The Composable Hero Landing Page

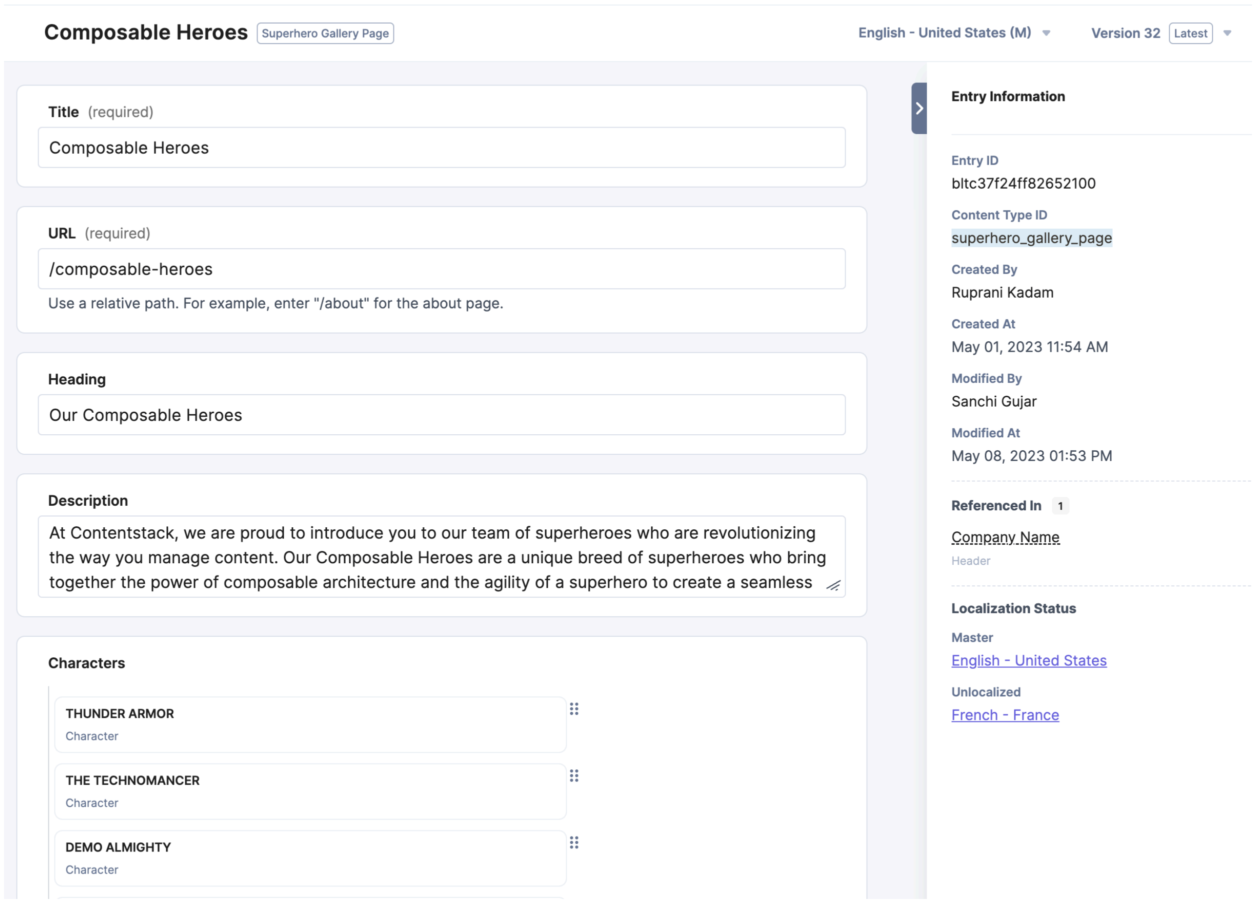
Welcome to this tutorial on creating a superhero gallery page using reference fields in NextJS! In this module, we'll walk you through the steps of creating a new content type called "Superhero Gallery Page" and adding heroes to it manually.

We'll start by showing you how to utilize reference fields to display hero content on the landing page with their images and names. This technique is commonly used in product promotions and article index pages, so it's a valuable skill to learn.

Next, we'll guide you through the process of creating a new NextJS component and query to retrieve the Landing Page Entry and Reference field Hero Entries. Our NextJS component will loop through all the heroes in the reference field and display their titles, images, and links to their respective detail pages.

Finally, we'll demonstrate how to create a new Landing Page Entry and publish it to the Development environment to see the results of our work.

By the end of this tutorial, you'll have gained valuable knowledge on how to utilize reference fields in NextJS to create reusable content and enhance the user experience on your website. Let's get started!



Composable Heroes Superhero Gallery Page

English - United States (M) Version 32 Latest

Title (required)

Composable Heroes

URL (required)

/composable-heroes

Use a relative path. For example, enter "/about" for the about page.

Heading

Our Composable Heroes

Description

At Contentstack, we are proud to introduce you to our team of superheroes who are revolutionizing the way you manage content. Our Composable Heroes are a unique breed of superheroes who bring together the power of composable architecture and the agility of a superhero to create a seamless

