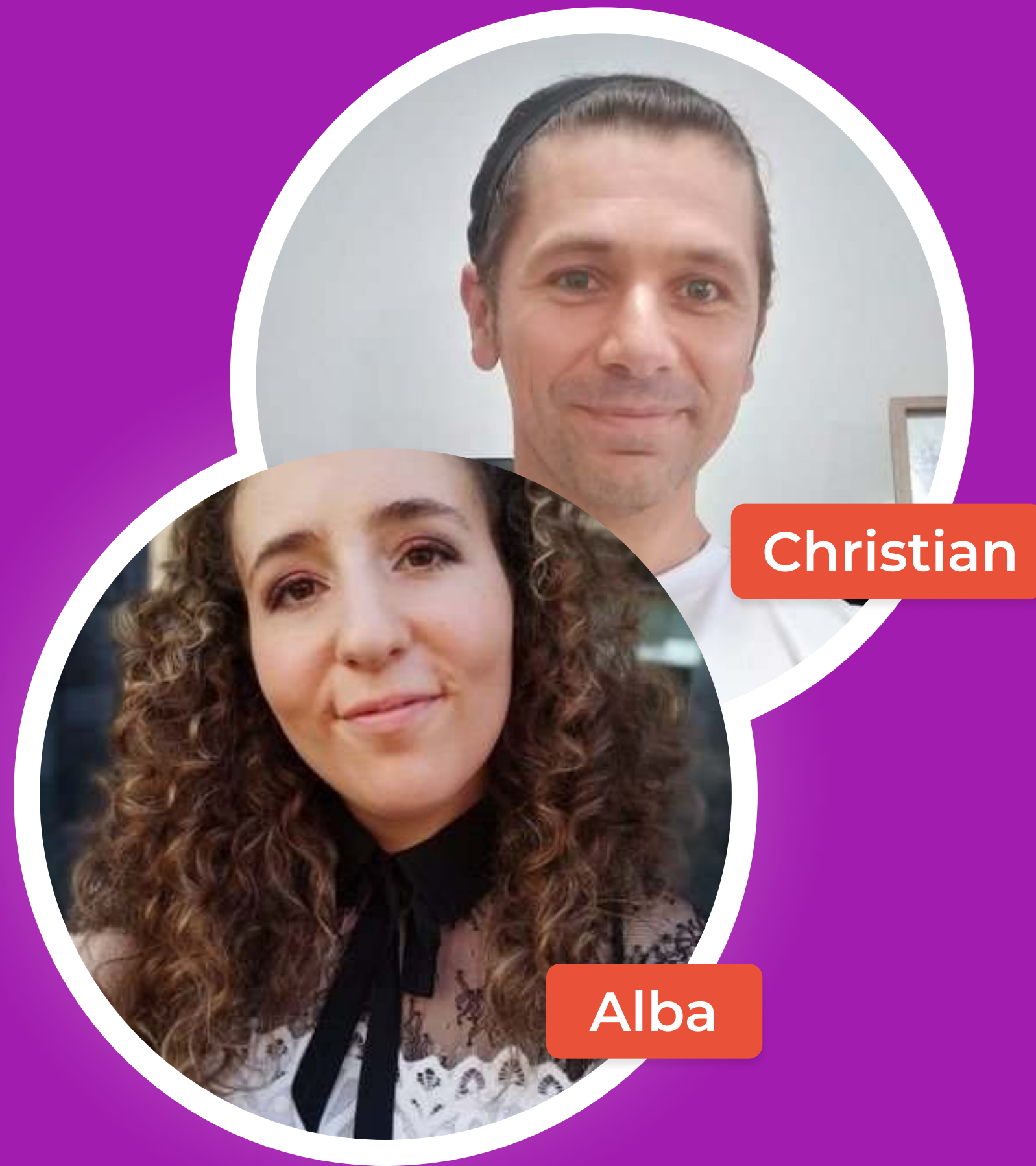


# How to migrate from WP to a Headless CMS



Christian

Alba

# How to migrate from WordPress to a Headless CMS

DAY

1

- Detect when it makes sense to migrate from WP to a Headless setup.
- Create and understand a Storyblok space.
- Create Content Types with the same schema as in WP, and relationships.
- Migrate your WP content to a new Headless CMS, configuring a script.

