2022

The Ultimate Contentful Guide

Here's everything you need to know to get started with Contentful and make your transition to this headless CMS a success.

IN PARTNERSHIP WITH contentful

HELLOBAYTREE.COM

baytree

Introduction

Hi there!

We're Baytree; a development studio that creates, manages and maintains next generation products for our clients. Fundamental to all our work is a belief in human centric design underpinned by strategic business thinking and world class engineering. We've been doing this for over 15 years. We've worked with Governments, SME's, multi-nationals and venture capital backed startups and have a huge amount of experience and learning to share from our diverse client roster.

We've put together this guide to help you learn some key points, tips and tricks that'll make your migration to Contentful a success.

We hope you gain useful insight from our substantial experience in implementing mission-critical Contentful systems, including one of the most highly trafficked news websites in the world.

Comparison with other CMSes

If you're coming from WordPress (or other legacy CMS), it's important to understand the pros and cons of headless CMSes before beginning your technical planning.

Reduced operational/maintenance burden

One of the first reasons we're such fans of Contentful is because it significantly reduces required maintenance as compared to that of traditional CMSes. Being cloud hosted; Contentful is responsible for managing, securing and upgrading the CMS API endpoint and admin interface.

Given the multitude of security issues and regular patching requirements of legacy CMS systems (and their plugins), for many developers this means once the site is launched you will have little to no maintenance to do with Contentful.

Furthermore, you can also say goodbye to those hughley annoying 'system upgrades' that traditional CMSs often introduce; requiring you to update your themes, logic and plugins with no option to remain on a previous version that was working just fine. Everything this happens, you have to take another close look at your platform architecture and security.

By moving to Contentful, this pain and work burden is removed. This has been an enormous cost saving for our clients - often freeing up multiple FTE developers and operational staff for other more important tasks.

KEY TAKEAWAY

Expect significantly lower development and operational expenditure with a headless CMS vs legacy CMS. And here's some good news — the bigger your platform is, the bigger benefits you can expect when you switch to a headless CMS.

CMS modifications — be aware that...

In general, it is more difficult to modify the 'admin interface' of Contentful than that of a legacy CMS. This is because Contentful hosts a standardised version for all customers; unlike WordPress where everyone has their own install. This allows significant benefits; but can be a hindrance if you are used to being able to make significant changes to the interface's look and feel.

Many customisations can still be performed via 'Contentful apps', small (or large) plugins you host yourself, and, webhooks and Contentful APIs. However, this doesn't tend to go as far as wholesale changes to the UI interface itself which may be possible with other legacy CMSes. In our experience, this isn't required often, and indeed can be a positive in so much that the Contentful admin interfaces tend to be very similar — reducing staff training time and cost as you roll the product out throughout your organisation.

KEY TAKEAWAY

Think carefully about the level of customisation you need with your admin interface. Speak with key stakeholders to discover if the out of the box experience of the Contentful admin UI works before starting development.

Decoupling of frontend from backend

Covered in more detail in the architecture section of this guide, one of the key benefits of Contentful and a headless CMS is being able to decouple the front end from the back end.

This offers many benefits. Typically, in WordPress, you'd have to use PHP based templating and business logic, which may not be a stack your development or operation team has much fluency in. With Contentful, you can use the technology stack you are most familiar with and have most experience with.

This makes it a lot easier to integrate with other systems (booking engines, CRMs, user authentication) as you can reuse a lot of your existing code and the libraries you use to interface with those systems.. There's no need, to write specialised code for your CMS-driven website, which, in our experience, leads to a website's UX being compromised and key business logic being moved to other subdomains/web apps. With a headless CMS such as Contentful, this becomes far less of a concern.

This does require a slight change in mindset for your development tea, who may be used to querying databases directly. With a headless CMS, all your data access will be performed via HTTP REST or GraphQL calls, which tends to force a more rigorous approach in accessing key data.

