## **Queue-it Project**

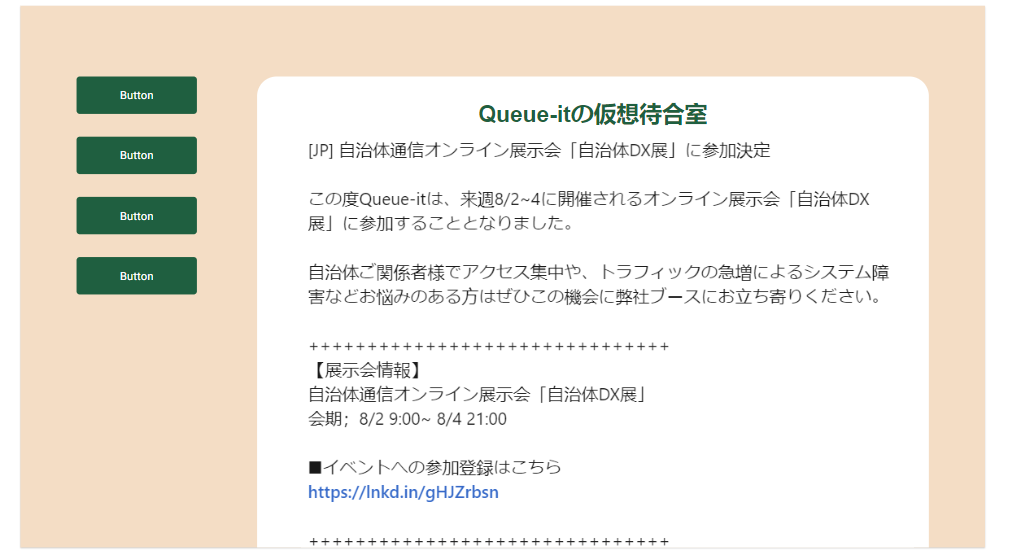
1. Create contents.
2. Call contentful (CMS) API(Post)
3. Svelte App call Svelte App
4. Host Svelte App
5. Load Svelte app using iFrame

**Overview**

A diagram of a software application

Description automatically generated with medium confidence

**Prototype**



**Code Standard:**

* Camel case for source code
* Kebab case for Html file
* Have a full name of variables

**Domain Model**

Noun & Verb List

Noun: marketing team, contentful, goPlatform, Svelte app, customer

Verb: create, upload, use, store, communicate, call, retrieve, analyze

A diagram of a software application

Description automatically generated with medium confidence

|  |  |  |  |
| --- | --- | --- | --- |
| **ToDo** |  | Memo | Status |
| 1. Create contents |  | Field: Text(title), Rich text(Body), Image |  |
| 1. Call contentful (CMS) using Content API Delivery | Call contentful (CMS) using Content API Delivery | **Content Delivery API** which is read-only API for delivering content from Contentful to apps, website and other media as files.  Need access token that has access to the environment you’re requesting content from. The access token, either as an **Authorization request header field**, or as an **access token URI query parameter** |  |
|  | Authentication | You first need to authenticate your app with an OAuth bearer token. A token provides read-only access to one or more environments. |  |
|  | Three steps | 1. Space ID 2. Environment ID 3. Access kye (API) |  |
| 1. Svelte App |  | Create a new project called blog-content-project |  |
|  | Call API |  |  |
|  | Mapping |  |  |

Grocery

|  |  |
| --- | --- |
| CDA | Content Delivery API : Read-only API for delivering content from Contentful to apps |
| CDN | Globally distributed content delivery network . CDNs cache content like web pages, images, and video in proxy servers near to your physical location |
| URI query parameter (Path Param)) | is basically used to identify a specific resource or resources whereas Query Parameter is used to sort/filter those resources. Let's consider an example where you want identify the employee on the basis of employeeID, and in that case, you will be using the URI param.  GET /employee/{employeeID} |
| Bearer Tokens | the predominant type of access token used with OAuth 2.0. |
| Markup | what HTML tags do to the text inside of them; they mark it as a specific type of text |
| Mapping | mapping data between your app and other API integrations |

Challenging

* API of Contentful: every resource returned by the content delivery API will have a sys property, which is an object containing system managed metadata.
* ~~Call API to fetch the data~~
* ~~Javascript~~
* ~~Svelte~~
* If/else error message “Error. Failed to fetch data from Contentful”
* Use store?
* Can I use something from svelt learning? e.g. binning, navigate, separate pages etc …
* onDestroy?