Characters

THUNDER ARMOR
Character

THE TECHNOMANCER
Character

DEMO ALMIGHTY
Character

Entry Information

Entry ID
bltc37f24ff82652100

Content Type ID
superhero_gallery_page

Created By
Ruprani Kadam

Created At
May 01, 2023 11:54 AM

Modified By
Sanchi Gujar

Modified At
May 08, 2023 01:53 PM

Referenced In 1

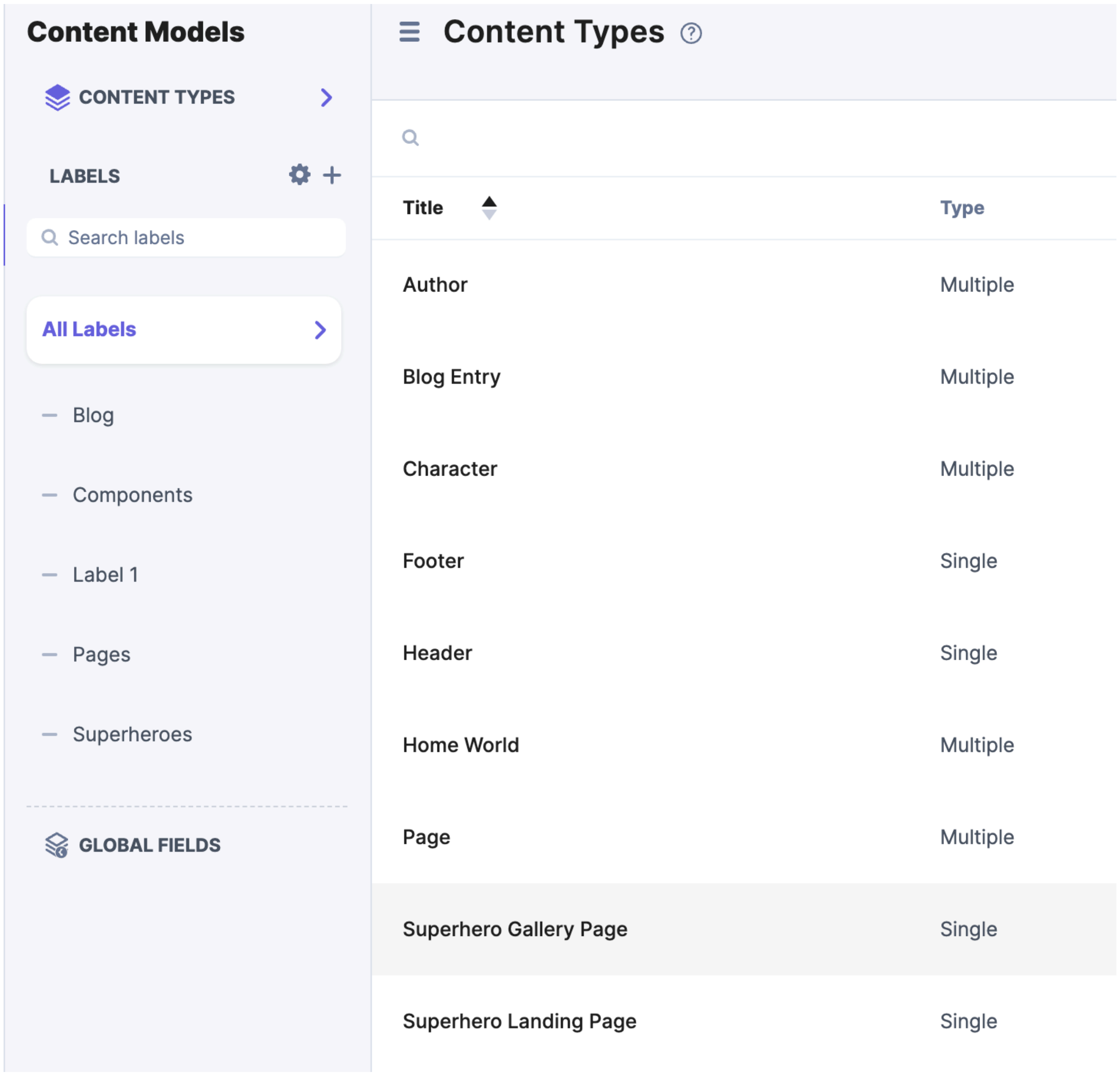
Company Name
Header

Localization Status

Master
English - United States

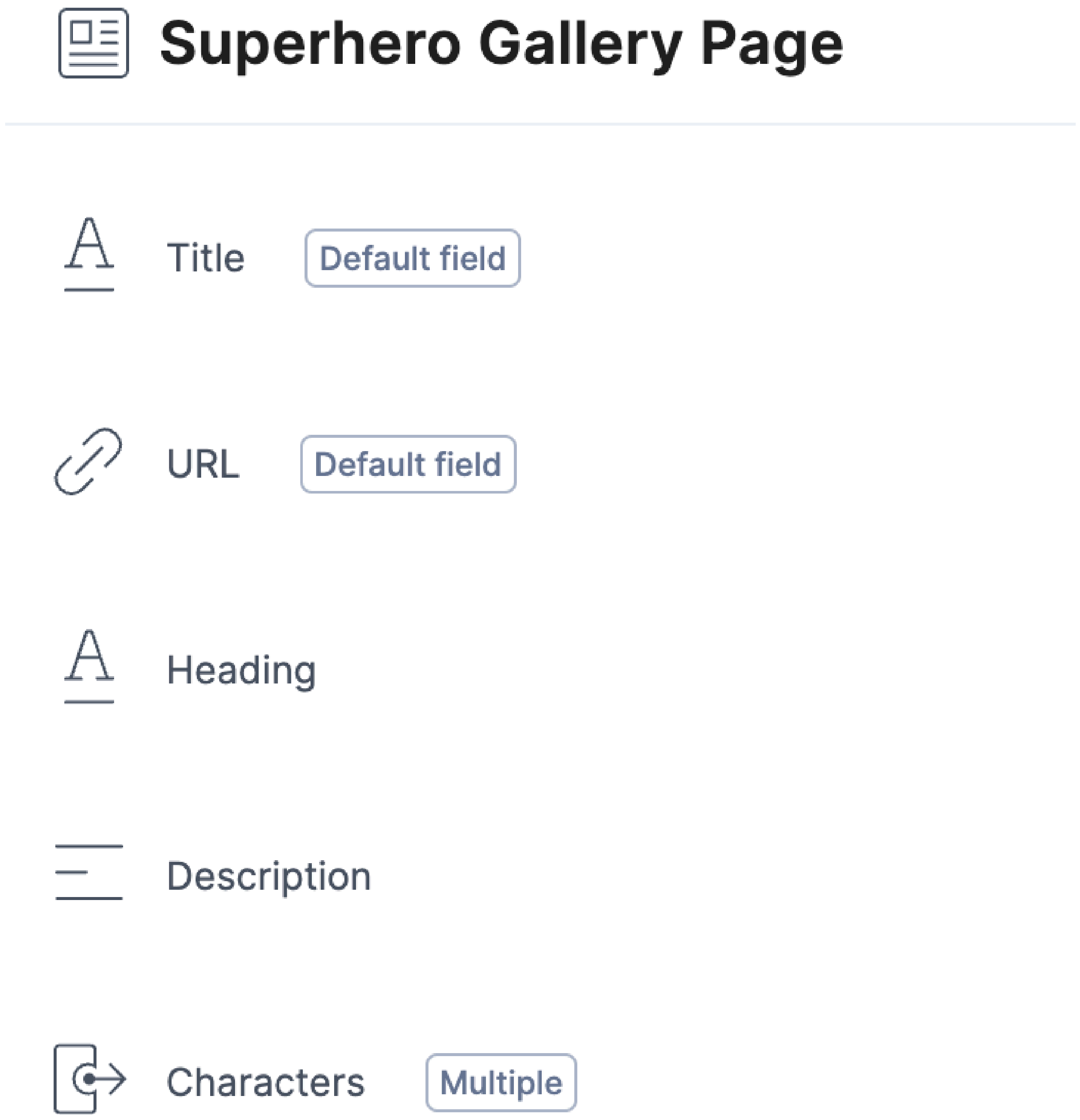
Unlocalized
French - France

Hero Gallery Page Content Type



Title	Type
Author	Multiple
Blog Entry	Multiple
Character	Multiple
Footer	Single
Header	Single
Home World	Multiple
Page	Multiple
Superhero Gallery Page	Single
Superhero Landing Page	Single

Open the Superhero Gallery Page and examine the properties of each field



Superhero Gallery Page

Title Default field

URL Default field

