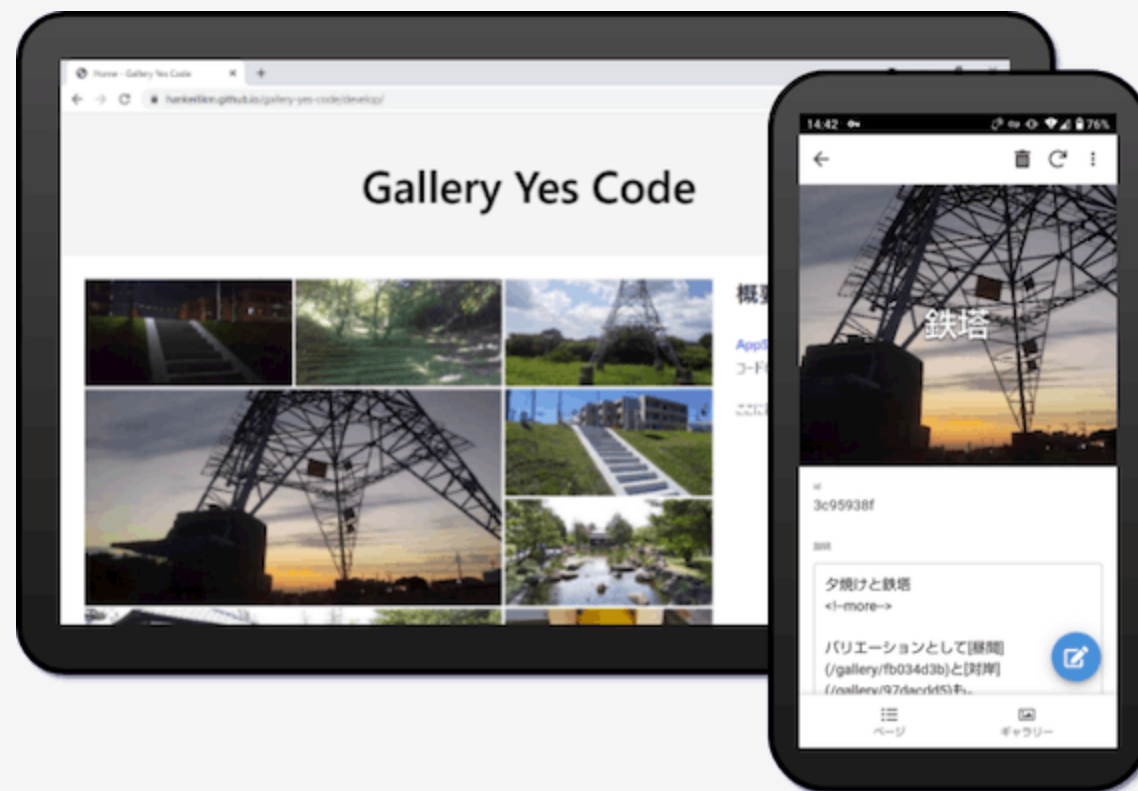


nuxt-content とリモート CMS

AppSheet で作る静的サイト



背景

nuxt-content はデフォルトで「Markdown を Nuxt で扱いやすい状態にしてくれる」ので利用したい。しかし Markdown はローカルのファイルへ保存が前提となっている。

できればリモート CMS のコンテンツを扱いたいので、あまり複雑なことはせずに対応できないか試してみた。

検討

検討してみた方針

1. `<nuxt-content>` コンポーネントのみ利用する
2. リモート CMS を FUSE でマウントする
3. 必要なコンテンツをすべてローカルへダウンロードする

<nuxt-content> コンポーネントのみ利用する

「コンポーネントのプロパティに Markdown をわたせばいいのかな」と思っていたので、最初は(`$content.fetch` を重要視してなかったのもあり)この方法でと考えていた。

