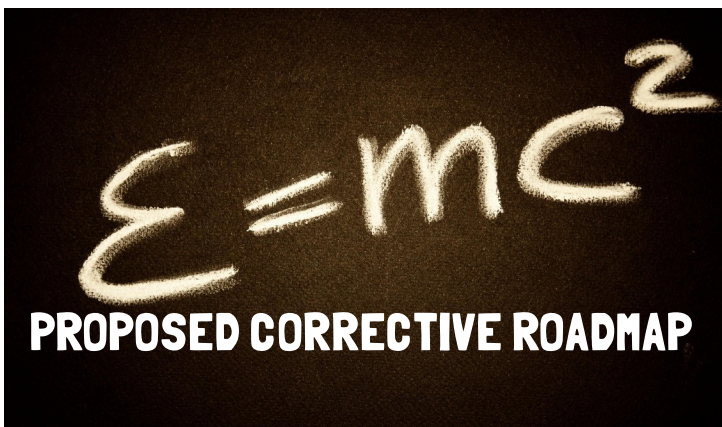




CONTOSO SITECORE QUICK START



TABLE OF CONTENTS





SITECORE INFORMATION ARCHITECTURE

Information architecture is very important in CMS systems, especially in Sitecore. As of the last few years, Sitecore has made a push to a best practices pattern that is referred to as Helix. Essentially it allows content authors and marketers to easily use all of the advanced Sitecore features. In addition, it makes it possible to easily maintain the code base, reducing costs. More on Sitecore Helix here: <https://helix.sitecore.net/>.

Looking through the IA of the existing site, we noticed a few issues.

Template structure does not follow helix

We found that, although it partially follows the recommended Helix structure, there are a few lapses. First, the order of inheritance is incorrect. The Feature layer should be inherited by the project layer, then the project layer should be used as the templates that build out a site under the content node. The reason that this is important is because it would make it easier to add more sites to the solution as well as ensure that the Sitecore templates are well ordered and easy to maintain.

Large number of items under a single parent

We found a few examples where there was or was the potential to have more than 100 items under a single parent item. This is generally against Sitecore best practices since this can cause potential performance issues for the solution, based on the way that Sitecore access and stores information in the database. One example was found here: `/sitecore/content/Contoso/Migration/Unknown/News Articles`. Generally, the best practice is that such instances should be converted to item buckets which allows Sitecore to quickly find items using the search index instead of expensive database queries.

CODE QUALITY

Generally speaking, code upkeep is something that is often ongoing especially in a dynamic application. After analyzing the code base used, we found the following issues.

Unused code files

In the solution we found several locations where there were empty cs files:

- `src\Feature\Accordion\code\Repositories\AccordionRepository.cs`
- `src\Feature\Accordion\code\Repositories\IAccordionRepository.cs`
- `src\Feature\Accordion\code\Templates.cs`
- `src\Feature\ArticlePages\code\Repositories\ArticlePagesRepository.cs`
- `src\Feature\Carousel\code\Repositories\CarouselRepository.cs`
- There were many more

One thing to keep in mind is that we were provided with an exported copy of the code base, so perhaps in other branches (in a Git repository) this was already corrected. The reason that this is an issue is because it will be confusing for other development teams that need to maintain the code base but did not build it originally. It also violates basic development practices concerning keeping ones code clean.

Empty test projects

In the solution we found several examples of test projects that were created, but not implemented yet.

- Contoso.Feature.CallToAction.Tests
- Contoso.Feature.AssociateBio.Tests
- Contoso.Feature.Navigation.Tests
- Contoso.Feature.VideoPlayer.Tests
- Contoso.Feature.Valo.Tests
- There were more examples

This can be an issue as it is generally recommended to use test projects to catch potential issues at build time which helps prevent issues/bugs from making it into the higher environments, especially production. In addition, this helps automate the testing process, facilitating better resource utilization. The other issue with this is that it would violate general coding standards since one does not leave unused projects or class files, etc, in a solution, especially one that is live.

Dependency injection is not used consistently

There were several examples of where dependency injection was not consistently used throughout the project.

- src\Feature>ContactRequest\code\Mailing\RecipientProvider.cs
- src\Feature\GlobalSearch\code\Repositories\GlobalSearchSettingsRepository.cs
- src\Feature\ArticlePages\code\Services\ArticleService.cs
- There were more examples

Dependency injection is used as a common design pattern in modern development because it allows easier automated testing, dynamically choose functionality at runtime and makes it easier to set well defined parameters around what can be extended and what cannot be. The current code base violates general coding practices as it is not consistent, although it does use it in other parts of the code base. This can cause issues for future developers that would need to learn the system so as to maintain or extend it.

Inefficient methods for accessing Sitecore items

There are several issues that were found that not only go against Sitecore API best practice but can have an effect on the maintainability. First, there are several locations where examples of accessing Sitecore items using hard coded field names

- src\Feature>ContactRequest\code\Mailing\RecipientProvider.cs
- src\Feature\Location\code\Controllers\LocationApiController.cs
- src\Project\Common\code\Controllers\SectionColumnsController.cs
- There are more examples

Generally, accessing Sitecore items using hard coded field names is discouraged. This can cause issues with testing, allowing bugs/issues to reach production. This is also very difficult to manage change, for example, if someone altered the field name, any related functionality would be adversely affected.

Another issue is using Sitecore quires to access items. Here are some examples

- src\Feature>ContactRequest\code\Mailing\RecipientProvider.cs
- src\Feature\Location\code\Sc\Pipelines\HttpRequest\ContextLocationResolver.cs
- src\Feature\GlobalSearch\code\ComputedFields\BaseContentField.cs

- src\Feature\GlobalSearch\code\SearchAdapters\ResultMapper.cs

The issue with this is related to performance. In general, the Sitecore recommended way to access a single item is using an item id and when needing to access many items one should use the search API. This reduces the strain on database access, which can cause very slow load times.

Violations of Helix principles

Within the solution there appears to be several violations of Helix principles. An example of such would be the fact that there are features referencing other features. Some examples would be found on:

- Contoso.Feature.Accordion project
- Contoso.Feature.ArticlePages project

In addition, there appears to be cross references between features and project layer of the solution. The main reason that this an issue is for maintainability of code. In general, when there is a function in a feature that needs to be used by others, it should be refactored into the foundational layer so that it can be shared between features.

FRONT END CODE QUALITY

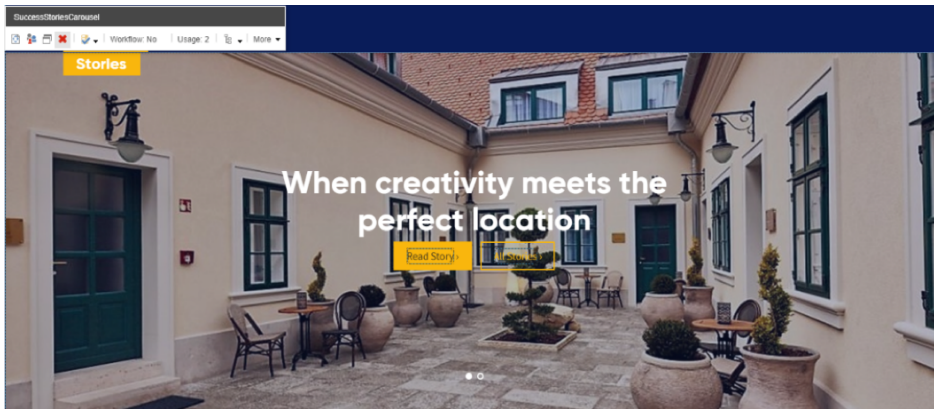
Many of the performance related issues were found to be on the front end of the application. Here are some issues that were identified.