Heading

Description

Characters Multiple

Review the following fields below:

- Title - Default Title field
- URL - /gallery/:title
- Heading - Text field
- Description - Multi-line text
- Characters - Reference field to the Character Content Type

Part 2: Writing the code to display the list of heroes

Let's list all the heroes in the reference field on the Gallery page. Users can manually create a list of heroes and our code will loop over all the heroes in the reference field and render the name, image, and link.

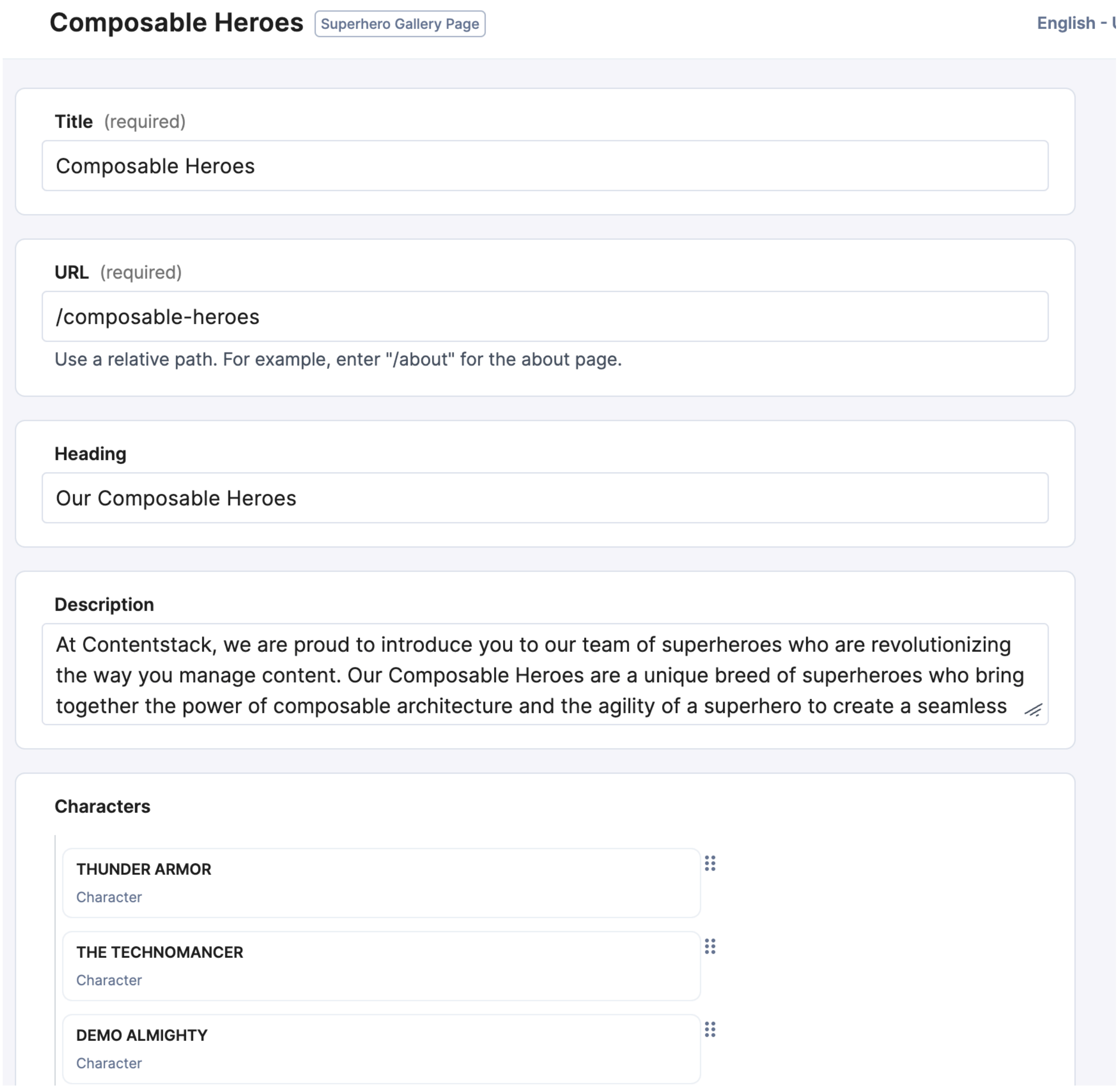
Rendering a list of Characters from a Reference Field

1. Create a NextJS Component for the Gallery page and loop through all the heroes in the reference field, rendering the title, image and a link to the detail page. This involves 2 steps, first writing the query function and then rendering the results.