# INTRODUCTION

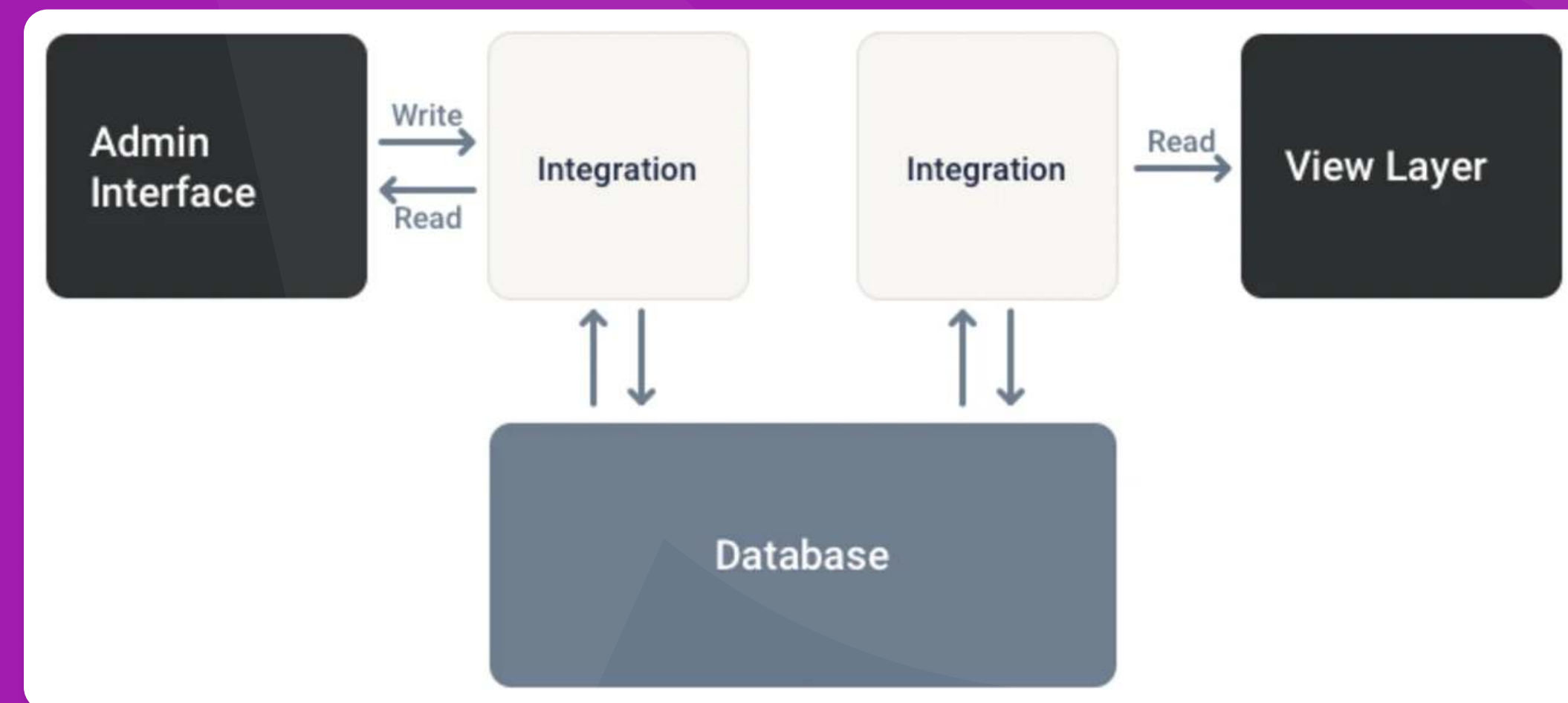
## Monolithic WordPress vs Headless.

- WordPress is the **most used website builder**

You can quickly create websites & has a rich plugin's ecosystem.

- WordPress architecture is **monolithic**

User interface and data access are on the same platform.

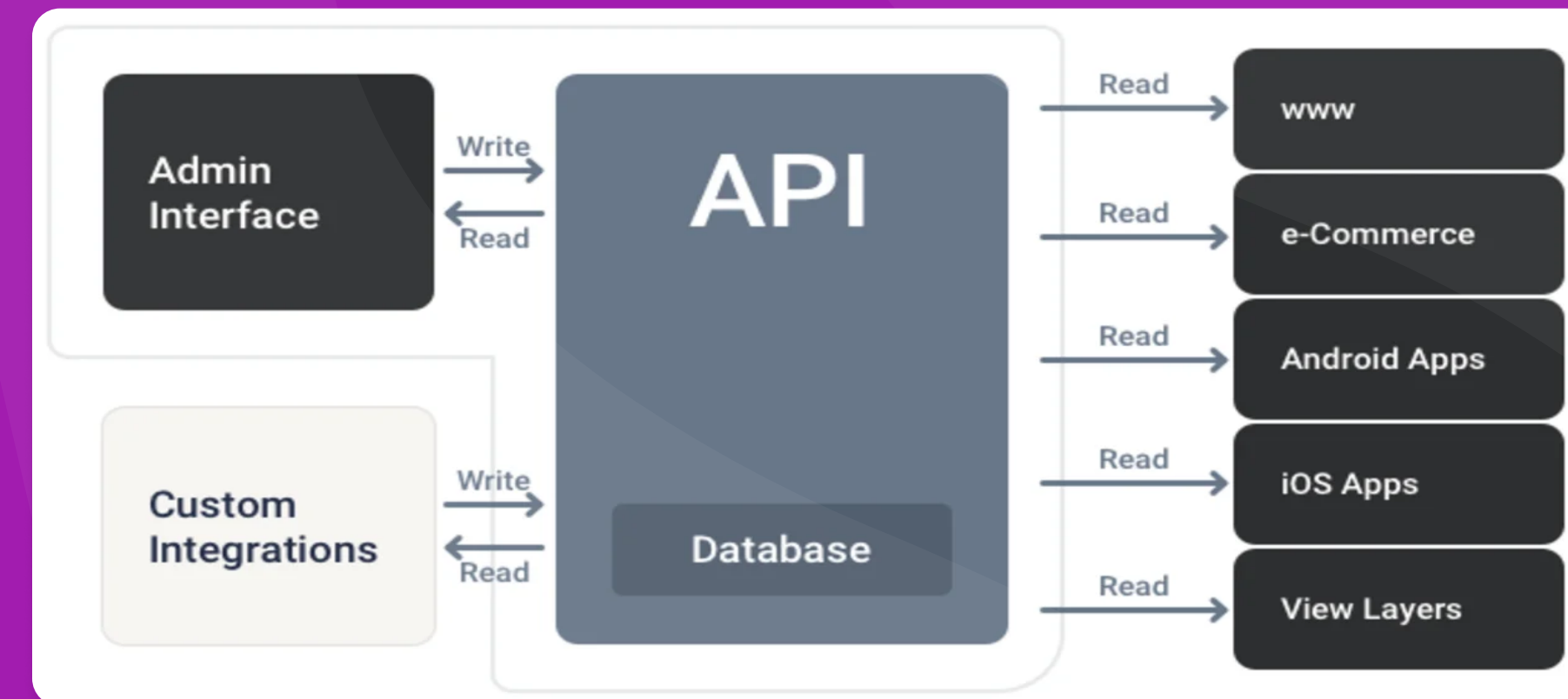


# INTRODUCTION

## Monolithic WordPress vs Headless.

- Using the new REST API WordPress can be headless

Use WP API as a backend and the front-end in a different project.