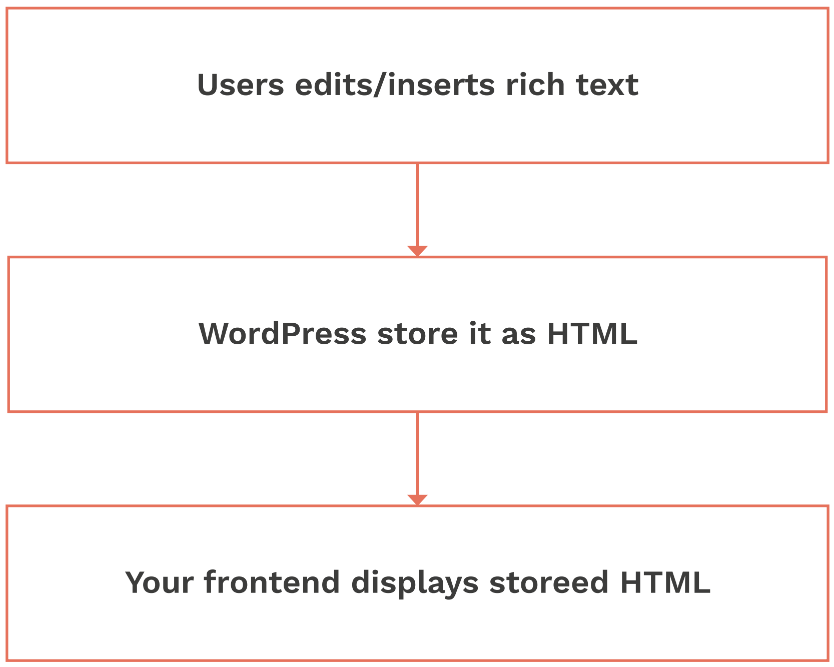
KEY TAKEAWAY

With a headless CMS you can expect to be able to reuse much more of your teams' previous experience and code.

Designed for multiple platforms (not just web!)

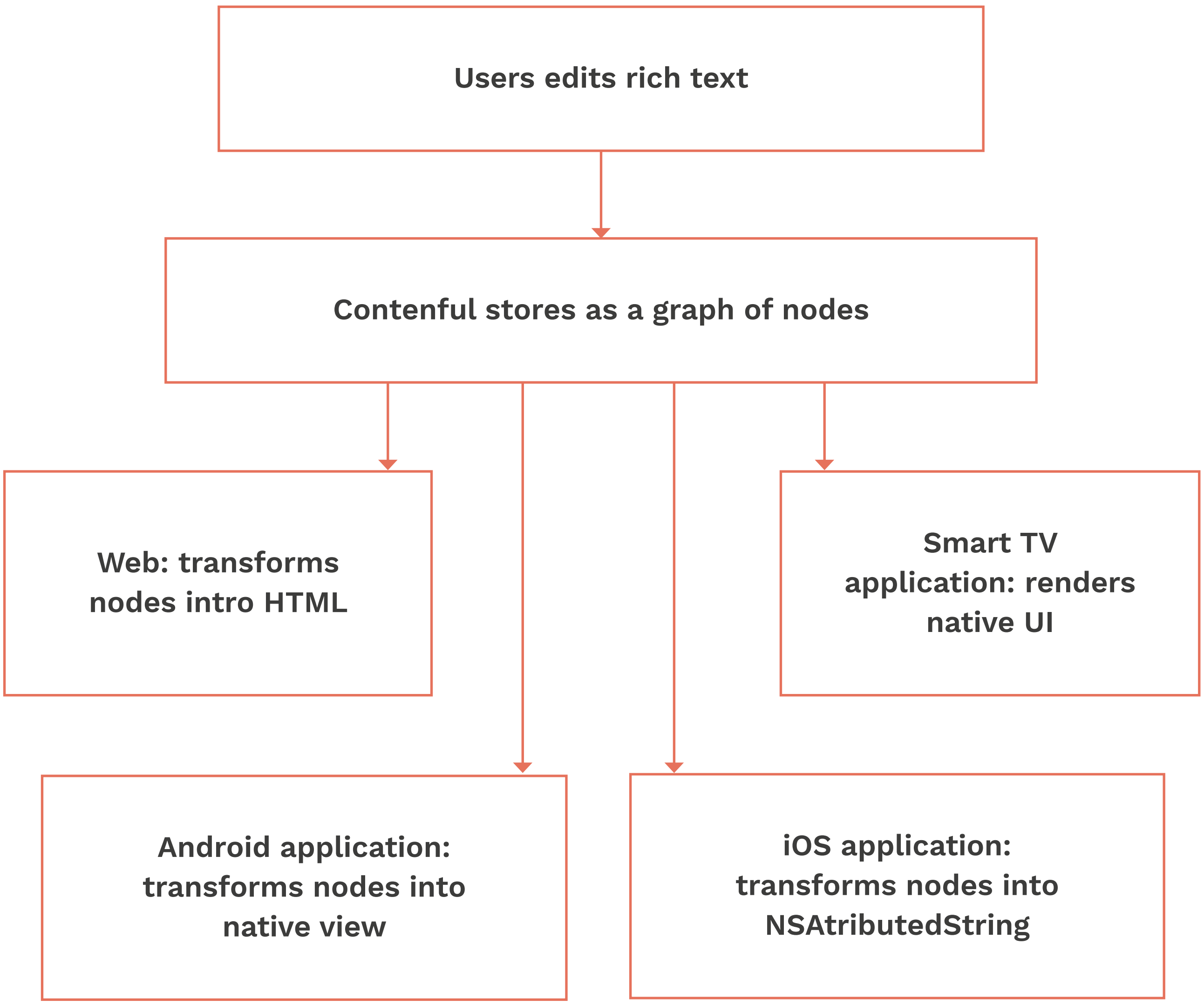
Given its dominance for the last 20+ years, most legacy CMSes are heavily web-oriented and assume that the content you input will be displayed on the web. This was once a fair assumption, but over the past decade it has become increasingly incorrect and a hidernace.

