

Sr.



CheatSheet

Storyblok NextJS

by Lisi Linhart

Setup

Start by installing the CLI tools for Storyblok and Expo.

```
$ npm init next-app
# or
$ yarn create next-app
```

HINT: Follow the detailed Storyblok tutorial here: <https://www.storyblok.com/tp/next-js-react-guide>

Connect Next.js with Storyblok

The Storyblok client allows you to request content from Storyblok's API, and the storyblok-react module gives you a wrapper component that creates editable blocks in the live visual editor.

```
$ npm install storyblok-js-client
storyblok-react --save
# OR
$ yarn add storyblok-js-client
storyblok-react
```

HINT: Start with the Starter repository: <https://github.com/storyblok/nextjs-multilanguage-website>

Create a Storyblok client

Get the preview token from your Storyblok space settings.

```
import StoryblokClient from
'storyblok-js-client'

this.client = new StoryblokClient({
  accessToken: 'sb-preview-token',
  cache: {
    clear: 'auto',
    type: 'memory'
  }
})
```

Create the Storyblok bridge

Our Storyblok Service allows you to create a bridge to Storyblok. We need to insert this bridge in our **Layout.js**.

```
import StoryblokService from '../
utils/storyblok-service'

const Layout = ({ children }) => (
  <div>
    <div className="max-w-5xl py-10
mx-auto">
      {children}
      {StoryblokService.bridge()}
    </div>
  </div>
)

export default Layout
```

Create components

with SbEditable

To make the components editable in Storyblok we need to pass our blok information to each component.

```
import SbEditable from 'storyblok-react'

const Feature = ({blok}) => (
  <SbEditable content={blok}>
    <div>
      <h2>{blok.name}</h2>
    </div>
  </SbEditable>
)

export default Feature
```

Enable Live Preview

For the Live Preview to work, we need to initialize the event listeners with the **initEditor** function in the Storyblok Service. Add a **componentDidMount** function to your **pages/index.js**.

```
import StoryblokService from
'../../utils/storyblok-service'

export default class extends
React.Component {
  componentDidMount() {
    StoryblokService.initEditor(this)
  }
}
```

Adding another language

As all our routes are dynamic, adding another language is simple. Create a new folder for each language and query the desired language with the Storyblok API.

```
import StoryblokService from
'../../utils/storyblok-service'

let [page, settings] = await
Promise.all([StoryblokService.get(
`cdn/stories/${asPath}`
), StoryblokService.get(
`cdn/stories/${query.language}/
settings`
)])
```

```
<li>
  <Link href="/de/blog">
    <a
      className="top-header__link">
        German
      </a>
    </Link>
  </li>
```

Rendering Richtext

To render Richtext you can use the **storyblok-rich-text-react-renderer** npm package.

```
import { render } from "storyblok-
rich-text-react-renderer"

<div>
  {render(blok.long_text)}
</div>
```

Directory Structure

Storyblok supports a component based approach to edit your content

components

- Feature.js
- Grid.js
- Page.js

Connected components

pages

- [language]/blog
- index.js

Multilanguage pages

utils

- storyblok-service.js

Storyblok connection

Deployment

To render Richtext you can use the **storyblok-rich-text-react-renderer** npm package.

```
import { render } from "storyblok-rich-text-react-renderer"

<div>
  {render(blok.long_text)}
</div>
```

HINT: Deploy your site easily with Vercels auto-detection: <https://nextjs.org/docs/deployment>