2. Querying Contentstack for the Characters in the Reference Field
In the /helpers/index.js file, review the function to get a list of heroes from the reference field

3. Review the code for the /composable-heroes/index.tsx file.

4. Review the Composable Heroes Gallery Entry


Composable Heroes Superhero Gallery Page English - U

Title (required)

Composable Heroes

URL (required)

/composable-heroes

Use a relative path. For example, enter "/about" for the about page.

Heading

Our Composable Heroes

Description

At Contentstack, we are proud to introduce you to our team of superheroes who are revolutionizing the way you manage content. Our Composable Heroes are a unique breed of superheroes who bring together the power of composable architecture and the agility of a superhero to create a seamless

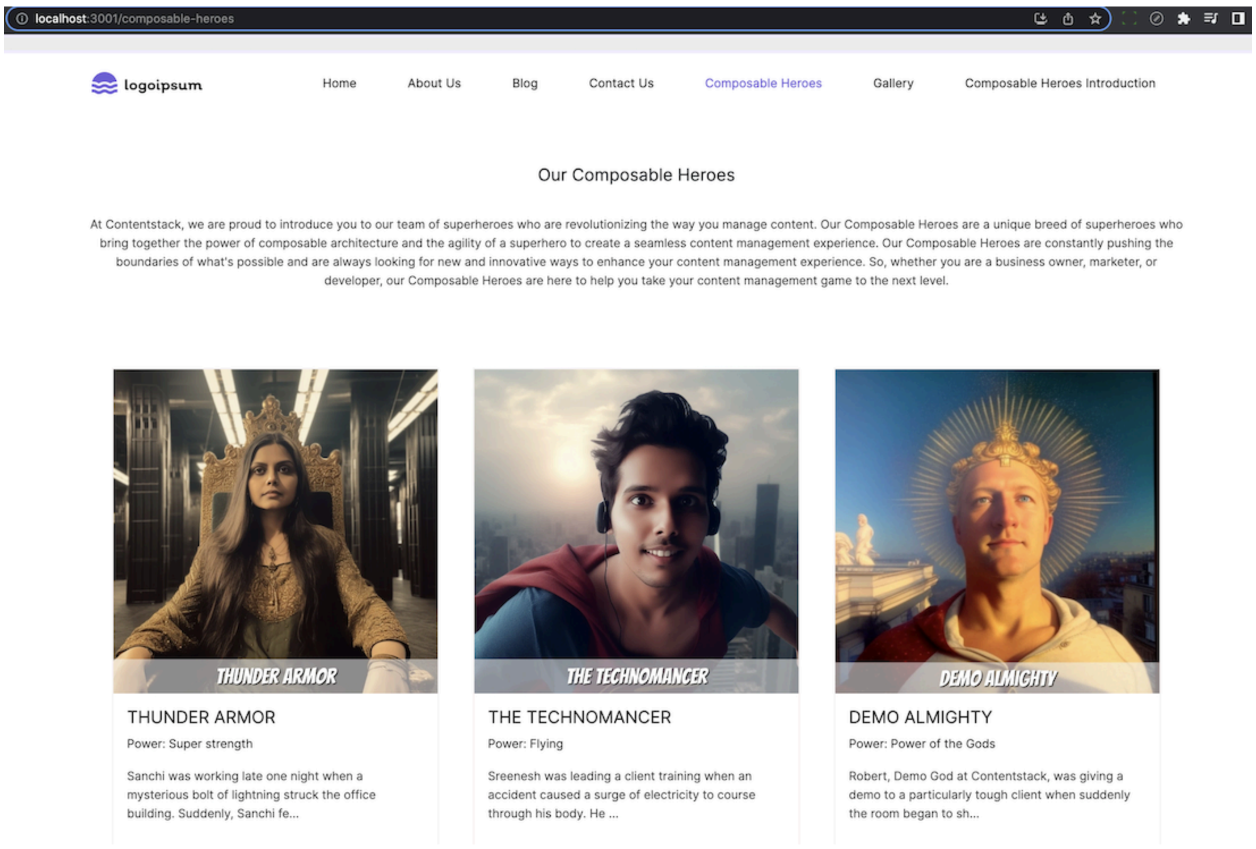
Characters

THUNDER ARMOR
Character

THE TECHNOMANCER
Character

DEMO ALMIGHTY
Character

5. From the NextJS application, open the Composable Heroes Gallery Page


localhost:3001/composable-heroes

Our Composable Heroes

At Contentstack, we are proud to introduce you to our team of superheroes who are revolutionizing the way you manage content. Our Composable Heroes are a unique breed of superheroes who bring together the power of composable architecture and the agility of a superhero to create a seamless content management experience. Our Composable Heroes are constantly pushing the boundaries of what's possible and are always looking for new and innovative ways to enhance your content management experience. So, whether you are a business owner, marketer, or developer, our Composable Heroes are here to help you take your content management game to the next level.

THUNDER ARMOR

Power: Super strength

Sanchi was working late one night when a mysterious bolt of lightning struck the office building. Suddenly, Sanchi fe...

