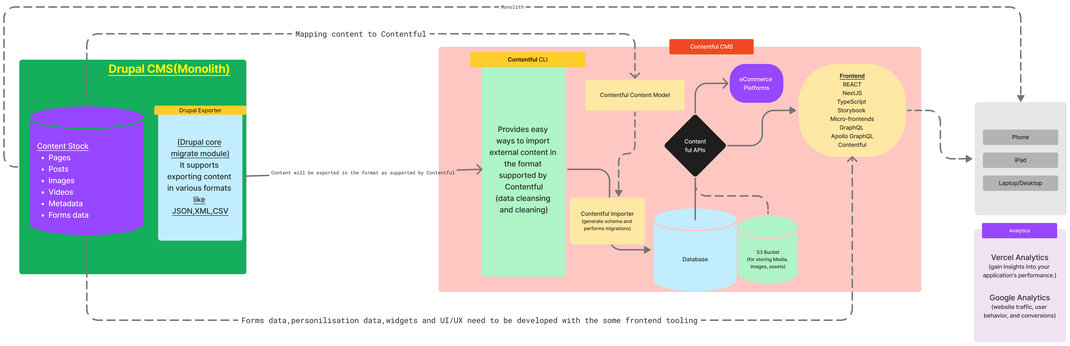
Migration of Monolith CMS to a Headless-based approach.



Goals

1. Target new customer segments and expand as fast as possible including internationally with a new headless platform - Contentful.

2. Enable Product and marketing teams to build pages faster and manage content independently of developers and empower them to build pages, test features, and release new content with minimum help from developers.

3. The new CMS will empower product and marketing teams to build pages, test features, and release new content without developers.

Tasks and responsibilities you would expect the engineers on the team to be undertaking.

Tech Lead

Setup focused DevOps cross-functional Agile feature teams. For example
1. Platform (responsible for developing and maintaining global features i.e Header, Footer, Navigation and SEARCH)
2. Identity (Responsible for Login, Profiling and Customisation, Personalisation)
3. Content (composing page types and categories).

Team

Before starting the migration, it is important for the team under the supervision and directives of the Tech Lead to plan out the process and identify any potential challenges or obstacles. Mapping out customer journeys, and pain points and writing and integrating agile user stories into core workflows.

User Stories

1. **Content identification:**
- Identify the content that needs to be migrated. Could include pages, posts, images, videos, and other types of content. Categorising them. Constructing an up to date Information Architecture (IA) to list out the pages of content that need to be considered for migration or the pages that do not.
- determine the format and structure of the content.
- set up a schedule for the migration.

2. **Exporting content: Export the content from the monolith CMS (probably WordPress or Drupal)**. This may involve using built-in export features or writing custom scripts to extract the content. Tools for exporting content, such as XML or CSV exports are normally built in. It is important to ensure that the content is exported in a format that can be easily imported into the headless CMS. Alternatively, typically involves writing scripts or code that interact with the CMS's database and API to retrieve the desired content and format it for export. eg. https://github.com/jonashcroft/wp-to-contentful . Ensure the necessary team permissions are in place to access the Database and that a backup is in place.

3. **Transforming data:** Once the content has been exported, it will have to be transformed to fit the format and structure expected by the headless CMS. This will involve cleaning up the content, splitting it into smaller pieces, or combining it with other data. The exported data content could be in various formats, depending on the export solutions that may have been used.

4. **Clean and prepare transformed data for import**: This will include removing any duplicates, fixing any formatting issues, and ensuring that the content is in a format that can be easily imported into the headless CMS. My recommendation is that when working with rich text elements, it's safer to go minimalistic and remove any unwanted and unrecognized tags that are not allowed before importing the content into the new CMS Headless CMS.

5. **Importing content:** The final step is to import the transformed content into the headless CMS. This may involve using built-in import features (CLI), CSV import or writing custom scripts to load the content into the CMS. Contentful has an in-built CLI utility to import content. It is important to carefully test the imported content to ensure that it has been migrated successfully.

6. Re-usable Component and Layout build: Create reusable structured React component library aligned to the business brand image in Storybook, as well as documentation in Storybook (reusable components: ie Header, Footer, search, Navigation, FORM validation), laying the foundation for a design system.

7. Construct API - and render content: Build APIs and queries with GraphQL, Prisma-ORM and NextJS to fetch data, leverage interactivity and make components composable and functional to reduce initial load time by adding support for SSR, ISR, SSG and images optimisation.

8. Image migration: Low-resolution images may need phasing out for better images for the new CMS. Better naming conventions may be brought into place. In those situations, basic image requirements will be made into a task to help filter out the good from the bad. Revisions and duplications would need to be removed to avoid clutter in the new CMS.

9. Formats of data fields: Need to look at the format of the different data fields from the new CMS and map them via transformation scripts to meet the new requirements. Checking on instances where the text, date, and number fields could be in slightly different data formats, and can cause a lot of pain if it is not sorted out before the import process.