A simple example is how WordPress deals with rich text editing:


Users edits/inserts rich text
WordPress store it as HTML
Your frontend displays storeed HTML

This approach works great for websites - simple and effective.
But typically, legacy CMSs using this approach require your mobile applications to be littered with webviews to display rich text (as the CMS assumes it can render HTML), which significantly compromises performance and user experience of your application.

Contentful works differently, using this approach:


Users edits rich text
Contenful stores as a graph of nodes
Web: transforms nodes intro HTML
Smart TV application: renders native UI
Android application: transforms nodes into native view
iOS application: transforms nodes into NSAtributedString

This process has major benefits in a multiplatform world. Instead of assuming the client will be able to render HTML, it stores the data in an intermediate node format. The client then retrieves this data, and transforms it into the right format for the platform it is on.

While this sounds more complicated, in reality, there are many well tested and easy to use libraries that perform this transformation in often what is one line of code e.g.

Web/HTML: https://github.com/contentful/rich-text/tree/master/packages/rich-text-html-renderer

React/React Native: https://github.com/contentful/rich-text/tree/master/packages/rich-text-react-renderer

iOS: https://github.com/contentful/rich-text-renderer.swift

Android: https://github.com/contentful/rich-text-renderer-java

Contentful gives you an enormous amount of flexibility and helps future proof your content. And most importantly, it's easy to perform!

KEY TAKEAWAYS

Contentful offers far more flexibility in how it renders content, which is essential when you're deploying code across more than one platform. You can leverage tried and tested and high performing libraries that are available for handling the transform to your clients' environment.

Architecture

The key to a good Contentful build is choosing the right architecture from the start. Here are top areas of consideration.