- Disadvantages

Configure and update WP, make it secure, and depend on its technology for new functionalities.



DAY

1

# WHEN MAKES SENSE

## Migrate from WordPress to a Headless setup.

When you want to...

- **Choose your tech.**  
Develop the front-end independently of the platform where the content is added.
- **Maintain brand consistency.**  
Create a front-end that represents the company's brand and unifies the look&feel of all products.
- **Create multiplatform projects** without depending on different admin panels.
- **Make development easier.**  
Front-end is isolated, you can change the visual appearance without modifying the content structure.

DAY

1

# WHEN MAKES SENSE

## Migrate from WordPress to a Headless setup.

A Headless CMS...

- Takes care of **maintaining** the **security** of the platform and **updating** it on your behalf.
- Takes care of the content **management** and the content **delivery**.  
Allowing to set up custom **content workflows**, so developers can focus on the rendering part.
- Provides services to **optimize assets** and the **speed** of response  
with which they are served to us. All the data being delivered through CDNs.
- Like Storyblok has a **Visual Editor**,  
Allows editors to access the panel to add content, seeing a **preview** of how it will look on production.

DAY

1

# WHEN MAKES SENSE

## Migrate from WordPress to a Headless setup.

“ If you are thinking of migrating a personal site, it may not be necessary, but it could be very useful in the long run.

“ If there're more people involved, not just devs, but content creators, then you should strongly consider adopting Headless CMS.



# DAY 1

- ~~Detect when it makes sense to migrate from WP to a Headless setup.~~
- Create and understand a Storyblok space.
- Create Content Types with the same schema as in WP, and relationships.
- Migrate your WP content to a new Headless CMS, configuring a script.

# GLOSSARY

DAY

1

- **Content Type** (Like *Custom Post Types* on WP)  
Define the type of content entry and can hold the basic fields for your content entries.
- **Nested components** (Blocs)  
A reusable component that you can place inside a Content Type or another component.
- **Story** (Content entry)  
A collection of components filled with information by the content creator. If you want to create a website, a story is a page.
- **Jamstack architecture**  
Set of web development best practices focused on providing the highest performance, security and lowest cost, designed to make the web faster, more secure and easier to scale.

# STORYBLOK SPACE

Create and understand a Storyblok space.



- **My account**

Where we modify our account details and create access tokens for the Storyblok Management API.

- **Content** (Editors section)

Where the content is stored, where the editors/marketers will spend most of their time.

- **Assets**

Where all images are stored, which you can get through the CDN service, optimized and of any size.

- **Components** (DEVs section)

Where you will create the Content Types and Nested components.

- **Settings** (IT team section)

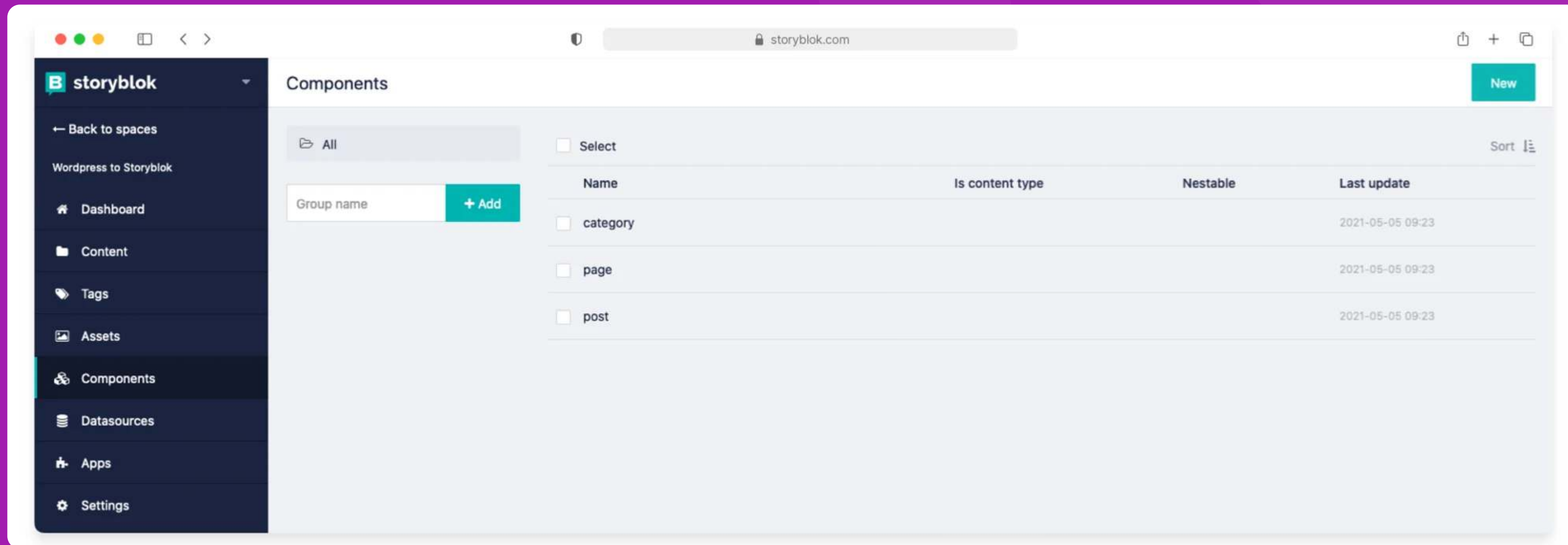
Where space data is configured: the languages, custom workflows, user permissions, etc.