No bundling or minification

According to several scans that we ran (more on that later) and from looking through the front-end code base, there appears to be no minification or bundling of CSS or JS files. The reason that this is important is because it reduces the number of calls to the server when the page is loading. Obviously, the more calls made, the longer it takes to load. Also, minification is a process of removing all whitespace and reducing the character size of methods, properties, etc, to reduce the size of the file that is loaded, causing it to load faster.

Static parts of components

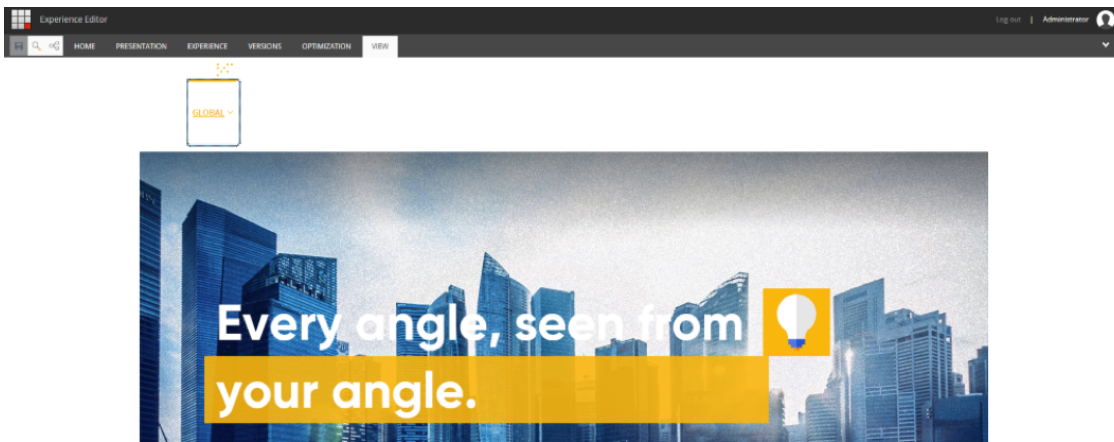
There are several issues that were found around experience editor functionality in the current implementation. There are several items, so here we list only some examples of what has been found. Some components were found to have portions of it that are not editable in Experience Editor. For example, the footer and header component have a few areas that are not editable, for example:



A lot of this is due to the information architecture and how the components were built. This can cause issues for editing as well as possibly have an effect on personalization and other Sitecore functionality in the future.

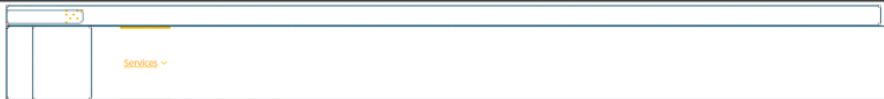
Components not optimized for Experience Editor

There were a few examples that were found where a component was not optimized for editing in Experience Editor. One example is the Header component:

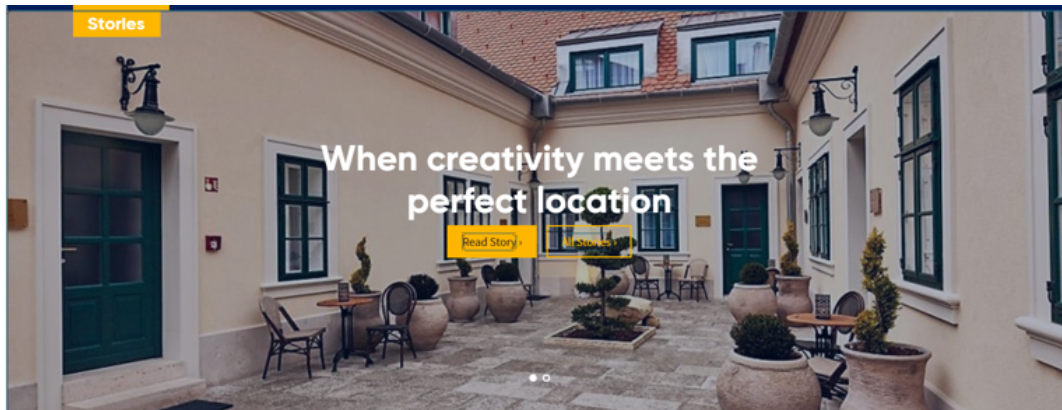


There are other examples of this:

Navigation components, not showing the subnavigation)



The slider component is an example of a control that is not easily manageable in the Experience Editor interface. Normally things like rendering parameters or edit frames would be used



Other items were noticed that seemed to be bugs, for example blinking pagination per click

SEO

SEO is an acronym for Search Engine Optimization. Essentially, it ensures that a web application will be picked up and indexed in a favorable way by a search engine, i.e., Google. There are several standards for SEO optimization that Google has published to assist site owners in their discoverability in Google searches. Using an SEO scanning tool online, <http://www.seowebpageanalyzer.com/>, we received the following highlights from the generated report:

- Overall score was 41 (of 100)
- No www redirection, i.e., going to <http://contoso.com> does not redirect to www.contoso.com
- Many pages have the title tag, however, many more pages are missing the description tag. This is important as it helps your page ranking by declaring what each page is about
- Improper use of heading tags. Heading tags are the `<h1>`, `<h2>`, etc in HTML. The problem is that sometimes they are used for styling instead of for declaring the level of importance of each headline on the page. In general, there should be only one h1 tag on the page near the top that is a short description about the page. There should only be 2 h2's and 3 h3's and they should be in order on the page, i.e., h3 should not be before h2

```

Heading
<h1> getting occupiers the best results
  <h2> latest insights
    <h6> News
    <h6> December 12, 2018
  <h3> cresa expands leadership team with new executive vice president of enterprise accounts
    lisa fry
    <h6> News
    <h6> December 12, 2018
  <h3> the ripple effect of amazon's move to crystal city
    <h6> News
    <h6> December 10, 2018
  <h3> it's all about people
  <h2> what we do
    <h5> Consulting
    <h5> Facilities Management
    <h5> Global Portfolio Solutions
    <h5> Investment Banking
    <h5> Lease Administration
    <h5> Location Strategy & Economic Incentives
    <h5> Project Management
    <h5> Transaction Management
    <h5> Workplace Intelligence
  <h2> how can we help you?
    <h3> lease about to expire?
    <h3> need a real estate strategy?
    <h3> looking for a better workplace?
  <h2> success stories
  <h2> a global real estate partnership with akamai
  <h2> marshalls distribution center
  <h2> discovering shadow space in a tight submarket
    <h5> Global
    <h6> Contact
    <h6> Social
    <h6> Links

```

Here a sample header tag hiarchy is demonstrated based on current content in the site. Notice the placement of the header 6 tag? This would generate less than optimal results for your ranking in Google SEO as well as others

- Many images are missing the alt tag or the alt tag is empty. This can also have a negative effect on SEO and is also a violation of 508 compliancy as it causes issues for those that use screen readers, for example

Here are some of the important points to remember when it comes to SEO for an online presence.

Create good titles and snippets in search results

If your document appears in a search results page, the contents of the title tag may appear in the first line of the results (if you're unfamiliar with the different parts of a Google search result, you might want to check out the anatomy of a search result video¹⁷).

The title for your homepage can list the name of your website/business and could include other bits of important information like the physical location of the business or maybe a few of its main focuses or offerings.

Best Practices

Accurately describe the page's content

Choose a title that reads naturally and effectively communicates the topic of the page's content.