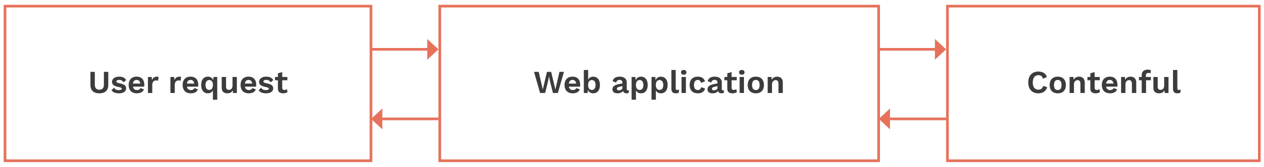
Choose how you will render pages

As Contentful is a headless CMS, it gives you significantly more flexibility in how you render your pages.

With a legacy CMS, you'd use server side rendering and your CMS templating engine to build the pages. With Contentful, you have more option:

Server side generation

In this approach, you have a web server running an application that renders pages. This application takes queries from your users, calls Contentful (via REST or GraphQL), and renders the pages, and serves them back to your users.



For most use cases this will be the preferred starting point. Why? Our main considerations are:

Page load speed: Generally, this allows for the fastest possible page load speed, as the server compiles the page and serves to one user in one go, without requiring the JavaScript VM on the client to initialise and request content. This has allowed us to serve pages from Contentful in less than 50ms. For SEO and user experience, fast page loads are critical.

Crawlability: As the page is provided 'complete', search engines can crawl the content easily.

Flexibility: This approach allows you to do complex transforms and joining of data — including other third party data sources. These tend to get extremely complicated to do securely on the client side.

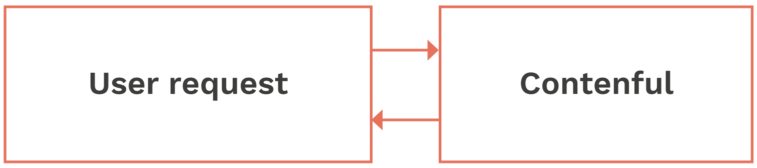
Reduction of client side code complexity: This may or may not be a benefit depending on your use case but, in general server side pages are simpler to test and automate than client side state.

If these are less important to you (for example, a private intranet site), this may not be as essential.

Achieving this design is simple - you can leverage your previous web framework (ASP.NET, Django, Ruby on Rails, etc), but instead of calling data from a database, you call it via GraphQL to Contentful.

Pure client side rendering

This is a newer approach made possible by the way headless CMSes are architected. Here your client side code calls Contentful directly, with no web server in between.



This has significant benefits in itself:

One application: You only have one set of client side code to worry about, with no server side server to run.

Easy deployment: As your application is essentially a static collection of files, it can be hosted easily on static hosts, such as S3, Azure Blob, GCP Storage or Netifly.

More UI flexibility: If you require a lot of UI interaction that is not well suited to server side rendering, this may be an important consideration, However, please bear in mind the page load and crawlability drawbacks of this approach. We have been involved in many projects which started with this architecture but had to revert to server side rendering or the hybrid approach outlined below as the hit to SEO scores was too great.