# STORYBLOK SPACE

Create and understand a Storyblok space.

DAY

1



DAY

1

- Detect when it makes sense to migrate from WP to a Headless setup.
- Create and understand a Storyblok space.
- Create Content Types with the same schema as in WP, and relationships.
- Migrate your WP content to a new Headless CMS, configuring a script.



# CONTENT TYPES AND RELATIONSHIPS

## WordPress entries schema: Page, post & category

DAY

1

REST API Handbook

Search ...

Browse: Home / REST API Handbook / Reference / Pages

### Pages [Edit](#)

#### TOPICS

- Schema
- List Pages
  - Definition; Example Request; Arguments
- Create a Page
  - Arguments; Definition
- Retrieve a Page
  - Definition & Example Request; Arguments
- Update a Page
  - Arguments; Definition; Example Request
- Delete a Page
  - Arguments; Definition; Example Request

#### Schema #

The schema defines all the fields that exist within a page record. Any response from these endpoints can be expected to contain the fields below unless the `_filter` query parameter is used or the schema field only appears in a specific context.

<b>date</b> string or null, datetime ( <a href="#">details</a> )	The date the object was published, in the site's timezone. Context: <b>view, edit, embed</b>
<b>date_gmt</b> string or null, datetime ( <a href="#">details</a> )	The date the object was published, as GMT. Context: <b>view, edit</b>
<b>guid</b> object	The globally unique identifier for the object. Read only Context: <b>view, edit</b>

WP 2 Storyblok

Dashboard

Posts [Add New](#)

All (2) | Published (2)

Bulk actions [Apply](#) All dates [Filter](#) All Categories [Filter](#) Search Posts

<input type="checkbox"/>	Title	Author	Categories	Tags		Date
<input type="checkbox"/>	<a href="#">Migrating a page</a>	storyblok	Migration, Page	—	—	Published 2021/10/18 at 6:33 pm
<input type="checkbox"/>	<a href="#">Migrating an entry</a>	storyblok	Entry, Migration	—	—	Published 2021/10/14 at 2:35 pm
<input type="checkbox"/>	Title	Author	Categories	Tags		Date

Bulk actions [Apply](#)

Thank you for creating with [WordPress](#).

Version 5.8.1

# CONTENT TYPES AND RELATIONSHIPS

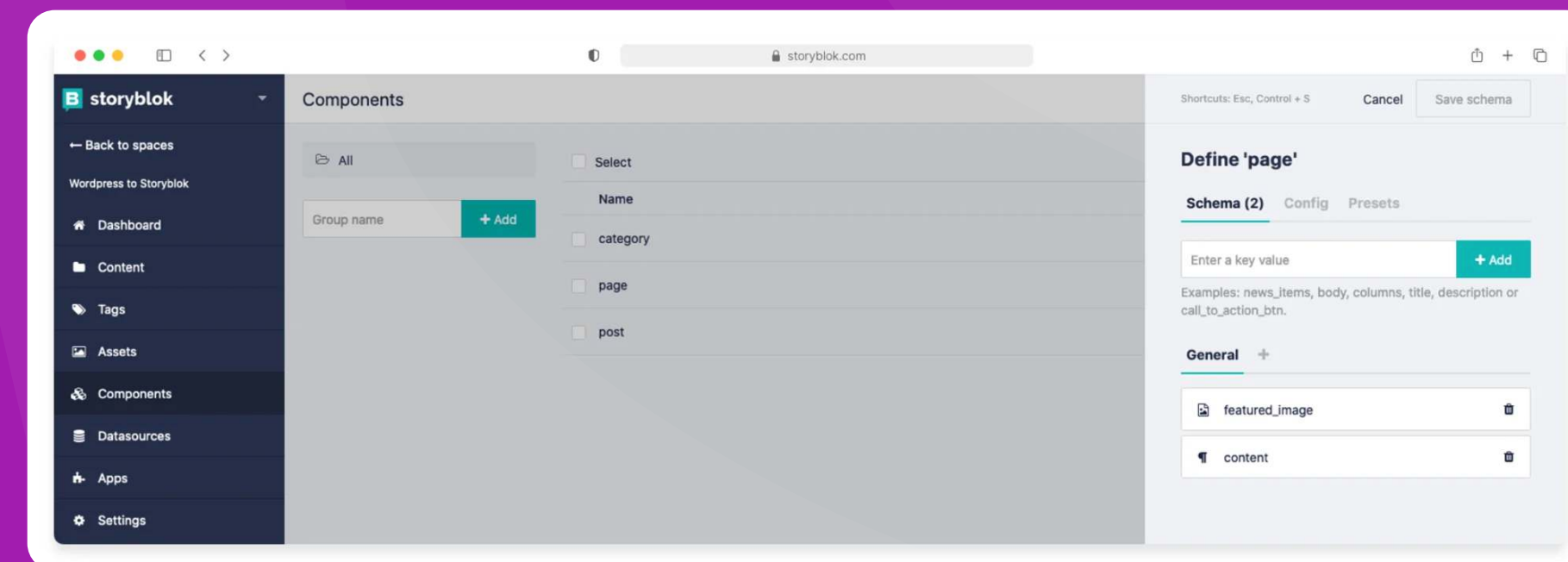
## Creating Content Types in the Storyblok space