Avoid:

- Choosing a title that has no relation to the content on the page.
- Using default or vague titles like "Untitled" or "New Page 1".

Create unique titles for each page

Each page on your site should ideally have a unique title, which helps Google know how the page is distinct from the others on your site. If your site uses separate mobile pages, remember to use good titles on the mobile versions too.

Avoid:

- Using a single title across all of your site's pages or a large group of pages.

Use brief, but descriptive titles

Titles can be both short and informative. If the title is too long or otherwise deemed less relevant, Google may show only a portion of it or one that's automatically generated in the search result. Google may also show different titles depending on the user's query or device used for searching.

Avoid:

- Using extremely lengthy titles that are unhelpful to users.
- Stuffing unneeded keywords in your title tags.

Use the "description" meta tag

A page's description meta tag gives Google and other search engines a summary of what the page is about. A page's title may be a few words or a phrase, whereas a page's description meta tag might be a sentence or two or even a short paragraph. Like the <title> tag, the description meta tag is placed within the <head> element of your HTML document.

What are the merits of description meta tags?

Description meta tags are important because Google might use them as snippets for your pages. Note that we say "might" because Google may choose to use a relevant section of your page's visible text if it does a good job of matching up with a user's query. Adding description meta tags to each of your pages is always a good practice in case Google cannot find a good selection of text to use in the snippet. The Webmaster Central Blog has informative posts on improving snippets with better description meta tags and better snippets for your users. We also have a handy Help Center article on how to create good titles and snippets.

Accurately summarize the page content

Write a description that would both inform and interest users if they saw your description meta tag as a snippet in a search result. While there's no minimal or maximal length for the text in a description meta tag, we recommend making sure that it's long enough to be fully shown in Search (note that users may see different sized snippets depending on how and where they search), and contains all the relevant information users would need to determine whether the page will be useful and relevant to them.

Avoid:

- Writing a description meta tag that has no relation to the content on the page.
- Using generic descriptions like "This is a web page" or "Page about baseball cards".
- Filling the description with only keywords.
- Copying and pasting the entire content of the document into the description meta tag.

Use unique descriptions for each page

Having a different description meta tag for each page helps both users and Google, especially in searches where users may bring up multiple pages on your domain (for example, searches using the site: operator). If your site has thousands or even millions of pages, hand-crafting description meta tags probably isn't feasible. In this case, you could automatically generate description meta tags based on each page's content.

Avoid:

- Using a single description meta tag across all of your site's pages or a large group of pages.

Use heading tags to emphasize important text

Since heading tags typically make text contained in them larger than normal text on the page, this is a visual cue to users that this text is important and could help them understand something about the type of content underneath the heading text. Multiple heading sizes used in order create a hierarchical structure for your content, making it easier for users to navigate through your document.

Best Practices

Imagine you're writing an outline

Similar to writing an outline for a large paper, put some thought into what the main points and sub-points of the content on the page will be and decide where to use heading tags appropriately.

Avoid:

- Placing text in heading tags that wouldn't be helpful in defining the structure of the page.
- Using heading tags where other tags like `` and `` may be more appropriate.
- Erratically moving from one heading tag size to another.

Use headings sparingly across the page

Use heading tags where it makes sense. Too many heading tags on a page can make it hard for users to scan the content and determine where one topic ends and another begins.

Avoid:

- Excessive use of heading tags on a page.
- Very long headings.
- Using heading tags only for styling text and not presenting structure.

OVERALL PERFORMANCE OF SITE

Other important issue was site performance. To help set a baseline, we ran 2 performance tests on the live site to determine what areas can be improved.

<https://gtmetrix.com>: according to this scan there were a few issues effecting the overall performance of the site:

- Minify JS. As mentioned before, the lack of bundling and minification of JS is causing performance issues with the site
- Optimization of images. Some of the images are not optimized for web. In general, Photoshop (or other apps) can be used to reduce the footprint of an image, resulting in quicker load times
- Browser caching. Browser caching should be used to reduce the amount of repeat loads of content, helping improve site speed
- Gzip compression reduces the size of content that the server delivers to the client before decompressing it. This also increases page load time
- A lot of inline JS and CSS. Sometimes with a CMS system, inline CSS can be useful but should be avoided as much as possible since it will effect maintainability and speed
- Refactor of JS/too many DOM components. Based on this scan, it appears that the JS code base is pretty fragmented and creates too many DOM components on a page
- There appears to be a lot of images that are loading onto the page. Sitecore media library is effective for some media content; however, in the case of large amounts of media, documents, images, video and others, a CDN is recommended. There are several that can be used

Page speed was roughly averaging 5.1 seconds. Cleaning up these issues and using Sitecore cache should be able to drop that average to between 1 – 3 seconds. (*report shown on next page*)

<https://developers.google.com/web/tools/lighthouse> : According to this tools scan, the following issues were found. (shown on next page)

Many of the same issues were identified in this report:

- Defer offscreen images. This simply means using a lazy loading technique were a page above the

fold (or visible content) will load first while deferring other content to be loaded later, i.e., when the page is scrolled, a component clicked, etc

- Serve images in next gen formats. JPEG 2000, JPEG XR, and WebP are image formats that have superior compression and quality characteristics compared to their older JPEG and PNG counterparts. Encoding your images in these formats rather than JPEG or PNG means that they will load faster and consume less cellular data
- Minify JS. This was mentioned in the previous report
- Defer unused CSS. This is a method of not loading CSS classes that are not used on a specific page
- Excessive DOM size. This is another concern as it means slower load times and it can also mean that the JS that has been coded so far is not in an optimal and maintainable state
- Caching was mentioned in the previous report as well



Performance Report for:

Report generated: Wed, Dec 19, 2018, 2:27 AM -0800
 Test Server Region: Vancouver, Canada
 Using: Chrome (Desktop) 62.0.3202.94, PageSpeed 1.15-gt1, YSlow 3.1.8

PageSpeed Score C (71%)	YSlow Score C (74%)	Fully Loaded Time 5.1s	Total Page Size 2.60MB	Requests 35
-----------------------------------	-------------------------------	----------------------------------	----------------------------------	-----------------------

Top 5 Priority Issues

Minify JavaScript	<input type="text" value="F (0)"/>	AVG SCORE: 89%	JS	HIGH
Optimize images	<input type="text" value="F (41)"/>	AVG SCORE: 72%	IMAGES	HIGH
Leverage browser caching	<input type="text" value="E (56)"/>	AVG SCORE: 61%	SERVER	HIGH
Enable gzip compression	<input type="text" value="C (73)"/>	AVG SCORE: 86%	SERVER	HIGH
Inline small CSS	<input type="text" value="A (92)"/>	AVG SCORE: 96%	CSS	HIGH

How does this affect me?

Studies show that users leave a site if it hasn't loaded in 4 seconds; keep your users happy and engaged by providing a fast performing website.

As if you didn't need more incentive, **Google has announced that they are using page speed in their ranking algorithm.**

About GTmetrix

We can help you develop a faster, more efficient, and all-around improved website experience for your users. We use Google PageSpeed and Yahoo! YSlow to grade your site's performance and provide actionable recommendations to fix these issues.

About the Developer

GT.net
Performance Hosting

GTmetrix is developed by the good folks at **GT.net**, a Vancouver-based performance hosting company with over 22 years experience in web technology.

<https://gt.net/>

What do these grades mean?

This report is an analysis of your site with Google and Yahoo!'s metrics for how to best develop a site for optimized speed. The **grades you see represent** how well the scanned URL adheres to those rules.

Lower grades (C or lower) mean that the page can stand to be faster using better practices and optimizing your settings.