Hybrid

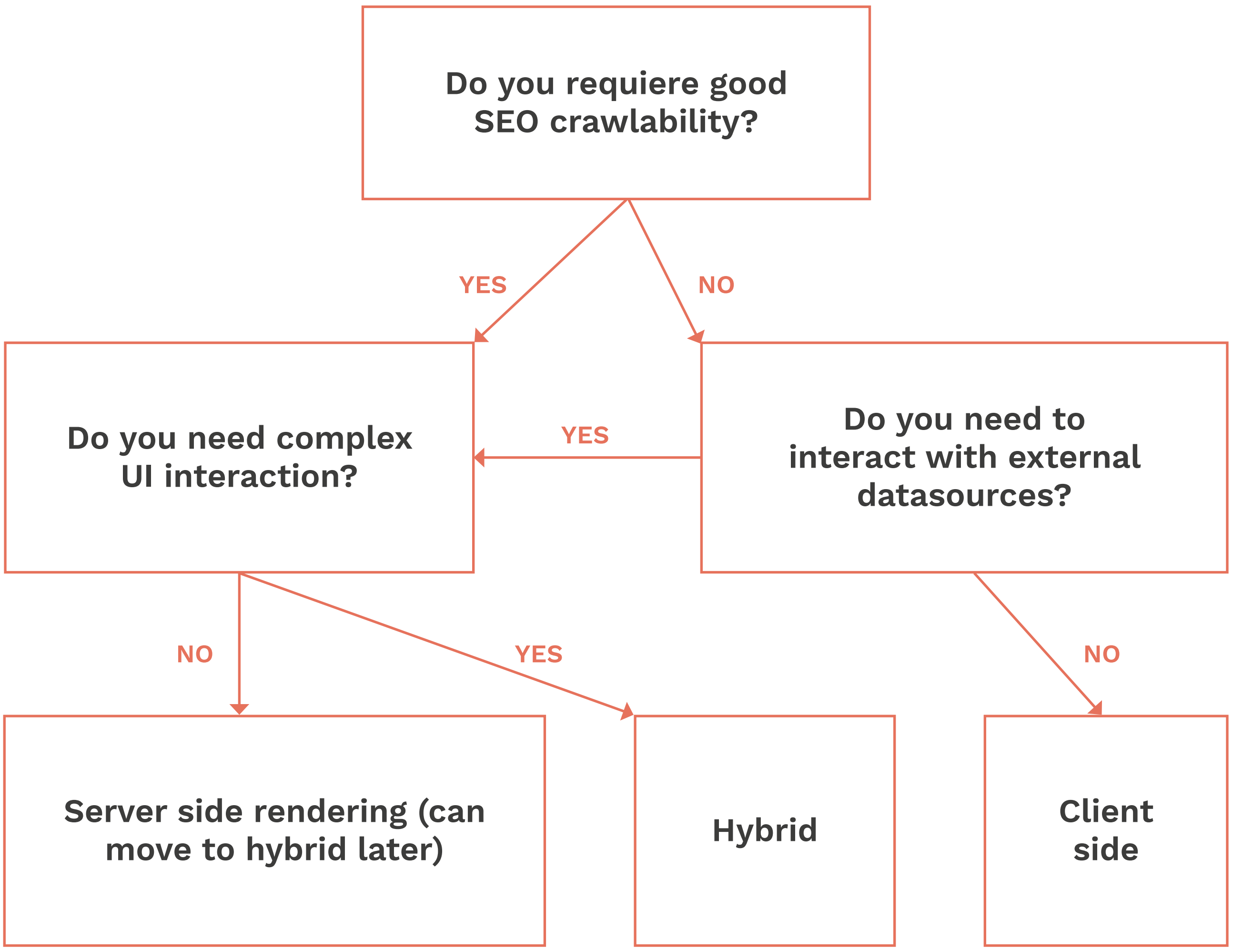
For more complex applications, this is generally the correct approach.

It can be achieved one of two ways. The first approach is with a framework using a 'hydration' approach (frameworks like NextJS handle this well), where a server prerenders your client side code into static pages automatically. This can be a great way to get the benefits of both worlds; but bear in mind if you are displaying external (non-CMS) datasets this can still get very complicated and can require a lot of custom code to call it in a secure way.

The other approach is to have most pages be server side generated, but for areas which require more interaction, embed the client side approach (for example, a React application).

Conclusion

Getting this right at the start of your project is extremely important. We've tried to summarise the key considerations in this flowchart:


Do you requiere good SEO crawlability?
YES
NO
Do you need complex UI interaction?
YES
Do you need to interact with external datasources?
NO
YES
NO
Server side rendering (can move to hybrid later)
Hybrid
Client side

GraphQL type generation

If you're using a statically typed language (C#, Java, Typescript, etc), you'll want to generate type definitions for use in your application. This allows a multitude of benefits for your development team. They'll get context aware code completion in their IDE or text editor, allowing them to discover the structure of the content quickly. Plus your compiler will quickly stop any invalid operations or data access without requiring further tests.