DAY

1

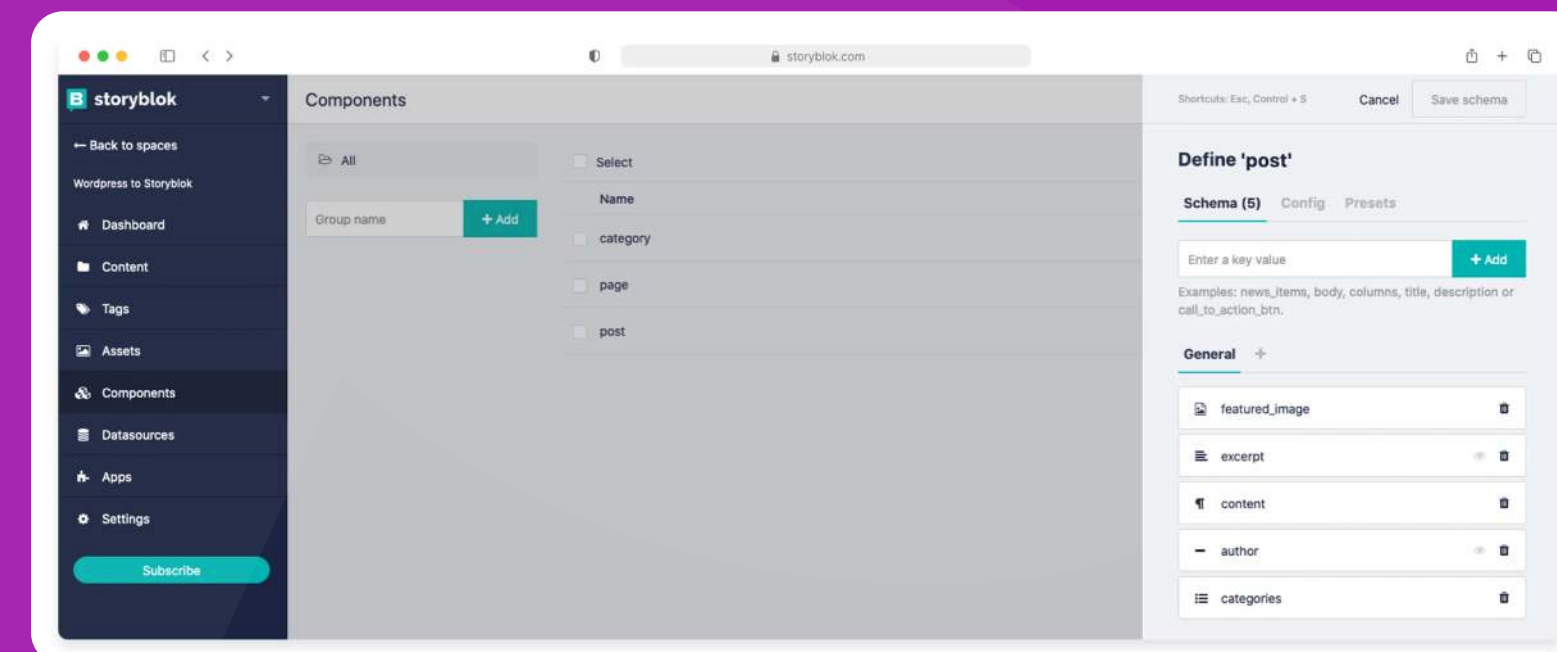
### Page schema

Default Storyblok fields: Name, Slug, Tags, Date.  
Add **featured\_image** (Assets > Images).  
Add **body** (Blocks).



### Post schema

Default Storyblok fields: Name, Slug, Tags, Date.  
Add **featured\_image**.  
Add **excerpt** and **content**.  
Add relationships to other types, such as **Categories**.