しかしながら、実際には「Markdown から生成された AST(微妙に独自仕様ぽい?)」を渡すことになっていた。

AST を作成するメソッドは `$content` の中から引っ張り出せたが、非公式のようなので利用は断念した。

リモート CMS を FUSE でマウントする

リモートの CMS を [FUSE](#) でローカルのファイルシステムにマウントしてしまう(ネタ的には一番面白そう)。

以前に [xrosfs](#) を作った経験からすると技術的にはおそらく実現可能だが、以下の問題が予想されるので、これも断念。

- 実用的なパフォーマンスが得られるか怪しい
- コンテナ内から FUSE でマウントするはそれなりのケーパビリティが必要
 - (試してはないが)GitHub Actions などではマウントできないと思われる

必要なコンテンツをすべてローカルへダウンロードする

必要なコンテンツをあらかじめローカル側へすべてダウンロードしておく。

言葉の響きの的に「効率悪そう」だが「まとめてダウンロードできる」ので API 実行回数的にそれほど悪くないと予想。

落としどころとしてはこの辺かなと。

作ってみた

コンテンツをダウンロードするコマンド

実証実験的に [AppSheet](#) 内のコンテンツをダウンロードするコマンド ([appsheet-to-nuxt-content](#)) を作成。

- 指定したテーブルのコンテンツを行単位ですべてファイルへ保存
- 各カラムのカラム名と型を nuxt-content で扱いやすいように変換
- コンテンツに含まれる **Image** もすべてローカルファイルへ保存
 - 画像のメタ情報(現状ではサイズのみ)も取得
- 参考: [AppSheet をヘッドレス CMS 風を使う](#)

サンプルの静的サイト

ダウンロードしたコンテンツを表示するための静的サイト ([hankei6km/gallery-yes-code](https://hankei6km.github.io/gallery-yes-code/))も作成。

- コンテンツの(ローカルからの)取得と表示は `nuxt-content` を利用
- 画像表示は [nuxt-image](#) を利用
- GitHub Actions で GitHub Pages へデプロイ