Access elements in array;

**Title**

posts.items[0].fields.title

**Rich text**

**(text only)**

posts.items[0].fields.richText.content[0].content[0].value

**(text with hyperlink)**

posts.items[0].fields.richText.content[0].content[1].content[0].value

**(text with hyperlink)**

posts.items[0].fields.richText.content[0].content[1].data.uri

**(text with bold, underline)**

posts.items[0].fields.richText.content[0].content[0].marks[0].type

**(text with list)**

posts.items[0].fields.richText.content[0].content[0].nodeType

**(text with table:row 2)**

posts.items[0].fields.richText.content[1].content[0]

posts.items[0].fields.richText.content[1].content[1]

**(image)**

posts.includes["Asset"][0].fields.file.url

**Problem Formulation**

If it’s only a plan text, then Array is 0, then render it on the browser.

If I use bold, hyperlink etc, a new array is created automatically, so I assume that if I use other functions in Contentful, then it’ll have issues.

* I fetched data directly from API of contentful directly.
* Use ‘contentful’ page using React, node.js etc …
* Each blog is different.
* Nasted.tree: If I use hyperlink, bold etc, then Nested tree is changed.

Solution

* Fetch data dynamically Title, Body etc
* Use React or node.js

**Show the title**

<p>{posts.items[0].fields.title}</p>

**Show the richText, If the richText is only text;**

{posts.items[0].fields.richText.content[0].content[0].value}

**Show the richText, If the richText is a text with hyperlink and the text;**

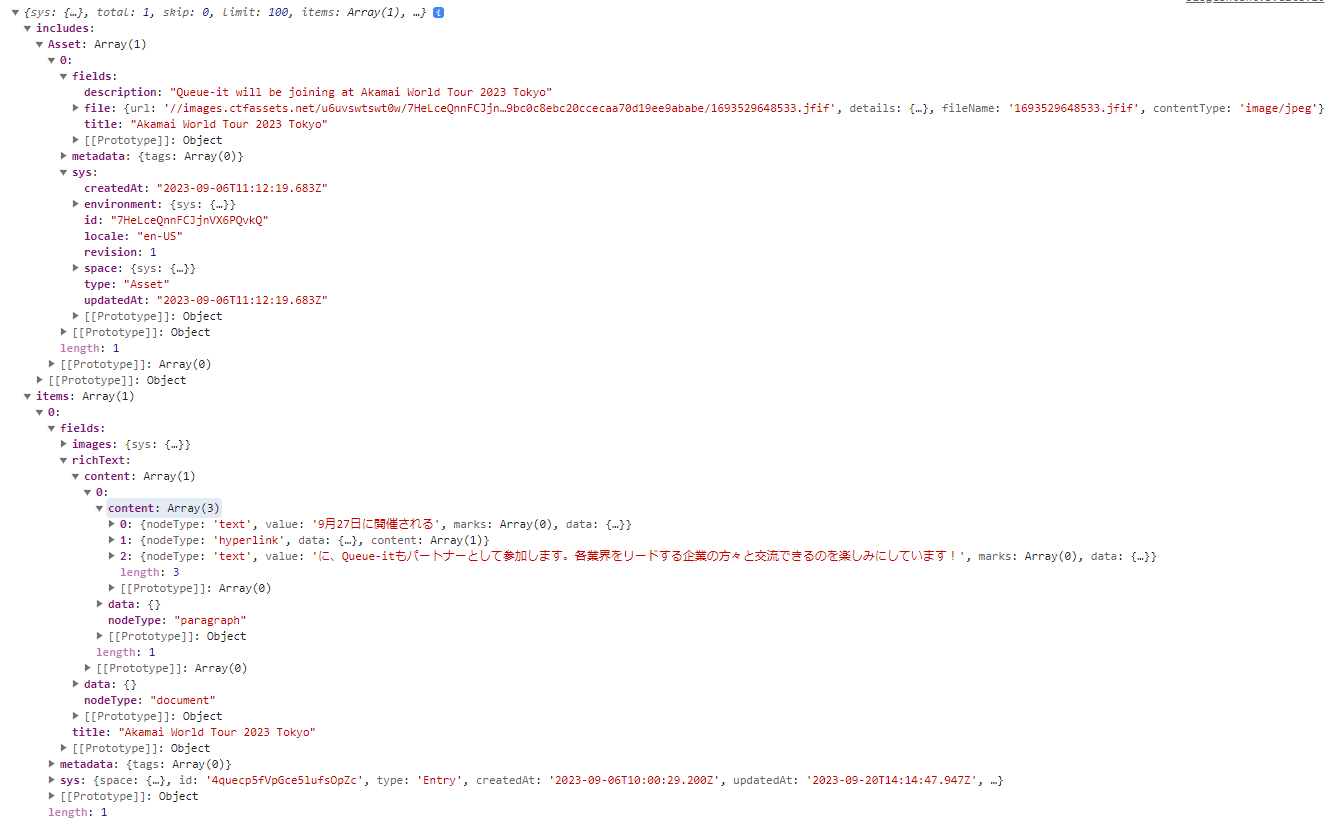
<a href={posts.items[0].fields.richText.content[0].content[1].data.uri}>