THE TECHNOMANCER

Power: Flying

Sreenesh was leading a client training when an accident caused a surge of electricity to course through his body. He ...

DEMO ALMIGHTY

Power: Power of the Gods

Robert, Demo God at Contentstack, was giving a demo to a particularly tough client when suddenly the room began to sh...

2. Introduction to using the Query functions of the Content Delivery API

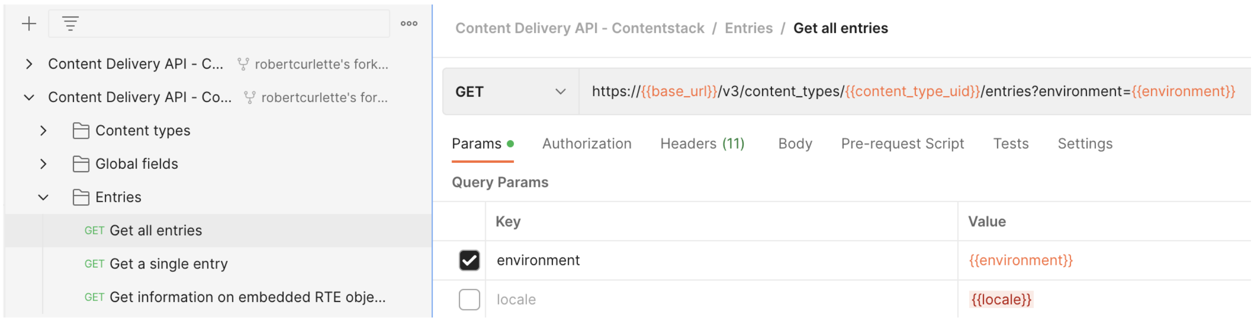
Contentstack provides certain queries that you can use to fetch filtered results. You can use queries for Entries and Assets API requests. Please review the Content Delivery Query API documentation and examine the RegEx, Limit and Only functions.

For the first part of the exercise, we can use a simple query to find all the entries with the Content Type of Character.

Postman

1. Open Postman and select the Entries/Get Entries call
Update the parameters / variables so that:

- content_type_uid = character
- environment = development


Content Delivery API - C... robertcurlette's fork...
Content Delivery API - Co... robertcurlette's for...
Content types
Global fields
Entries
GET Get all entries
GET Get a single entry
GET Get information on embedded RTE obje...
Content Delivery API - Contentstack / Entries / Get all entries
GET
Params
Authorization
Headers (11)
Body
Pre-request Script
Tests
Settings
Query Params

```
https://{{base_url}}/v3/content_types/{{content_type_uid}}/entries?environment={{environment}}
```

	Key	Value
✓	environment	{{environment}}
☐	locale	{{locale}}

You should now have a list of the characters published to the development environment. In the next section, we will use the JavaScript SDK to get the same list and pass it to the NextJS App to render.

3. Query using Javascript SDK

In our web application we will mostly use the Javascript Content Delivery SDK. Please have a look at the documentation for the query methods.

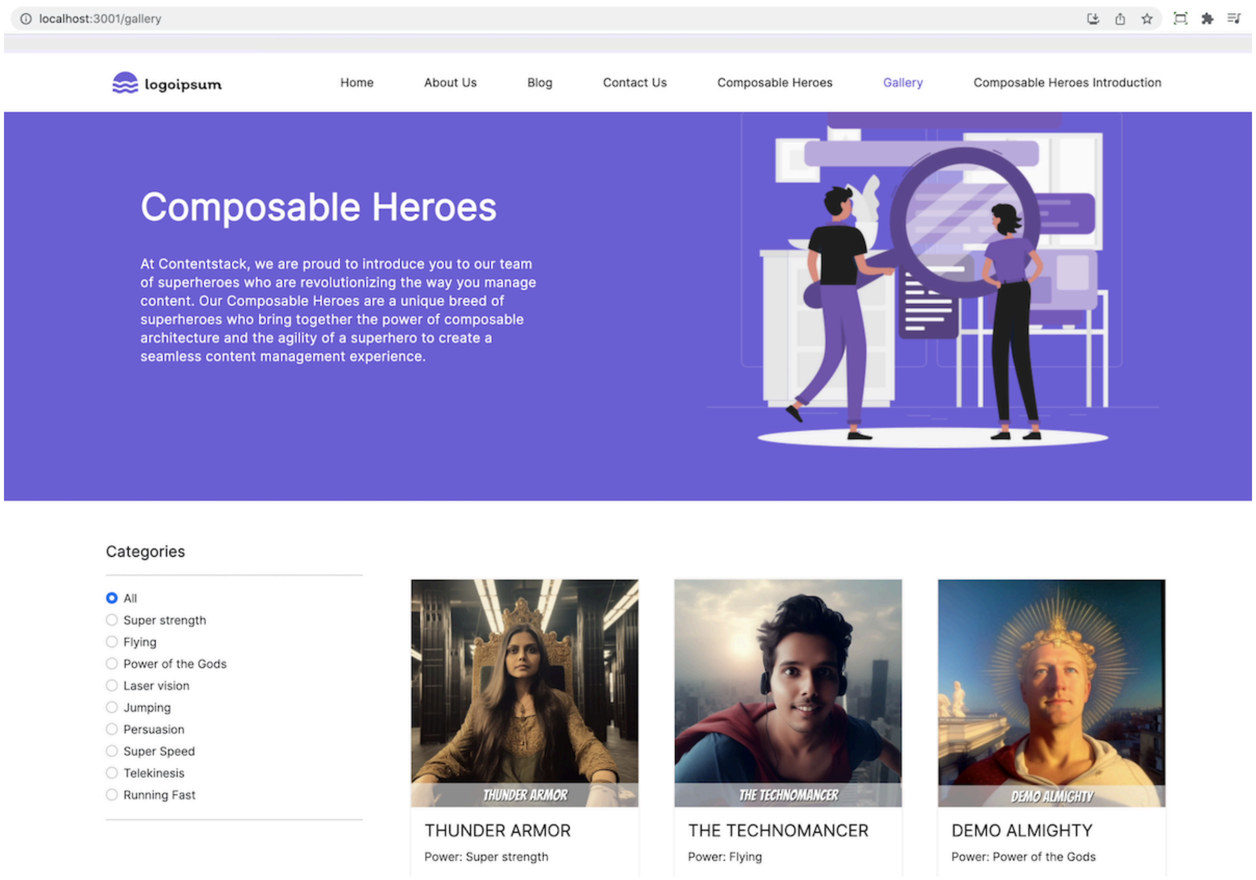
Other SDKs can be found on the SDKs documentation page

```
import Contentstack from 'contentstack';
const Stack = Contentstack.Stack({"api_key": "api_key", "delivery_token": "delivery_token",
"environment": "environment"});
const result = await Stack.ContentType('character').Query().toJSON().find();
```