使ってみた

コンテンツの編集

編集方法は利用している CMS によって異なるが、今回は AppSheet で作成したフォームを利用しているので通常のアプリと同様に編集可能。

むしろモバイル対応のアプリにより、外出先での編集でもストレスが少ない(Markdown の編集は厳しいが)。

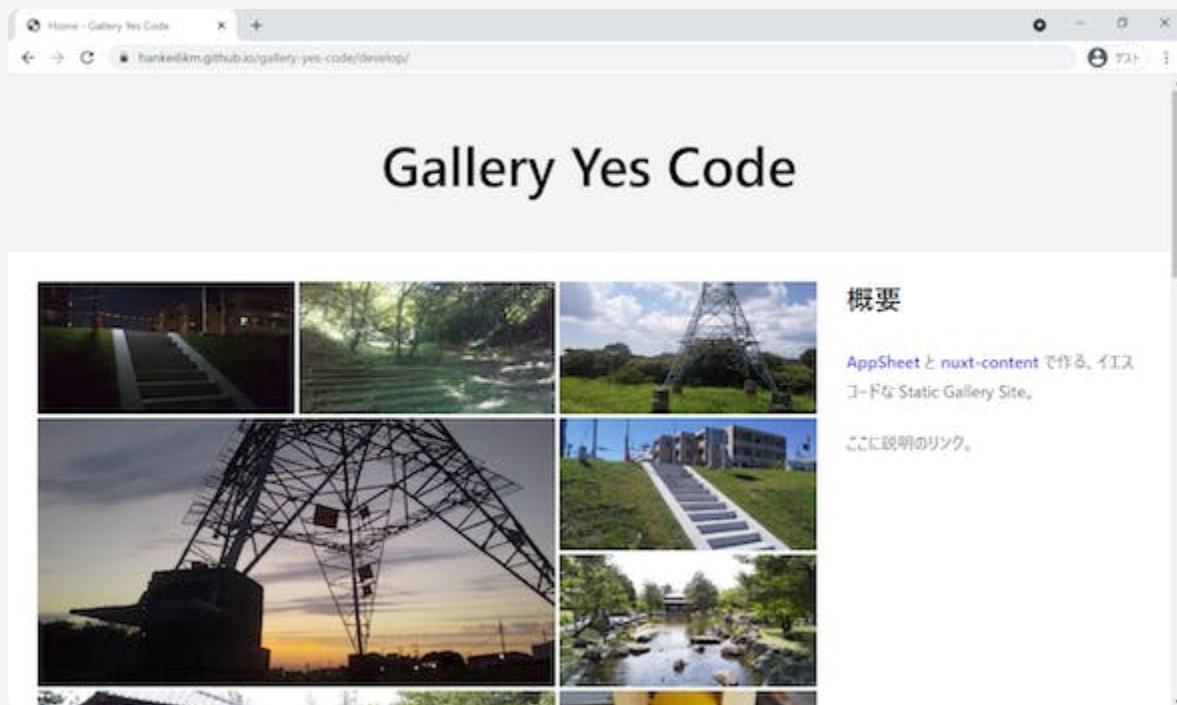
ただしプレビュー画面を実装できていないので、そこについてはやりにくい。



デプロイ

CMS を普通に使う方法と比べても普通に静的サイトが出来上がる。

- [Gallery Yes Code](#)



結果

主な結果

リモート CMS と nuxt-content でサイトを作成してみた感触としては。

- リモートコンテンツの取得とサイト生成(表示)の処理を疎結合にできる
- (今回は)ローカルに画像があるので nuxt-image の最適化が常に使える
- API リクエスト実行回数を減らせる
- 部分的な変更には別途対応が必要

疎結合

疎結合により受けられるメリットはいくつかある。

- API のシークレットなどを「サイトの generate から切り離せる」
 - 別リポジトリで全件ダウンロードしてから push もできる
- ページネーションなどのロジック的な部分とデザイン的な表示部分の作業を分担しやすい
 - 手間はかかるが、ダウンロード時に `content/posts/pages/1.md`
`2.md` `3.md` ...なども作成可能
- コンテンツへのアクセスが `$content.fetch` に統一される
 - リモート CMS を切り替えても生成処理は影響を受けにくい

画像の最適化

nuxt-image の Provider 未対応のメディアライブラリー(今回の Appsheet など)でもローカル(`static/`)に画像をダウンロードしているので、最適化されるようになる。

ただし、キャッシュ処理などを実装しない場合は画像を毎回ダウンロードすることになるので、メディアライブラリーの特性にあわせてどこまで処理するか検討しておく方が良いと思われる。

API 実行回数

Nuxt では generate 時に [payload](#) が使えるので一概にはいえないが、「全件受信しておく」という言葉の響きに反して API の実行回数を減らせる場面は多いと推測できる。

なお、AppSheet API では `429 Too Many Requests` 等が返ってくることがあるが、今回の実験中ではほぼ発生しなかった(429 が 1 回、500 が 1 回発生した)。

課題

やはりコンテンツの更新に制限がかかりやすい。

- コンテンツの部分的な変更に対応しにくい
 - `content/` の下を大幅に変更するとホットリロードが失敗する
- ローカル側のファイルは常に変更される(非破壊的な更新が難しい)
 - プレビューの実装が面倒
- ローカル側の変更を CMS 側へ反映しにくい
 - nuxt-content の [ライブ編集](#) が使えない
- その他、メディアファイルを対象とする場合は転送量が増大する