What's in this report?

This report covers basic to technical analyses on your page. It is categorized under many headings:

- **Executive:** Overall score information and Priority Issues
- **History:** Graphed history of past performance
- **Waterfall:** Graph of your site's loading timeline
- **Technical:** In-depth PageSpeed & YSlow information

These will provide you with a snapshot of your performance.



Performance



Progressive Web App



Accessibility



Best Practices



SEO

Score scale: 0-49 50-89 90-100

Performance



Metrics

First Contentful Paint	2.0 s ✓	First Meaningful Paint	2.1 s ✓
Speed Index	5.3 s ⓘ	First CPU Idle	3.7 s ⓘ
Time to Interactive	10.4 s ⚠	Estimated Input Latency	10 ms ✓

View Trace

Values are estimated and may vary.



Opportunities

These optimizations can speed up your page load.

	Opportunity	Estimated Savings
1	Defer offscreen images	6.6 s ✓
2	Serve images in next-gen formats	5.7 s ✓
3	Efficiently encode images	2.7 s ✓
4	Minify JavaScript	1.05 s ✓
5	Defer unused CSS	0.3 s ✓
6	Eliminate render-blocking resources	0.24 s ✓

Diagnostics

More information about the performance of your application.

1	Serve static assets with an efficient cache policy	27 resources found ⚠
2	Avoid an excessive DOM size	1,315 nodes ⓘ
3	Minimize Critical Requests Depth	8 chains found

Passed audits

13 audits ✓

INFRASTRUCTURE REVIEW



OVERVIEW

The current infrastructure is in Azure and is managed by the vendor Sitecore. This is a quick check to ensure that things are still properly configured and check several performance related settings.

REVIEW OF AZURE INFRASTRUCTURE

Currently there are two environments; Test and Production and both are managed by the vendor Sitecore through the managed services offering. Attached is a diagram of both environments. Overall, the environments seem to be ok. One thing worth noting is that we found that SOLR is used instead of Azure search (the default) although this is not a requirement. Here we have attached an image, the visio diagram is provided separately.

Current resource tiers

One of the benefits of Azure is the ability to reallocate resources based on need per service. One thing that was noticed was that some of the tiers being used could cause performance issues. Here is a highlighted list of what was found:

- CD (production): P1 V2
 - o 3.5 GB memory
 - o 210 ACU
 - o On the production tier
- CM (production): P2 V2
 - o 7 GB memory
 - o 420 ACU
 - o On the production tier
- CD (test): B2
 - o 3.5 GB
 - o 200 ACU
 - o On the dev/test tier
- CM (test): B3
 - o 7 GB memory
 - o 400 ACU
 - o On the dev/test tier

There are other services that are a part of Sitecore that can be tuned in the future; however, the services mentioned above can be tuned to increase performance. One thing to note is that upping the tier from dev/test to production tier will change outgoing IP addresses and doing so will increase the monthly billing from Azure.

Errors found

There were several issues found in error reporting on both environments. Using Microsoft AI (Application Insights), there are over 1.1 million notifications with about 890k + of them being exceptions (errors caused by configs or custom code). Here is a short list of the more common ones as well as a brief description:

- An unhandled exception occurred. Error while rendering view: '/Views/Website/Layouts/MVC Default.cshtml' (model: 'Sitecore.Mvc.Presentation.RenderingModel, Sitecore.Mvc')...
 - o Several such errors that are around custom code or modules that have been customized

- Sitecore.XConnect.Operations.UpdateDeviceProfileOperation:
Sitecore.XConnect.Operations.EntityOperationException: Operation #0, Conflict, DeviceProfile {94ed84c1-d46d-48e2-b8fb-04e32facb88e}
- Operation #17, Multiple operations exist for the target Contact {48aa78ef-45f1-0000-0000-0565f2379a5e}, KeyBehaviorCache
 - o There are several errors like this one that revolve around the xConnect/xDB system (analytics, personalization, etc)
- Several errors around the SOLR search system were also found
- The analytics system seems to be broken on production since October

Unfortunately, due to the number of exceptions (over 890k over 45 days) we cannot list them all here. These were only examples of related errors that show up frequently.

The real issue with errors in the system like this is that some of the errors are around infrastructure, i.e., xDB and xConnect. This perhaps means that configs and or code was altered in a way that resulted in broken Sitecore functionality. It would need to be determined if this was introduced through Sitecore support, in which they would fix it or if it was from the vendor. The other issue is that when Sitecore has large clusters of errors, it will have a noticeable effect on performance.

DevOps

Currently there is a DevOps process (a method of delivering code/content to environments) that uses the following technologies:

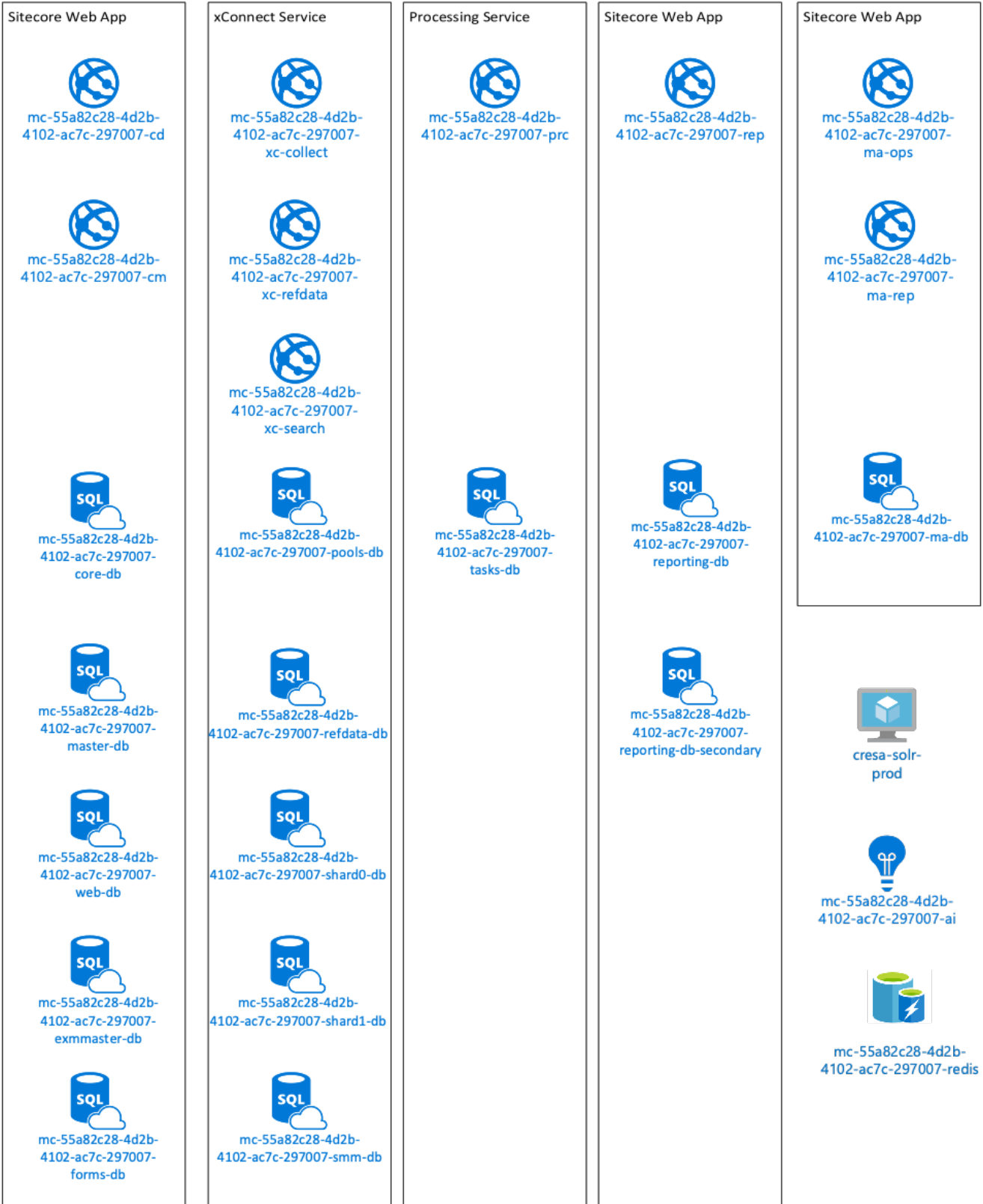
- TDS (Team Development for Sitecore)
- Team City, used for building the Solution
- Octopus deploy which is used for deploying code/content packages