The key part is to use the `get-graphql-schema2` NPM package to download your schema:

```
get-graphql-schema https://graphql.contentful.com/
content/v1/spaces/{SPACE}/environments/{ENVIROMENT}
-h Authorization="Bearer {APIKEY}" > schema.graphql
```

- Where SPACE is your space ID
- ENVIRONMENT is your environment (typically staging or master)
- APIKEY is your Delivery API Key

This will generate a GraphQL type definition file. This can then be uploaded to https://www.graphql-code-generator.com/ imported and exported for your programming environment (which you can select in the dropdown).

You'll get a type file you can include in your project. This will make interacting with Contentful vastly easier.

Furthermore, this can be further automated. Check out the graphql-codegen library: https://github.com/dotansimha/graphql-code-generator. With this, you could integrate this as part of your CI/CD pipeline to ensure the types are always up to date; and fail your deployment if it is running on outdated definitions.

Most read articles

One area many projects get stuck on is implementing most read. For low traffic sites, you can simply add an integer field to your content model and increment it when you pull the content down.

However, for high traffic sites this doesn't work well. It consumes a lot of GraphQL non-CDN API calls, which can get expensive. Even worse it means every page load invalidates your CDN cache, which can cause major performance issues.

We typically implement this with a separate simple database (Azure table storage, DynamoDB, even Postgres). Store each page load as a row, with the datetime and Contentful ID of the page.

You can then use an ajax (to improve cachability of the main page) request on page load to a simple endpoint that can manage this logic.

When you want to rank items by most read, call both Contentful and this table storage, and perform a simple query on it to transform and sort by most read.

Performance

Caching strategies

If you're running a high load site and using server side rendering (>100requests/second), we recommend a cache "in depth" strategy. We'd also recommend putting the site behind a CDN, such as Cloudflare or Imperva, with the correct cache-control headers set on your webpages.

This will significantly reduce the amount of traffic to your application servers. Furthermore, Contentful also caches your GraphQL responses, so you have in effect two caches working in tandem.

This is less of an issue with client side rendering as Contentful will handle all the caching for you.

Everything public (and personal via JS)

A great tip for high load sites that use server side rendering is to try and keep everything cachable — but then alter the site via AJAX to personalise. For example, if you have a login button on your homepage you want to customise, instead of doing so on the server side, you can have the profile information come via Javascript which then changes the 'login' button to 'my account'.

Depending on how much of the homepage is customised, this may or may not be a worthwhile strategy.

However, if it is a small amount, it means all page loads will be cached, with just a small load from the personalisation data that is required, as opposed to not caching these pages at all as they have PII on them.

GraphQL CDN doesn't count towards API rate limits

An often overlooked fact is that when you are calling the GraphQL endpoint, only requests which miss the CDN cache are counted towards your rate limits (which can be quite limiting, at 55/sec). This gives more flexibility but requires some thought to not invalidate the cache by modifying entities too frequently (eg every page load). See the most read section for an example of how to deal with this.

Image management

Contentful has great image management features that work like you'd expect. If you are dealing with a design that requires images in various aspect ratios in different places there are a couple of features which you may find extremely useful to achieve this (without requiring your authors to upload various combinations of those images).