More information about this is found in the Javascript SDK Docs

Composable Hero Landing Page

The Composable Hero Landing Page uses Modular Blocks to provide a flexible structure for authors to create content and add as many content blocks to the page as they wish and also to re-order the blocks.



Composable Heroes

At Contentstack, we are proud to introduce you to our team of superheroes who are revolutionizing the way you manage content. Our Composable Heroes are a unique breed of superheroes who bring together the power of composable architecture and the agility of a superhero to create a seamless content management experience.

Categories

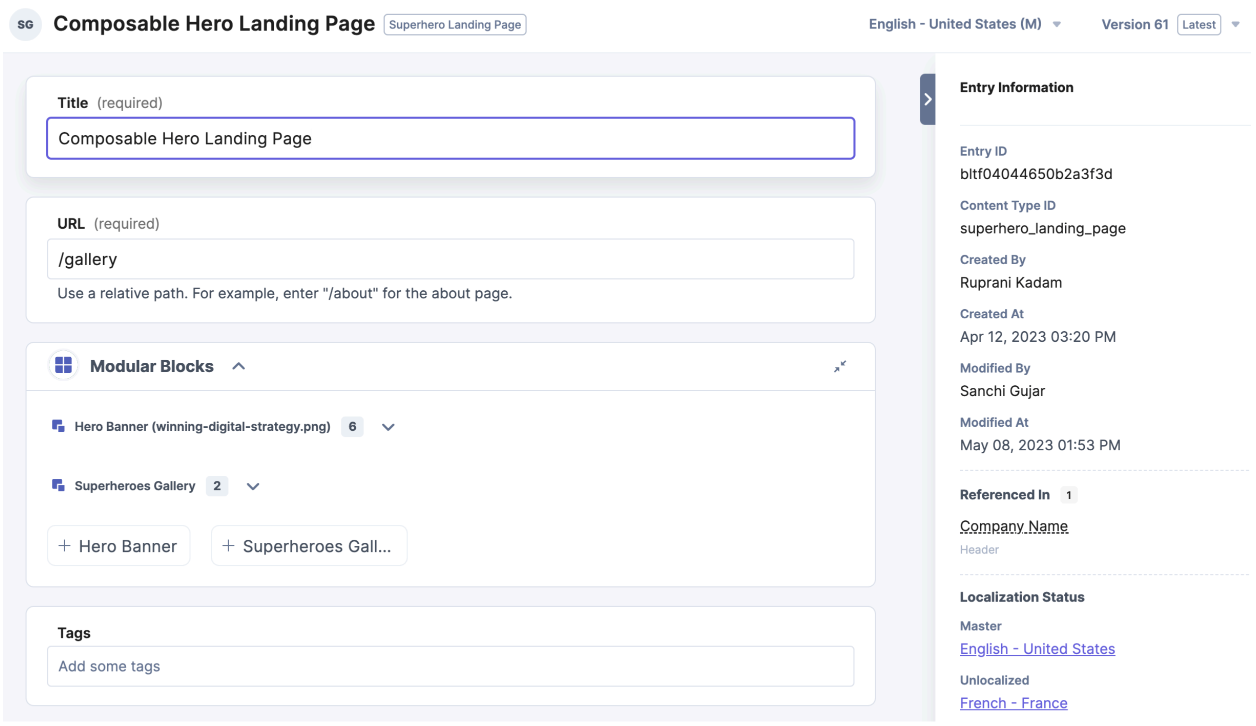
- [✓ All]
- Super strength
- Flying
- Power of the Gods
- Laser vision
- Jumping
- Persuasion
- Super Speed
- Telekinesis
- Running Fast

THUNDER ARMOR
Power: Super strength

THE TECHNOMANCER
Power: Flying

DEMO ALMIGHTY
Power: Power of the Gods

Open the Superhero Landing Page Entry and review the content



SG Composable Hero Landing Page Superhero Landing Page

English - United States (M) Version 61 Latest

Title (required)

Composable Hero Landing Page

URL (required)

/gallery

Use a relative path. For example, enter "/about" for the about page.

Modular Blocks

Hero Banner (winning-digital-strategy.png) 6

Superheroes Gallery 2

+ Hero Banner + Superheroes Gall...