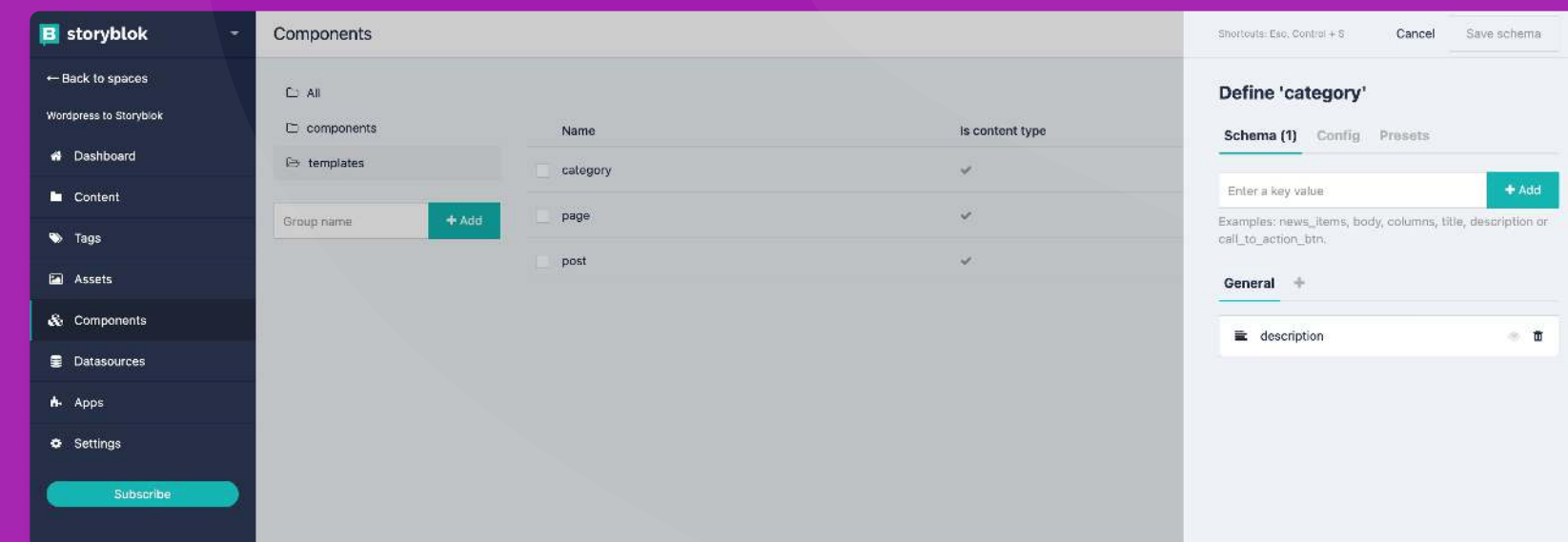
# CONTENT TYPES AND RELATIONSHIPS

## Creating relationships between Content Types

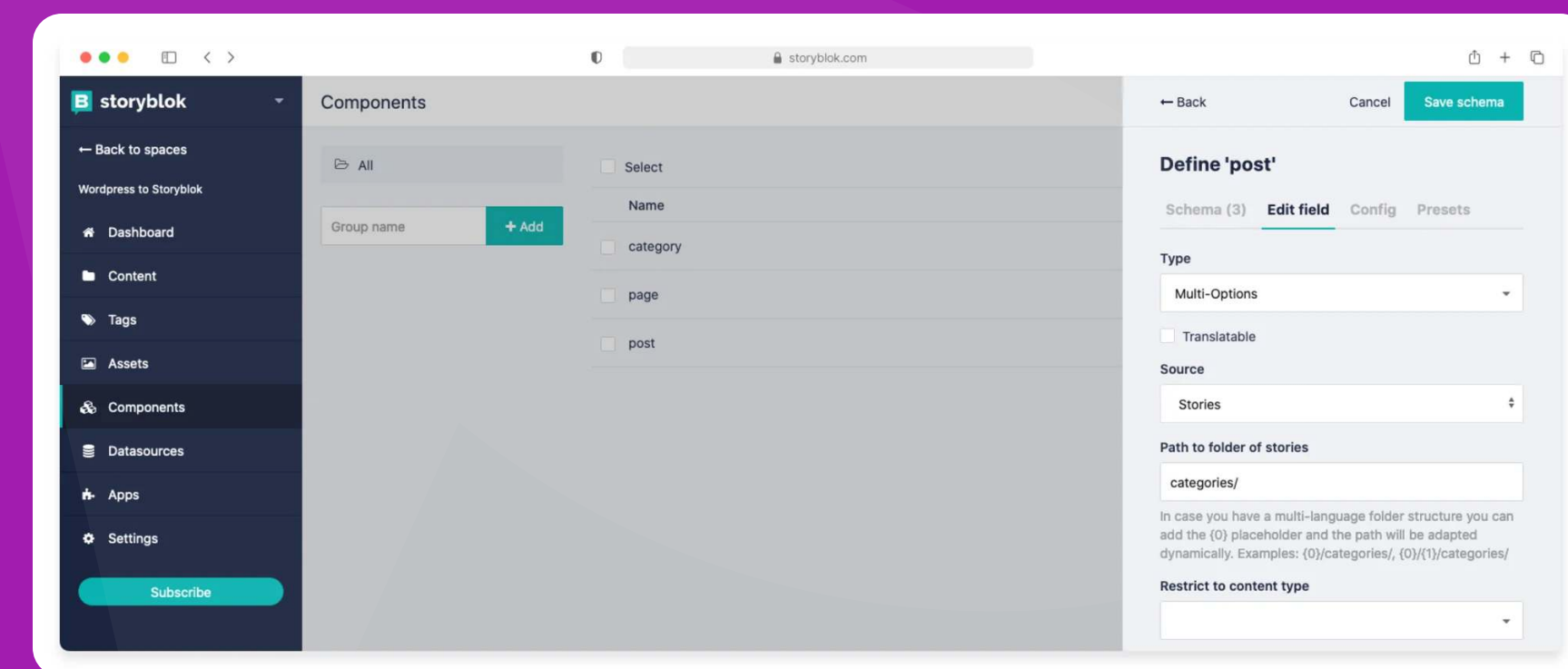
DAY

1

- Category Content-Type  
Default Storyblok fields: name, slug.  
Add **description** (Textarea).



- Categories relationship field  
Add **categories** field in Post (Multi-Options):
  - Source: **Stories**
  - Folder: **categories/**



DAY

1

- Detect when it makes sense to migrate from WP to a Headless setup.
- Create and understand a Storyblok space.
- Create Content Types with the same schema as in WP, and relationships.
- Migrate your WP content to a new Headless CMS, configuring a script.



# MIGRATION USING A SCRIPT

## PREREQUISITES

- WordPress version  $\geq 5$  (REST API v2).
- REST JSON API publicly available. Access to `/wp-json`.
- WP Permalink Settings option selected: **Post name**.
- Node version  $\geq 14$ .
- Storyblok space to obtain the Space\_id and the OAuth token.