One issue is that this process is currently owned by the vendor and will need to be replicated within the Contoso environment, so that normal DevOps may continue in the environment. Another issue that was brought up is that the development/integration server currently belongs to the current vendor.

Sitecore user management

During the audit process, it was found that there are users that have been created in the system however, there is no documentation that was provided. When looking through the user roles, it did not seem that all of the roles were set up, however, it is difficult to go through in great detail without any documentation around what is needed.

Production Environment – XP Scaled



Test Environment – XP Scaled

Sitecore Web App



aa038ba8-2b4e-4e47-bb45-453240-cd



aa038ba8-2b4e-4e47-bb45-453240-cm



aa038ba8-2b4e-4e47-bb45-453240-core-db



aa038ba8-2b4e-4e47-bb45-453240-master-db



aa038ba8-2b4e-4e47-bb45-453240-web-db



aa038ba8-2b4e-4e47-bb45-453240-exmmaster-db



aa038ba8-2b4e-4e47-bb45-453240-forms-db

xConnect Service



aa038ba8-2b4e-4e47-bb45-453240-xc-collect



aa038ba8-2b4e-4e47-bb45-453240-xc-refdata



aa038ba8-2b4e-4e47-bb45-453240-xc-search



aa038ba8-2b4e-4e47-bb45-453240-pools-db



aa038ba8-2b4e-4e47-bb45-453240-refdata-db



aa038ba8-2b4e-4e47-bb45-453240-shard0-db



aa038ba8-2b4e-4e47-bb45-453240-shard1-db



aa038ba8-2b4e-4e47-bb45-453240-smm-db

Processing Service



aa038ba8-2b4e-4e47-bb45-453240-prc



aa038ba8-2b4e-4e47-bb45-453240-tasks-db

Sitecore Web App



aa038ba8-2b4e-4e47-bb45-453240-rep



aa038ba8-2b4e-4e47-bb45-453240-reporting-db



aa038ba8-2b4e-4e47-bb45-453240-reporting-db-secondary

Sitecore Web App



aa038ba8-2b4e-4e47-bb45-453240-ma-ops



aa038ba8-2b4e-4e47-bb45-453240-ma-rep



aa038ba8-2b4e-4e47-bb45-453240-ma-db



cresa-solr-stg

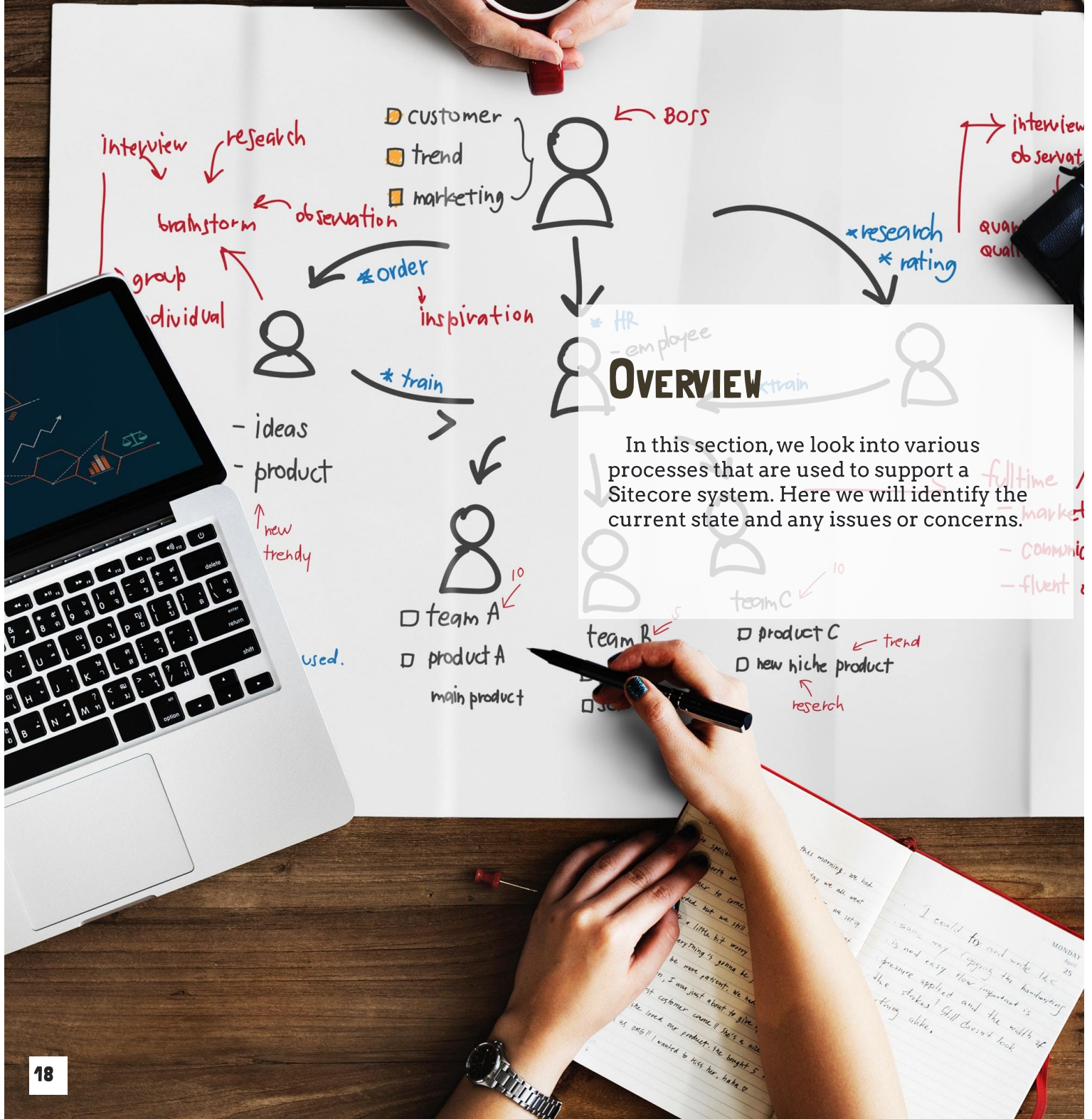


aa038ba8-2b4e-4e47-bb45-453240-ai



aa038ba8-2b4e-4e47-bb45-453240-redis

PROCESSES REVIEW



WORK PROCESSES

Using the term 'work process', we are referring to the method used to manage work performed, whether development, content authoring, training, etc. Through the system review, we found that most of these tools used are from the current vendor and will no longer be accessible after the current vendor leaves. The current systems used for work tracking by the vendor are:

- Jira (ticket tracking, i.e., tasks, stories, etc)
 - o Appear to be following a Agile/Scrum process
- Team City, Octopus and Team Development for Sitecore; used for DevOps
- Bit bucket; used for code version control
 - o Assuming that the vendor was following GitFlow
- More of ad-hoc deployment windows

These tools will have to be replaced as the client will be taking on the ownership of the infrastructure. More on recommendations in the second half of the document.