Contentful image API

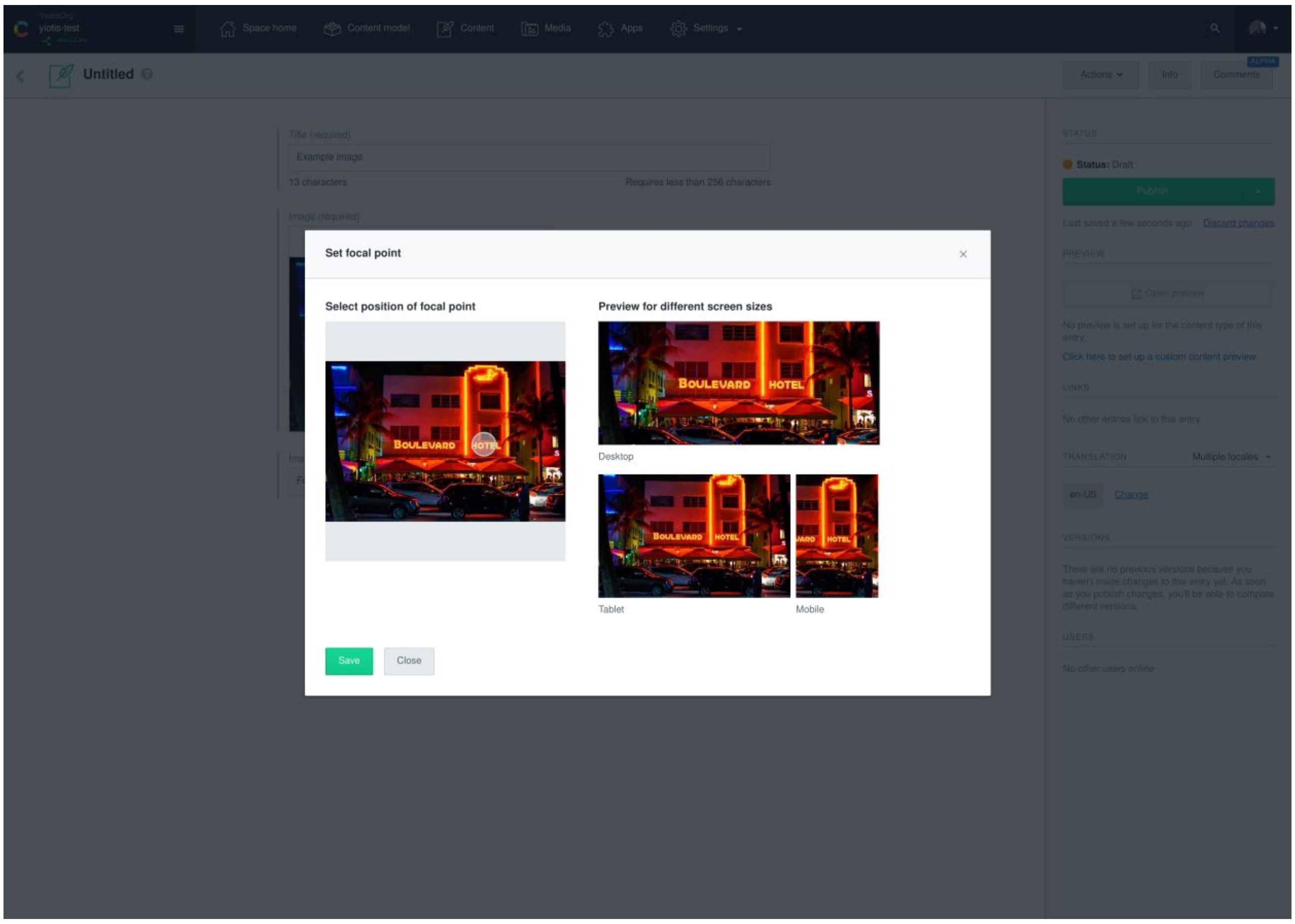
The Contentful image API has a great AI feature which can detect faces and crop the image intelligently. If you're using the contentful image API you can simply add the following:

- `face` for the largest face detected.
- `faces` for all the faces detected.

For many use cases this is simple and works effectively.

Image focal points

If you require further control; or your image tend not to have people in them, you can use the image focal point editor app by Contentful (https://www.contentful.com/help/image-focal-point-app/)


Untitled
Set focal point
Select position of focal point
Preview for different screen sizes
Desktop
Tablet
Mobile
Save
Close

This allows your authors to set the focus of their images and see a preview of various aspect ratios, giving them great control.

An important point: at time of writing, the Contentful image API doesn't support taking the data from this to crop. As such, you will either need to crop on the frontend with this focal point data, or use a 3rd party service such as imgix which does support this data.

Conclusion & about Baytree

We hope you enjoyed this guide to Contentful. Baytree is a leading solution provider for Contentful consultancy and software builds, and we would love to hear from you if you think we can help accelerate your migration to the exciting world of headless CMSes.

Contact us at toma@hellobaytree.com