# MIGRATION USING A SCRIPT

## STEPS for creating the script

- Clone the wordpress-importer repo:

```
git clone https://github.com/storyblok/wordpress-importer.git
```

- Install NPM packages: `cd ./wordpress-importer && npm install`

- Create `migrate-wp-to-storyblok.js` file in the project's root.

- Add a command in the `package.json` to run the script:

```
"migrate": "node ./migrate-wp-to-storyblok.js"
```

# MIGRATION USING A SCRIPT

## STEPS for defining the script

- Add the REST API URL: <https://wp2.storyblok.com/wp-json>
- Add Space\_id (Settings > General)
- Create & copy/paste the OAuth token (My Account > Personal access tokens)

```
import { Wp2Storyblok } from './index.js'
const wp2storyblok = new Wp2Storyblok('https://wp2.storyblok.com/wp-json', {
  token: '[OAuth token]', // My Account > Personal access tokens
  space_id: [Space_id], // Settings
})
wp2storyblok.migrate()
```

# MIGRATION USING A SCRIPT

## Schema mapping

```
content_types: [  
  {  
    name: 'pages', // Post type in WP  
    new_content_type: 'page', // Equivalent Content type in Storyblok  
    folder: '', // OPTIONAL: To save all the content of the same type in a specific folder  
    schema_mapping: {  
      "title": "name", // "Field in WP": "Field in Storyblok"  
    }  
  }  
]
```

# MIGRATION USING A SCRIPT

## Schema mapping: types of fields

- A simple field, such as title.

```
"title": "name", // "Field in WP": "Field in Storyblok"
```

- A sub-property of a field, as the feature image.

```
"_links.wp:featuredmedia.0": "content.preview_image",
```

- A field migrated to a nested block in Storyblok.

```
"content": {  
  field: 'content.body_items', // Field name in Storyblok  
  component: "rich-text", // Component name inside the above field  
  component_field: "content" // Field name inside the component you are migrating to  
}
```

# MIGRATION USING A SCRIPT

## Taxonomies

```
{
  name: 'posts',
  new_content_type: 'post',
  folder: 'articles',
  taxonomies: [{ // [Optional] Array of Objects, the taxonomies of the content type
    name: 'categories', // Name of the taxonomy in WordPress
    field: 'categories', // Name of the source field in WordPress
  }],
  schema_mapping: {
    // other fields...
    categories: 'content.categories', // Categories field mapping WP:Storyblok
  }
}
```



# MIGRATION USING A SCRIPT

## Blocks mapping (Import Gutenberg blocks as components in Storyblok)

- Install [REST API blocks plugin](#) in your WP site.

```
blocks_mapping: [  
  {  
    name: 'core/heading', // Block's name from Gutenberg  
    new_block_name: 'core-heading', // Component's name in Storyblok  
    schema_mapping: { // Same format as in Content-Types  
      'attrs.level': 'level',  
      'attrs.content': 'content'  
    }  
  }  
],
```

# MIGRATION USING A SCRIPT

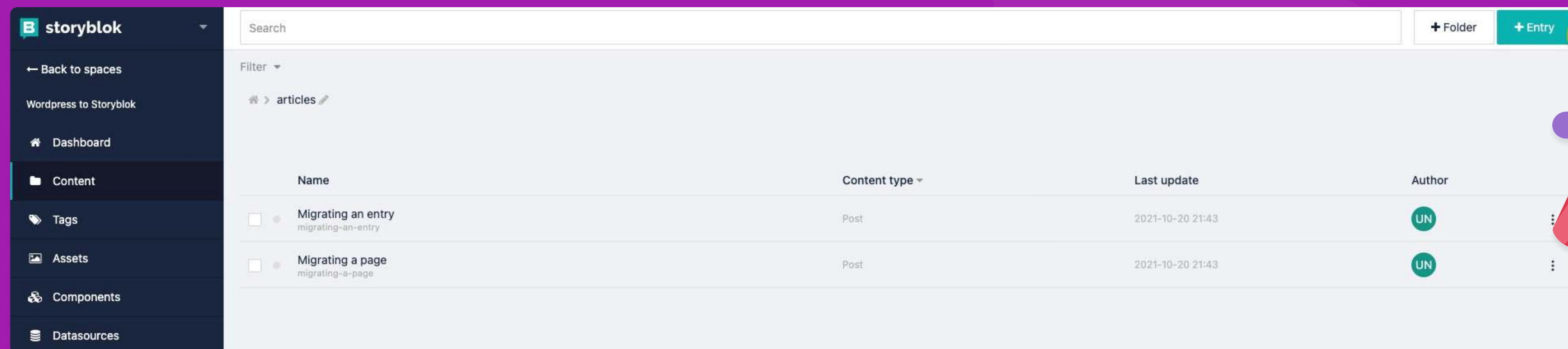
## Run the migration

DAY

1

- Run the script:

```
npm run migrate
```



The screenshot shows the Storyblok dashboard interface. On the left is a dark sidebar with navigation links: 'Back to spaces', 'Wordpress to Storyblok', 'Dashboard', 'Content' (highlighted), 'Tags', 'Assets', 'Components', and 'Datasources'. The main area has a search bar and a '+ Folder' button. Below is a filter dropdown set to 'articles'. A table displays two migrated items:

Name	Content type	Last update	Author
<input type="checkbox"/> Migrating an entry migrating-an-entry	Post	2021-10-20 21:43	UN
<input type="checkbox"/> Migrating a page migrating-a-page	Post	2021-10-20 21:43	UN

A red confetti icon is positioned to the right of the table.

DAY

1

- Detect when it makes sense to migrate from WP to a Headless setup.
- Create and understand a Storyblok space.
- Create Content Types with the same schema as in WP, and relationships.
- Migrate your WP content to a new Headless CMS, configuring a script.