10. Isolating Script tags: Monolith CMS normally allow the capability to include script tags for execution. Tasks will be scheduled to remove anything that might disrupt or cause the import process to fail for the new CMS.

11. Feature Flagging and A/B Testing: Set up a feature flagging script that is able to turn parts of the Website ON or OFF once ISR is implemented with NextJS. Could use LaunchDarkly - https://launchdarkly.com/

Challenging tasks and ensuring continued operation of existing CMS

1. Integrating new CMS with parts of existing technology stack: New APIs and integration points will be built as tasks to connect Contentful headless CMS to Front-end applications and other systems. Using NextJS, TypeScript, GraphQL, Prisma ORM, and Nexus to support typedefs.

2. Capturing FORMS and tabular data: Handling features like form capturing or data stored in tabular format to integrate this into the new CMS. Use the embedded Rich Text module.

3. Handling Personalisation from monolith: It's important to understand that moving to a headless CMS some features and functionality utilized in the monolith CMS will not be available in Contentful, hence need to consider third-party offerings to help bridge this gap. This includes personalisation features and widgets. A strategy for this will be factored in to enable content to be utilized in the new CMS, which could then put this piece of content into a user story category of the "need to apply after migration". This will also include handling custom data types and data structures that are specific to existing CMS.

4. Maintain secure access to content or assets that are viewed via authentication: Making sustainable steps to maintain secure access to content or assets that are viewed via authentication on the website away from new content that may not need secure protection. Achieved by creating an authentication model with multiple security levels of roles using OAuth or similar authentication libraries. A separate task will be built to export old profiles if necessary. Also, note old published content will not be exported to the new CMS. New content to the Website will come from the new CMS when publishing starts.

5. Content lockdown: As content from the monolith CMS may still be edited by the existing team there will be a need to consider content lockdown or investigate the last modified date or even the ability to export changes and factor this into your solution when gathering the exported content.

Non-functional considerations and implications.

1. Integrate analytics using Amplitude, **Vercel analytics or Google Analytics** and assess the impact of new features, flows and customer behaviour with A/B testing.

2. A/B Testing will be crucial to maintain the monolith working as BAU whiles building the new CMS and managing expectations and baselines.

3. The tools should:
- Offer capabilities for Segmentation to target preferred audience groups.
- Provide statistical analysis and reporting to provide ROI, and CRO (Conversion Rate Optimisation).
- Options for both Server Side and Client side testings based on complexity.
- Should sync and Integrate seamlessly with the software product.

Processes and workflows the team needs to have in place and why.

1. Workflows implemented using Agile methodology and SCRUM Teams.

2. Scrum Master to streamline cross-functional communication, enabling development team to scale, align, and pivot with the business.

3. Feature-focused scrum teams will improve productivity by gathering feedback regularly, reducing context switching, and facilitating predictive delivery.

4. Scrum teams will resolve bugs earlier and reduce the burden on quality assurance with continuous testing involving unit, E2E, and acceptance tests using automated CI/CD pipeline.

Stakeholders that you would need to engage with, and potential implications for them, as part of these changes.

1. Cost and operational excellence: A business decision needs to be taken on the operational excellence, cost optimisation and benefits of adopting a headless CMS.

2. Payment, Subscription plans and Hosting cost: These need to be investigated, either subscription-based or fixed-cost - which one will be more beneficial to the business.

3. Learning and Training: On top of all that, there is always a learning curve for the engineers to get through just as with any digital transformation. Training and presentations to the Leadership team members would be a positive approach, and a first step to sharing knowledge and benefits that come with the change.

4. Imported content from external sources: e.g., PIM (Product Information Management), CRM (Customer Relationship Management), PostgreSQL. Using ORM like Sequilizer, Prisma and Apollo GraphQL in conjunction with NextJS, such external content can be extracted.

5. Localisation: Another consideration is whether all the categories need to cover localized content, and if they do, that would add another level of complexity to consider the handling of content and its translations for the different types of content categories.

6. Versioning: Recommendations would be to only migrate published content as the destination CMS would have no history of versioning from the old CMS. Checks would be carried out to ensure, all content that needs to be migrated is in the correct published state.

7. Personalised content: It's important to understand when moving to a headless CMS that some features and functionality utilized in the monolith CMS may not be available in Contentful, hence need to consider third-party offerings to help bridge this gap. A strategy for this should be factored in to enable content to be utilized in the new CMS, which could then put this piece of content into the "need to apply after migration" category.

8. Page build Widgets from monolithic CMS: Worth exploring if need be to re-create often such complicated features in Contentful as they normally can't be exported.

References:

1. Contentful - https://www.contentful.com/developers/docs/tutorials/cli/import-and-export/
1. NextJS - https://nextjs.org/
2. Storybook - https://storybook.js.org/
3. GraphQL - https://graphql.org/
4. Apollo GraphQL - https://www.apollographql.com/
5. Prisma - https://www.prisma.io/apollo
6. LaunchDarkly - https://launchdarkly.com/