Tags

Add some tags

Entry Information

Entry ID
bltf04044650b2a3f3d

Content Type ID
superhero_landing_page

Created By
Ruprani Kadam

Created At
Apr 12, 2023 03:20 PM

Modified By
Sanchi Gujar

Modified At
May 08, 2023 01:53 PM

Referenced In 1

Company Name
Header

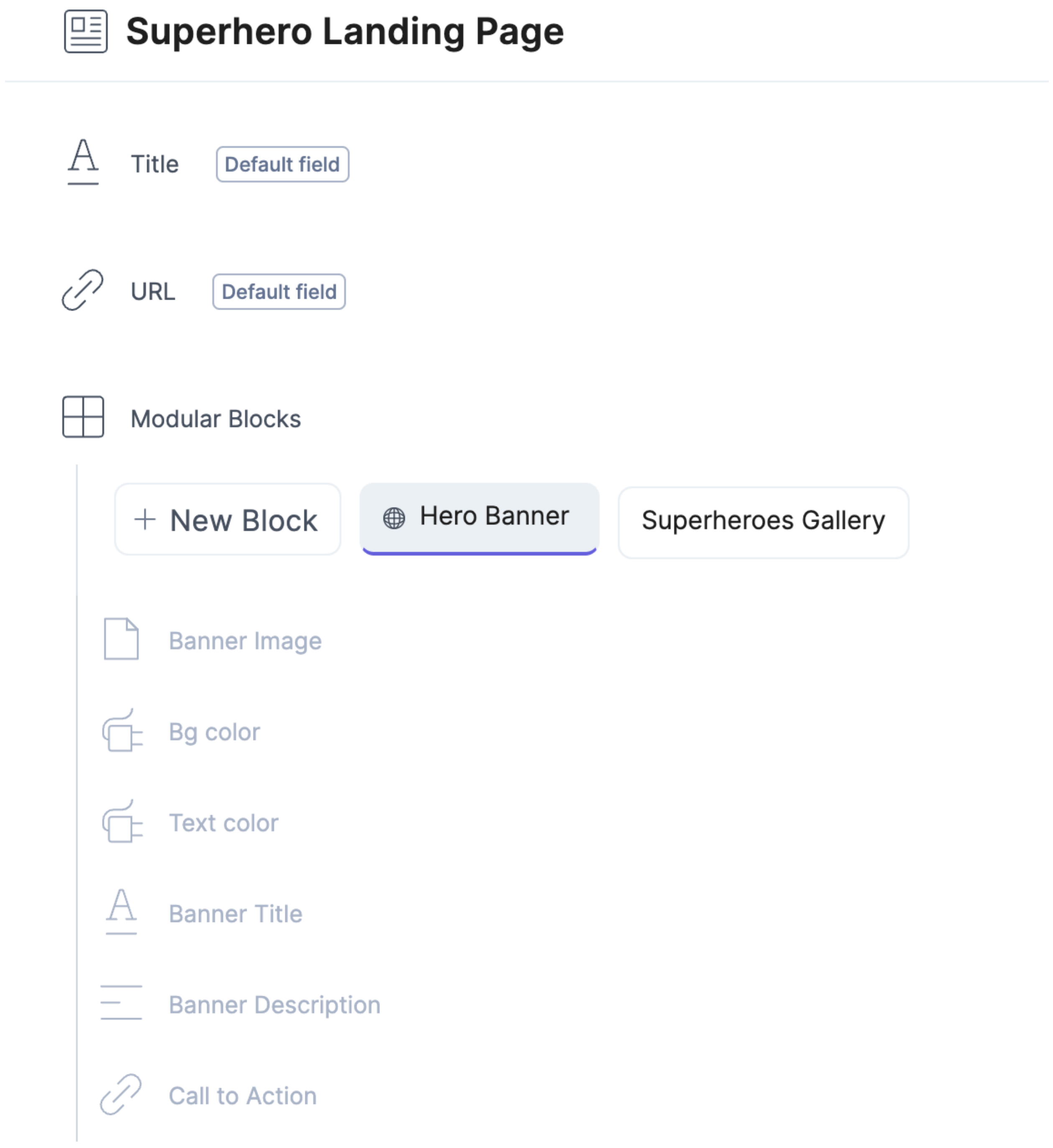
Localization Status

Master
English - United States

Unlocalized
French - France

Open the Content Type Definition and examine how the Modular Blocks field is used.

Notice the fields are greyed out - that is because the Modular Block fields are defined using Global Fields. This is a best practice.