おわりに

おわりに

実際に簡易的なサイトをしてみた感触としては「静的なサイトを作る方針として nuxt-content + リモート CMS はわりと良い」。

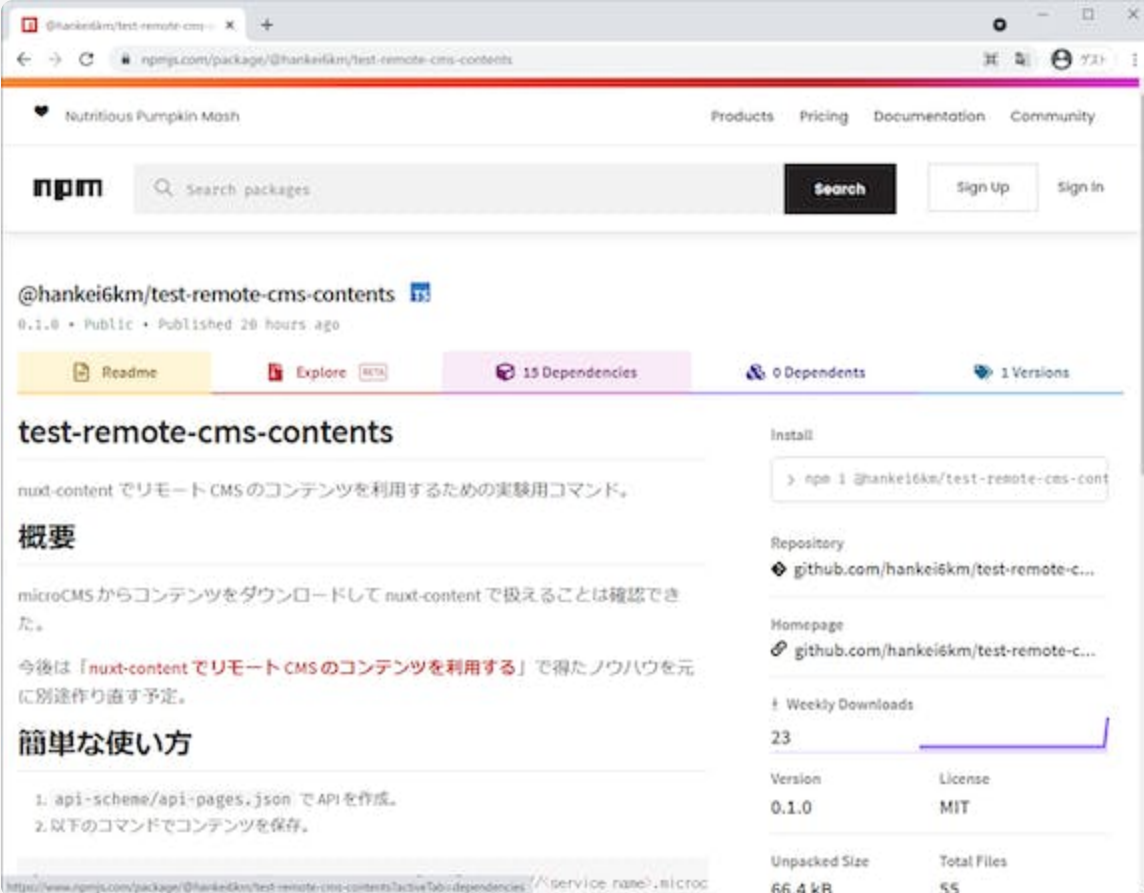
個人的には「リモート CMS 用の処理とサイトを生成する処理」を切り離せることが「特に良い」と感じている。

課題についてはいくつか対策も思い浮かんではいるので(ウェブフック経由で変更点のみ扱うなど)、いずれ対応できればと考えている。

おまけ

実は、AppSheet 版の前に試作したものもあります(こちらは microCMS からダウンロード)。

いまとなっては効率が悪いなどの問題も多いのですが、いちおう動作はします。



The screenshot shows the npm package page for `@hankei6km/test-remote-cms-contents`. The page includes the following information:

- Package name: `@hankei6km/test-remote-cms-contents`
- Version: `0.1.0`
- Published: 20 hours ago
- Dependencies: 15
- Dependents: 0
- Versions: 1

The main content area contains the following sections:

- test-remote-cms-contents**
- Install command: `npm i @hankei6km/test-remote-cms-cont`
- Repository: `github.com/hankei6km/test-remote-c...`
- Homepage: `github.com/hankei6km/test-remote-c...`
- Weekly Downloads: 23
- Version: `0.1.0`
- License: MIT
- Unpacked Size: 66.4 kB
- Total Files: 55

The description states: "nuxt-content でリモート CMS のコンテンツを利用するための実験用コマンド。" and "microCMS からコンテンツをダウンロードして nuxt-content で扱えることは確認できた。" It also mentions that the user will be documenting how to use remote CMS content with nuxt-content in the future.