in the route handler

1. Create a new folder called `components` in the root of the project
2. Add a new component there called `Siginin.tsx`
3. Move the signin logic there
4. Render the `signin` component in `app/(auth)signin/page.tsx`

Solution

▼ components/Siginin.tsx

```
export function Siginin() {  
  return <div className="h-screen flex justify-center flex-col">  
    <div className="flex justify-center">  
      <a href="#" className="block max-w-sm p-6 bg-white border border  
        <div>  
          "I font-extrabold">  
        Sign in  
      </div>  
    </div>
```



NextJS (Client side) 1 of 14

```
</div>
,
<LabelledInput label="Username" placeholder="harkirat@gma
<LabelledInput label="Password" type={"password"} placeholder="Pa
<button type="button" className="mt-8 w-full text-white bg-
    </div>
  </div>
</a>
</div>
</div>
}

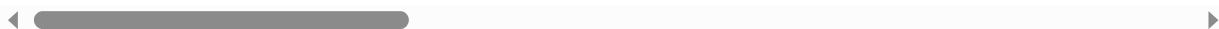
interface LabelledInputType {
  label: string;
  placeholder: string;
  type?: string;
}
```

```
function LabelledInput({ label, placeholder, type }: LabelledInputType) {
  return <div>
    <label className="block mb-2 text-sm text-black font-semibold pt-4">
      <input type={type || "text"} id="first_name" className="bg-gray-50 border
    </div>
}
```

▼ app/(auth)/Signin.tsx

```
import { Signin as SigninComponent } from "@/components/signin";

export default function Signin() {
  return <SigninComponent />
}
```



Step 12 – Add a button onclick handler

NextJS (Client side) 1 of 14

Now try adding a `onClick` handler to the `button` on the signin page

```
<button onClick={() => {  
  console.log("User clicked on signin")  
}} type="button" className="mt-8 w-full text-white bg-gray-800 focus:ring-2
```

You will notice an error when you open the page

What do you think is happening here? Let's explore in the next slide

Step 14 – Client and server components

Ref - <https://nextjs.org/learn/react-foundations/server-and-client-components>

NextJS expects you to identify all your components as either **client** or **server**

As the name suggests

1. Server components are rendered on the server
2. Client components are **pre-rendered** and are pushed to the client to be rendered again

By default, all components are **server** components.

If you want to mark a component as a **client** component, you need to add the following to the top of the component -

"use client"



When should you create **client components** ?

1. Whenever you get an error that tells you that you need to create a client component
2. Whenever you're using something that the server doesn't understand (useEffect, useState, onClick...)

Rule of thumb is to defer the client as much as possible

Assignment

Try updating **components/signin.tsx** to make it a client component

You will notice that the error goes away

Some nice readings -

<https://github.com/vercel/next.js/discussions/43153>**NextJS (Client side)** 1 of 14