{posts.items[0].fields.richText.content[0].content[1].**content**[0].value}</a>

**Show the only richText again;**

**{posts.items[0].fields.richText.content[0].content[2].value}**

|  |  |  |
| --- | --- | --- |
| Functions | Access | Status |
| Normal text |  | Done |
| Heading 1 |  |  |
| Heading 2 |  |  |
| Heading 3 |  |  |
| Heading 4 |  |  |
| Heading 5 |  |  |
| Heading 6 |  |  |
| New Line | posts.items[0].fields.richText.content[2].content[0].content[0].content[0].value |  |
| Bold |  | Done |
| Italic |  | Done |
| Underscore |  | Done |
| Hyperlink |  | Done |
| List | posts.items[0].fields.richText.content[2].content[0].nodeType |  |
| List value | posts.items[0].fields.richText.content[2].content[0].content[0].content[0].value  posts.items[0].fields.richText.content[2].content[1].content[0].content[0].value |  |
| Ordered list |  |  |
| Block Quote |  |  |
| HR |  |  |
| Table |  |  |
| image |  | Done |
| Image in the contents |  |  |



(My syntax: problem with the text is duplicated)

{#if paragraph.nodeType === "text"}

{paragraph.value}{/if}

{#if paragraph.marks}

{#each paragraph.marks as typo}

{#if typo.type === "italic" && paragraph.nodeType === "text"}

<i>{paragraph.value}</i>

{:else if typo.type === "bold" && paragraph.nodeType === "text"}

<b>{paragraph.value}</b>

{:else if typo.type === "underline" && paragraph.nodeType === "text"}

<u>{paragraph.value}</u>

{/if}

{/each}

{/if}

{/each}{/each}

</p>

(Simon helped me)

 <!-- text\_with\_typography -->

        <!-- {#if paragraph.nodeType === "text" && paragraph.marks.length === 0}

          {paragraph.value}

        {/if}

        {#if paragraph.marks}

          {#each paragraph.marks as typo}

            {#if typo.type === "italic"}

              <i>{paragraph.value}</i>

            {:else if typo.type === "bold"}

              <b>{paragraph.value}</b>

            {:else if typo.type === "underline"}

              <u>{paragraph.value}</u>

            {/if}

          {/each}

        {/if} -->

Simon advice

Instead of using <i>, <b> and <u> tags, it’s better to create class and use them in the element

   <p

                class={paragraph.value === "bold"

                  ? "bold"

                  : paragraph.value === "underline"

                  ? "underline"

                  : ""}

              >

                {paragraph.value}

              </p>

<style>

bold {

font-weight: bold;

}

Underling {

Text-decoration: underline;

}

</style>