Deployment processes

The current deployment process works in the following way:

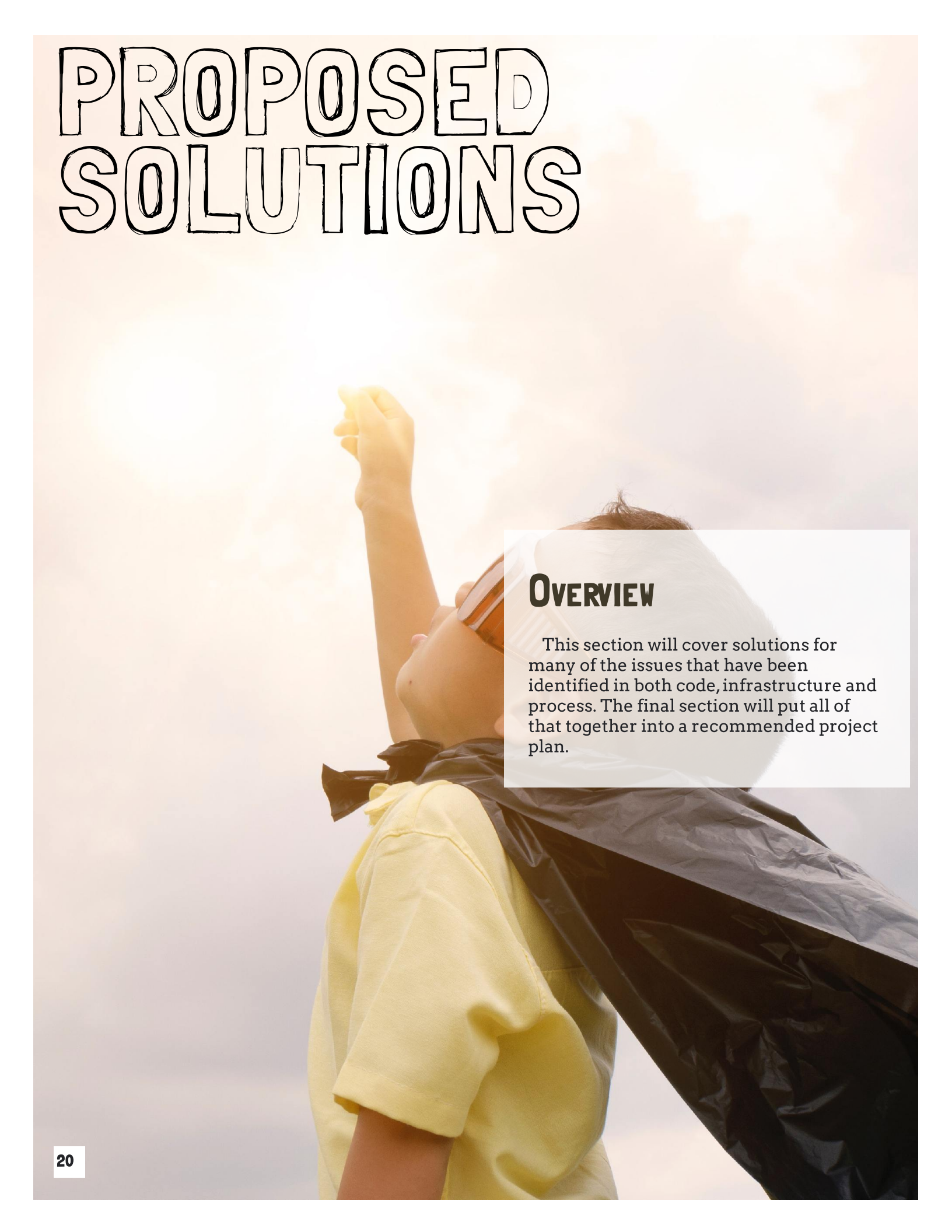
- Developers work code per the sprint
- When complete goes to their dev server for testing using their DevOps process (team city, Octopus and bit bucket)
- Once the release tested, it is then pushed to the test environment for UAT (user acceptance testing), using the DevOps process
- If approved, it goes to production (ad hoc deployment windows), first to the CM server, validated then to the CD server and validated, using the DevOps process

The real issue is that the development/integration server as well as the deployment process is currently owned by the vendor and will need to be replicated. There will also need to be documentation provided.

Sitecore workflows

Generally, best practice in Sitecore is to use workflows. This is especially critical in a large public facing web sites as it requires approval from the client before it is allowed to publish. Currently workflows are not implemented in Production and there appears to be no current documentation around this (functional spec doc, process docs, etc).

PROPOSED SOLUTIONS

A child wearing a yellow t-shirt and a black cape is shown from the chest up, reaching their right arm up towards a bright sun in a cloudy sky. The child is wearing sunglasses and has a joyful expression. The background is a soft, hazy sky with light clouds.

OVERVIEW

This section will cover solutions for many of the issues that have been identified in both code, infrastructure and process. The final section will put all of that together into a recommended project plan.

CODE AND SITECORE INFRASTRUCTURE

Code base and information architecture

There were several issues that were found and already identified previously, here is a quick recap:

- Several empty, unused code files and projects
- Dependency injection is not consistently used throughout
- Use of hard coded field names when using Sitecore API
- Using inefficient Sitecore queries to get items (slow)
- Code base does not closely adhere to helix principals
- Sitecore template structure does not adhere to helix principals, would be difficult to maintain
- Potential to exceed 100 items per parent node
- No caching is being utilized to increase performance
- Large number of errors over 45 days; exceptions were around 860k +

Since several of these issues effect performance, it would be recommended to fix those first then perform some final code clean up, since that should normally be done as a new project is finishing up. Some of the major items that should be fixed first are:

- Caching: Sitecore offers several layers of cache that can quickly increase performance of a site with little effort. It will take a little tweaking since this is an Azure app and has limited resources based on what tier it is in. This will also help to identify the best balance between cost and expected performance. Rendering cache is the most common and the easiest to use
- Sitecore queries: this should be changed as this is generally not best practice and will cause severe performance issues as content increases. Generally, item ids should be used to fetch single items and the Sitecore search API to get many items. This will greatly improve the performance of the app throughout its lifetime
- Hard coded field names: this doesn't affect performance like the other items, however, this practice is discouraged as general practice as it can induce errors that are not caught when developing and can cause issues in production that can be very difficult to identify. Using a ORM (object relational model) like GlassMapper is the recommended best practice
- Exceeding 100 items per parent node: Sitecore recommends against this since it can lead to a large performance penalty. It would be recommended to convert these parent nodes to an Item Bucket, which ensures that this would never be an issue. This will require a little bit of work to ensure that existing functionality is not affected
- Errors: this was concerning since there were so many over a short period of time (1.1 million, of that 860k + were exceptions/code errors). Usually this would be caused by improperly configured Sitecore configuration or custom code. Many of the errors found seemed to be custom code. Even with error handling, when Sitecore has large clusters of errors, it will cause performance issues and can even lead to outages. The errors need to be corrected after identifying if it is infrastructure or custom code. This is because Sitecore maintains the infrastructure and would be able to fix it. Custom code errors would be on the development team to correct

Front end code quality

Some of the issues identified have an effect on performance, while others have more of an impact of usability of the Sitecore tool set. Here is a short recap of those issues:

- No minification/bundling of JS and CSS
- Large image sizes, often not optimized
- Gzip compression is not used
- Fragmented JS code
- Inefficient front end code (JS, CSS)
- Components not optimized for Experience editor

The first items that should be worked on is resolving the performance issues that were identified in the performance scans that were run as well as identified in the code review. These would be:

- **Minification/Bundling:** this is a process where the server will take all CSS files/code and minify (remove whitespace) and bundle (all files in one request). This can often dramatically help with page load times on modern browsers, especially on mobile devices
- **Non-optimal images:** many of the images are too large and use older file formats which can have an effect on performance. This is fairly straight forward to correct
- **Inefficient JS/CSS:** one thing to consider is rearchitecting the front end JS and CSS. On the performance scans, one major issue that was identified was how JS handles images and other items on page load. One strategy that works and is recommended, is to defer images and items below the fold on load time. This is a type of lazy loading concept where only the minimum resources are presented on page load, increasing performance. Although this would be recommended, one would first have to consider time required to implement as this could take some time; resource cost versus performance improvement
- **Optimize components for Experience Editor:** this is important for several reasons. One reason is that if a component is not manageable within Experience Editor, it can interfere with or prevent content authors and marketers from fully using the marketing tools that come with the system, i.e., personalization, AB testing, etc. This is one of the reasons that Sitecore created the helix set of principles so that as they product is constantly updated, it would not interfere with ones implementation of Sitecore since everyone would be following the same set of infrastructure rules.

SEO

Search Engine Optimization was an item that was identified as frequently missing from pages on the main site through the scans that were run. This appears to be an easy fix since a lot of the fields that are needed are already apart of the infrastructure, they just need to be used. In the previous section, time was spent to identify some of the more critical aspects of how to implement SEO properly. Here is a quick recap of some of the identified issues:

- Improper placement of H1, H2, H3, etc, on the page
- Missing Robots.txt
- Not all pages have a title/description
- No redirection, i.e., <https://contoso.com> => <https://www.contoso.com>

All the issues above are simple to correct, with the exception of the placement of the 'H' tags on a page. This would require some rework of HTML, limiting what content authors are allowed to put on a page and may

require some rework of components that are effected. Generally, the recommended handling of SEO in context of Sitecore is the following:

- Require all SEO fields to be filled in before publish (can set this as a restriction)
- Run regular scans of the site using a tool, i.e., google webmaster tools, to check SEO on the site
- Do not allow content authors free use of the rich text field in Sitecore as this allows them to use an 'H' tag improperly. Instead make use of Sitecore renderings and rendering parameters to control/restrict use on a page
- Have an SEO state on a Sitecore workflow where someone is required to scan the page before it can get published to ensure compliance
- Ensure that redirection is handled so that the top domain name always resolves to https://www.
- When setting up an authoring server, ensure that it is behind a firewall and not publicly accessible. Besides the security risk, often clients will give the authoring a cname off of the main domain name of the public site, i.e., cms.myDomain.com and www.myDomain.com . When google crawls both sites, it will of course find duplicate content in both the sub domain and main domain, since whatever is live is also in the authoring server and will have a negative effect on search ranking since google penalizes duplicate content, even if under a different sub domain
- Require all alt tags to be filled in

Overall performance of site

Several scans were run on the site (discussed in previous section) that showed several issues that have an effect on performance of the site. Implementing the suggested fixes on the front end and back end of the code base will increase the overall rating of the site. In general, there are a few recommended best practices for maintaining performance of a Sitecore site:

- Run regular scans of the live site(s) using an agreed upon tool
- Create a performance checklist and require all code that is developed to go through it before going live. This will catch common issues on both the front end and back end of the code base
- Keep a regular eye on the public facing Sitecore apps and their usage. It is very easy in Azure to scale up (more memory/processing) and out (more nodes) based on demand
- Regularly check Sitecore cache levels to ensure that it is sufficient for performance and that the tier the Sitecore app is running in has sufficient memory available
- Develop and document a list of Sitecore coding standards, for example one rule might be not to use Sitecore quires and instead use the Sitecore search API. This will help catch issues as part of the code review/pull request process

DevOps and documentation

One problem that was identified was that although there is a DevOps process, it is owned by the vendor and not contoso. This can be an issue, especially if Contoso wishes to have more control of the entire process or selects new vendors. Another issue that was identified was that there seemed to be a lack of documentation. Normally, a Sitecore instance should have the following documentation for team use:

- Developer documentation that contains coding standards, Git process, documentation that describes the code base
- Deployment/ DevOps documentation that shows the entire workflow from local instance to production

- Content author documentation that contains information on every control the site uses, fields, placement, etc
- Other general process documentation including Sitecore workflows

The proposed technology solution to address this is the following:

- Azure DevOps (formally known as Visual Team Services). This tool is not only free but also has the following features:
 - o Already integrated into the Azure account
 - o Provides a Git repository
 - o Provides work tracking in the agile/scrum manner, i.e., epics, stories, tasks, etc
 - o Is able to run the entire deployment/release management process to any server
 - o Anyone with a MSDN account can use it for free, past 5 users that do not have MSDN license it is \$5 per user, per month (stakeholders are free regardless)
- Sharepoint and Azure DevOps
 - o Azure DevOps has a developer centric documentation wiki that can be used to document their processes, expectations, workflow, etc
 - o Sharepoint for storing all the other documents

In addition to this, it was also mentioned that there was no documentation around user roles and requirements, although it appears to have been setup at least partially. It would be also highly recommended document all of the user roles and how people will be using the Sitecore system per region and responsibility.

In the proposed corrective roadmap section, suggestions on who should create this will be provided.

Sitecore workflows

One very important best practice when working with marketers and content authors is enabling workflows (currently none are used). A basic workflow would be simple to implement and would look like this:

- Draft state: content authors enter in their content after checking it out. Then they check in and submit to the next state
- SEO compliance: an automated step that will ensure that all required fields are filled in, if not it is rejected and sent back to draft
- Approval: someone that is designated to approve content would review. If it is rejected, it goes back to draft, but if it is approved it will go to publish
- Publish: this can be an automatic publish or a timed one. When there is a lot of content being generated, it is generally better to have a timed window in the day to do one publish. The reason being that many publishes at once can eat away and performance of the system. An auto publish would simply publish that content when it is approved

In addition to this, email notifications can be sent upon approval or rejection. Workflows in Sitecore can be as detailed as needed per requirements. For example, there could be several workflows per site, where one workflow is for a specific type of content (i.e., news) and another workflow is just for general brochure-ware content.

One last important detail that will be needed is to identify what basic roles everyone falls into. These roles are generally identified as:

- Contributor: they create content but cannot approve
- Approver: they can approve, but only certain types of items or scope of a site
- Admin: they can approve anything and are used as a backup

PROPOSED ROADMAP



OVERVIEW

In the following section, there is a proposal to fix the issues identified (first two months) and then switch into an ongoing support mechanism that can scale based on need.