DAY

2

- Create a Frontend project and connect it to our Storyblok space.
- Implement the components and render the migrated content.

# CREATE THE FRONTEND PROJECT

## STEPS for creating the project

- Create the frontend using the vue-nuxt-boilerplate template.
- Clone the repo:

```
git clone https://github.com/Your-User/repo-name.git
```

- Install dependencies:

```
cd ./repo-name && yarn
```



# CREATE THE FRONTEND PROJECT

## STEPS for connecting with Storyblok

- Create an `.env` file in the root of the project.
- Get the API token from the Storyblok space (Settings > API-Keys):

```
API_TOKEN='[API-Keys token]'
```

- Replace the `accessToken` at `nuxt.config.js` by:

```
process.env.API_TOKEN
```

- Run `yarn dev` and start creating the templates & components.

DAY

2

- Create a Frontend project and connect it to our Storyblok space.
- Implement the components and render the migrated content.

# IMPLEMENT TEMPLATES & COMPONENTS

## Creating templates

- Create a folder called **templates** in the component's folder to add there the **Content Types**:  
Add **Page.vue** (should be already there), **Post.vue**, **Category.vue**.
- Add the templates' import in the **components.js** (plugins folder):

```
/** Templates */  
import Page from '~/components/templates/Page.vue'  
import Post from '~/components/templates/Post.vue'  
import Category from '~/components/templates/Category.vue'
```

# IMPLEMENT TEMPLATES & COMPONENTS

## Creating templates

- Edit the `_vue` page (pages' folder) to have a dynamic component:

```
<component  
  v-if="story.content.component"  
  :key="story.content._uid"  
  :blok="story"  
  :is="story.content.component" />
```

\*This will render the corresponding Content Type (template) for each page.

# IMPLEMENT TEMPLATES & COMPONENTS

## Creating templates - Content Type: Page

- Create **AtomImage.vue** (will be reused in the other Content Types)  
To represent the **featured\_image** of the Page, coming from Storyblok, using the Image CDN.

```
// Template


export default { // Script
  props: { image: { type: Object, required: true }, size: { type: String, required: true } },
  methods: {
    transformImage(image, option) {
      if (!image) return ""; if (!option) return image;
      const path = (image.replace('https://a.storyblok.com', '')).replace('//a.storyblok.com', '');
      return `//img2.storyblok.com/${option}/filters:format(webp)${path}`;
    }
  }
}
```

# IMPLEMENT TEMPLATES & COMPONENTS

## Creating templates - Content Type: Page

- Define a dynamic component

Which will represent the blocks (nested components) defined in the body field of the page.

```
<component  
  v-for="blok in blok.content.body"  
  :key="blok._uid"  
  :blok="blok"  
  :is="blok.component"  
>
```



# IMPLEMENT TEMPLATES & COMPONENTS

## Creating templates - Content Type: Page

### ■ Resultant Page.vue

```
// Template
<section v-editable="blok">
  <atom-image :image="blok.content.featured_image" size="700x0" />
  <component
    v-for="blok in blok.content.body"
    :key="blok._uid"
    :blok="blok"
    :is="blok.component" />
</section>
```

# IMPLEMENT TEMPLATES & COMPONENTS

## Creating templates - Content Type: Post

- Create AtomRichText.vue  
To represent a rich-text field type.

```
// Template
<div v-if="richTextHTML" v-html="richTextHTML" class="prose"></div>

// Script
export default {
  props: { richText: { type: [Object, String], required: true } },
  computed: {
    richTextHTML() { return this.richText ? this.$storyapi.richTextResolver.render(this.richText) : '' }
  }
}
```

# IMPLEMENT TEMPLATES & COMPONENTS

## Creating templates - Content Type: Post

- Reuse the AtomImage.vue for the featured\_image field.
- Reuse the AtomRichText.vue for the content field.
- Resolve the relation with the Category Content Type and Post:

```
// _.vue page
return context.app.$storyapi.get(`cdn/stories/${fullSlug}`, {
  version,
  resolve_relations: 'post.categories' // Resolve the relation
})
```

# IMPLEMENT TEMPLATES & COMPONENTS

## Creating templates - Content Type: Post

### ■ Resultant Post.vue

```
// Template
<article v-editable="blok">
  <atom-rich-text :rich-text="blok.content.excerpt" />
  <atom-image :image="blok.content.featured_image" size="700x0" />
  <atom-rich-text :rich-text="blok.content.content" />
  <p>
    Posted in
    <nuxt-link v-for="category in blok.content.categories" :key="`category-${category.id}`" :to="`/${category.full_slug}`">
      {{ category.name }}
    </nuxt-link>
  </p>
</article>
```

# IMPLEMENT TEMPLATES & COMPONENTS

## Creating templates - Content Type: Category

### ■ Resultant Category.vue

```
<template>
  <section v-editable="blok">
    <h1>Category {{ blok.name }}</h1>
    <p>{{ blok.content.description }}</p>
  </section>
</template>
```

# IMPLEMENT TEMPLATES & COMPONENTS

## Creating nested components

- For nested components coming from WordPress, use the prefix Core.
- Add the defined nested components in the components folder.  
For the demo: CoreImage.vue, CoreHeading.vue, CoreParagraph.vue and CoreQuote.vue.
- Import them in the component.js plugin:

```
/** Components */  
import CoreHeading from '~/components/CoreHeading.vue'  
  
Vue.component('core-heading', CoreHeading)
```



DAY

2

- Create a Frontend project and connect it to our Storyblok space.
- Implement the components and render the migrated content.