Superhero Landing Page

Title Default field

URL Default field

Modular Blocks

+ New Block | Hero Banner | Superheroes Gallery

Banner Image

Bg color

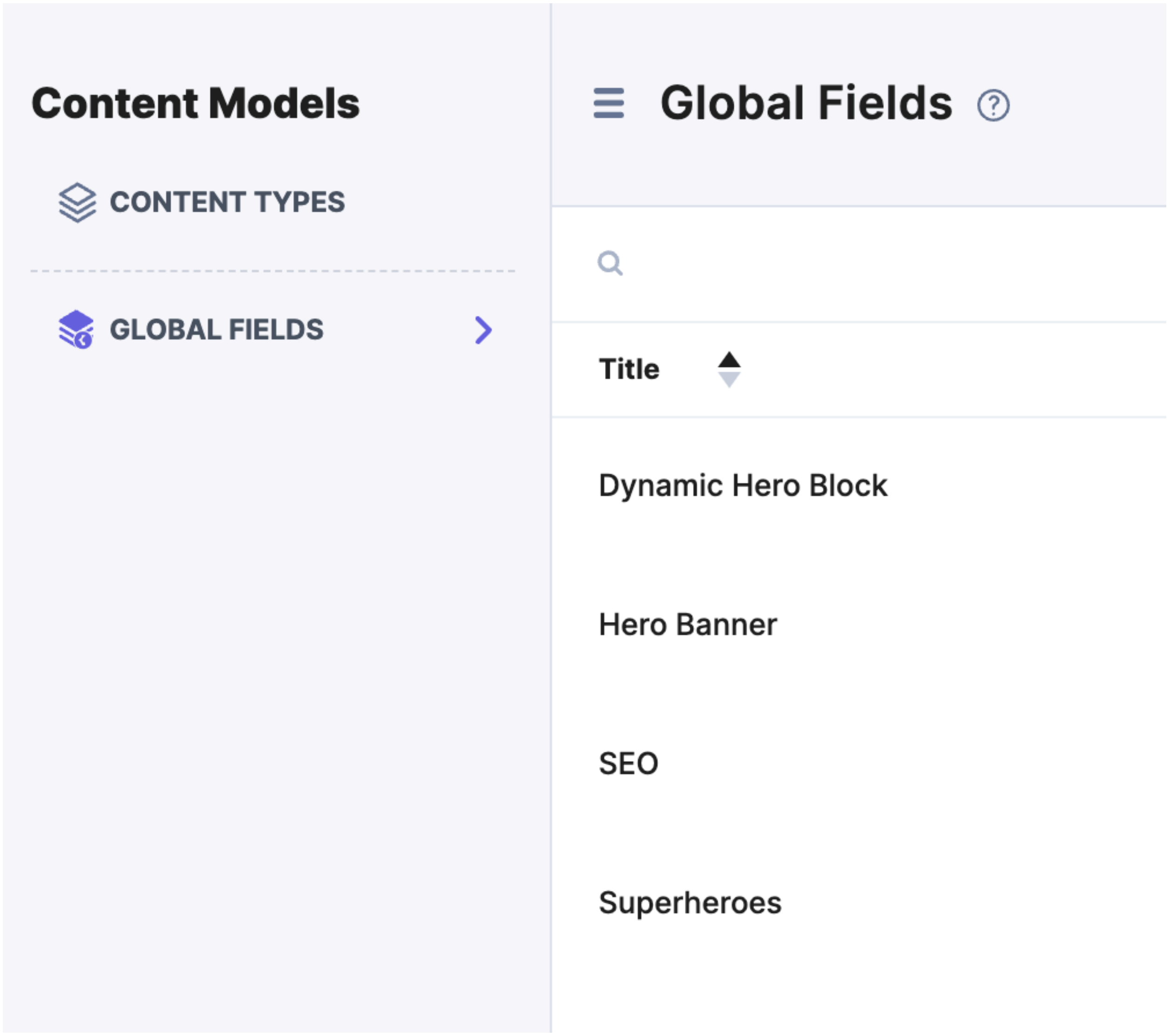
Text color

Banner Title

Banner Description

Call to Action

Global Field Definitions



Dynamic Hero Block

Hero Banner

SEO

Superheroes

Examine the Code used for the Hero Landing Page

The code for the Gallery Entry (Page) loops through all the Modular Blocks on the page and renders each one with its' own NextJS Component.

```
import React from 'react';
import { getComposableHeroGallery, getSuperheroGalleryRes } from '../helper';
import { PostPage, Context } from "../typescript/pages";
import { Posts } from '../typescript/layout';
import Gallery from '../components/gallery'
import HeroBanner from '../components/hero-banner';
import { Banner } from '../typescript/component';

const renderTemplateSection = (switchData: any[], modules: {
    hero_banner: Banner;
    super_heroes_gallery: {
        description: any; heroes: [], heading: string | undefined
};

}, index: any) => {
    const {
        super_heroes_gallery: superHeroGallery
    } = modules

    switch (switchData[0]) {
        case 'hero_banner':
            return (
                <HeroBanner banner={modules.hero_banner} key={`herobanner${index.toString()}`}} />
            )
        case 'super_heroes_gallery':
            return (
                <Gallery data={superHeroGallery?.heroes}
                    heading={superHeroGallery?.heading}
                    description={superHeroGallery?.description}
                    showFilter showDescription={false}
                    key={`gallery${index.toString()}`}
                />
            )

        default: return null
    }
}
```

Each block is rendered with its own NextJS Component

- Gallery Block

```
import React, { useState } from "react";
import parse from 'html-react-parser';

const GalleryReact = ({ data, heading, description, showFilter, showDescription }: { data: any, heading: any, showFilter: boolean, showDescription: boolean, description: any })
    const [items, setItems] = useState(data);
    const [active, setActive] = useState(false);
    const powersList: string[] = []

    const filterItem = (categItem: string) => {
        const updateItems = data?.filter((curElem: { powers: string[]; }) => {
            return curElem.powers[0] === categItem[0];
        });
        setItems(updateItems);
        setActive(true);
    };

    return (
        <>
            {
                heading ?
                    <>
                        <nav className="navbar mt-2">
                            <span className="navbar-brand mb-0 m-auto h1 text-center">
                                {heading}
                            </span>
                        </nav><br />
                    </>
                    : ''
            }
            {
                description ?
                    <div className='container'>
                        <p className=" text-center">
                            {description}
                        </p>
                    </div>
                    : ''
            }
```

- Hero Banner Block

```
import React from 'react';
import Link from 'next/link';
import { Image, Action } from "../typescript/action";

type AdditionalParam = {
  banner_title: string;
  banner_description: string;
}

type Banner = {
  bg_color: string;
  text_color: string;
  banner_title: string;
  banner_description: string;
  call_to_action: Action;
  banner_image: Image;
  $: AdditionalParam;
}

type BannerProps = {
  banner: Banner;
}

export default function HeroBanner(props: BannerProps) {

  const banner = props.banner;

  return (
    <div
      className='hero-banner'
      style={{
        background: banner?.bg_color ? banner.bg_color : '',
      }}
    >
      <div
        className='home-content'
        style={{
          color: banner?.text_color ? banner.text_color : '#000',
        }}
      >
        {banner.banner_title && (
          <h1 className='hero-title' {...banner.$?.banner_title as {}}>
            {banner.banner_title}
```

By adopting this approach, we can isolate the HTML/Code for each block within a single NextJS Component. This simplifies maintenance and enables easy extension of the page in the future by adding a Modular Block and NextJS Component for additional content block types.

With this method, you can modify the individual content blocks without having to make changes to the entire page structure. This approach makes the codebase more organized and easier to manage, especially when dealing with complex web applications.

By leveraging this technique, you can increase the scalability and modularity of your website, allowing you to add new features and functionalities with ease.