First two months

The goal for the first phase of the project would be to correct issues found and implement new processes. The items that would be covered in this time are:

- Correct performance related code issues for the backend (p.24 – 25)
- Correct most of the front-end related performance issues (p. 25)
- Setup DevOps process that deploys to all environments (p. 27 – 28)
- Implement and document developer workflows, i.e., GitFlow, pull requests, code reviews, etc (p. 26)
- Create workflows in Sitecore and implement (p. 28)
- Performance tuning of the Sitecore infrastructure (p. 27)
- Resolution of errors that were discovered, including fixing some configuration issues that seem to have broken (p. 21 – p. 22)
- Fill in the gaps with missing documentation (p. 27 – p. 28)
- Fix any gaps in SEO
 - o We can provide assistance with keywords/descriptions as well
 - o We would mainly be focused on the html portion, i.e., the 'H' tag placements

The overall goal of this is to stabilize the environment, increase performance, fill in missing documentation and bring the environment more under the control of Contoso. This is what the timeline would look like:

N	TASK NAME	START DATE	END DATE	DURATION (WORK DAYS)
1	Correct performance related code issues for the backend	01.07.2019	01.21.2019	10
2	Correct most of the front-end related performance issues	01.21.2019	01.28.2019	5
3	Setup <u>Devops</u> process that deploys to all environments	01.28.2019	01.31.2019	3
4	Implement and document developer workflows	01.31.2019	02.05.2019	3
5	Create workflows in Sitecore and implement	02.05.2019	02.08.2019	3
6	Performance tuning of the Sitecore infrastructure	02.08.2019	02.20.2019	7
7	Resolution of errors and fixing some configuration	02.20.2019	02.28.2019	5
8	Fill in the gaps with missing documentation	02.28.2019	03.12.2019	9
9	Fix any gaps in SEO	03.12.2019	03.20.2019	5

(Start dates are suggested and open to change)

Ongoing support

In this phase we would setup a support mechanism that can be scaled to support future maintenance as well as development, since increased utilization of Sitecore analytics and Salesforce integration is a goal that is on the radar. This team would be laid out like this:

Resource	Name	Responsibility	Hours per week	Rate
Sitecore Developer	Brian	Development, onshore	20	
Sitecore Developer	Yuri	Works back end and <u>front end</u> development	40	
Sitecore Developer	Andrey	Works back end and infrastructure	40	
PM/Architect	Seth	Project management, client support and development	8	

One advantage of this team layout is that we can add more or less with at least a few week's notice, based on need. Seth will remain active to help coordinate with Contoso as well as maintain the team efficiency based on need

USING THE SITECORE TOOLSET



OVERVIEW

Sitecore has a large toolset that allows marketers and content authors to better manage their online experiences. This section will go over some of those tools and how they may be used.

The Sitecore toolset

Sitecore analytics gives a granularized view of the content within. As such, there are several tools which can help better understand and plan for the site.

Experience Analytics: This tool will show several key items. First, it shows traffic through the site. Once goals and campaigns are implemented, it will then show how valuable each visit is to the business. Another useful feature is tracking by device. Once the service is purchased from Sitecore, this report will show every device that is visiting the site(s). Extremely useful as it will help the business make decisions such as where more time should be spent when device testing or perhaps drive the case to spend more time on mobile friendly design. In addition, the tool will give insight into what keywords are being used to search.

Experience Profile: With this tool, it is possible to see a granularized view of every person that is visiting the site, what they do, what they search for and so on. A few things to keep in mind though is to A) get approval to on what information about a user can be stored in regard to Exxon privacy policy B) make any customization changes to the tool depending on what information can or cannot be saved. One valuable use for this tool would be the ability to provide assistance to a user that is calling the help desk, as this will give a view into what they have been doing on the site. A few things to keep in mind though that might require some tweaking is that the information in this report is a combination of the index and SQL database. So this means that out of the box there is a delay in information being refreshed. If worth the time, it may be worth looking into either customizing this tool to be real-time or creating another tool that mimics this tool, instead pulling information from the collection DB/SQL database instead of being index reliant. Also consider integration using the Salesforce connector, which increases the amount of data that can be used when interacting with people online.

Path Analyzer: Using reports generated with this tool give a ball and stick view of how people move through the entire site, the paths they take. This is invaluable as it can show several things, for example, how many clicks it takes to get to valuable content, if users get frustrated and tend to bounce before getting to valuable parts of the site and can be used for funnel development as it helps one to quickly see which traffic patterns tend to funnel the most traffic and where they go to. This tool is even more valuable as goals/value is added to content.

Email Experience Manager: Using this tool, it would be possible to set up automated internal emails and general communications emails. Using the Sitecore service, it would then be trackable through xDB where one can see how many people click on the link and where they went in the site. This would also allow content authors to create emails that can be sent to users. It has further use as marketing automation begins to be used.

Using these tools, it is possible to answer questions like, how many people are coming to the site, how valuable is each visit, what devices are being used the most, what search keywords are being used, what types of people are coming into the site, what do the traffic patterns through the site look like and what is each person doing on the site.

Salesforce Connector:

This tool allows integration of Sitecore into an existing Salesforce environment, allowing marketing teams to use the Sitecore tool to better deliver relevant experiences across all channels. The connector is built upon the Data Exchange Framework, which allows easy integration of data to and from Salesforce, ensuring that as your users interact with the platform online, Sitecore is able to deliver relevant experiences in context to that specific user. In addition, since it is built on the Data Exchange Framework, a developer can create custom contacts to get a more granularized view of site traffic.

Personalization

Given the level of complexity when it comes to personalization, there are several things that should be considered before beginning to create the needed artifacts in the system:

- What are our goals for the site?
- What are some key areas of the site that should be personalized?
- What type of personalization is the best for our goals?

When the goals of personalization and what parts of the site should be personalized are identified, it is then necessary to identify how it should be personalized. Sitecore offers several different ways to personalize a site. One way is to define and create personas/persona keys and assign those throughout the site, which is behavioral based personalization. As a person moves through the site, Sitecore will automatically begin identifying that person based on what personas and persona keys have been set in the system. The other type of personalization uses conditional renderings. Conditional renderings can be used to set several rules and depending on which rule is met, render a specific set of content based on that. Behavioral based personalization takes more time to plan however, it can be used to automatically identify how site visitors are interacting with the site and help give a good idea of how the site is being used. Conditional renderings are good for granularizing content personalization, allowing for out of the box rules or custom ones to be used to switch out content. For example, personas could be created to determine what types of people are visiting the site, i.e., if they are interested in online help/learning or if they are interested in learning more about services that are being provided. This would be invaluable as it would help give Contoso a better idea based on data where more resources should be spent when improving the site.

Where to start with personalization? Based on the initial discovery, a start for personalization would look something like the following, using a service/help related site as an example:

- Profile: Interest
 - o Key: Help Interest
 - o Key: Service Interest
 - o Key: Search Interest
 - o NOTE: There can be as many keys as needed. The point is to start small and as comfort level increases, add more. Some example keys to add would be mobile, desktop, custom development, analytics, etc. Keys are what can be used to assign values to an item/page and what Sitecore uses to resolve what profile/pattern the visitor best fits into
 - o Profile Cards:
 - Just passing through
 - Interest in Services
 - Search power user
 - Online help interest
 - o Pattern Cards: NOTE: names are for demonstration purposes only. The items can be named however needed. There are only three shown; however, as comfort level increases more can be added to better fit needs
 - John the online learner
 - Steve the window shopper
 - Suzan the department manager

The next step would be to assign this profile strategically through the site. For example, when Sitecore identifies someone that is in the 'Search power user' profile and then matches them to the pattern 'Jane the searcher', any components on the site that are conditionally rendering specific content will begin to show content for this type of user. One thing to keep in mind is that Sitecore will only assign one pattern card to a person that is considered the best fit. So as the user begins using search, the component can be customized to a point where it increases the bias towards 'Search power user' profile. Sitecore will then automatically match the closest pattern to that user. Now the component can bias the search results based on what they are most interested in, i.e., services. As use continues, more keys can be added to help granularize the site content to better track and personalize content to site visitors.



High Score Labs is a premier provider of custom .NET programming services for businesses worldwide. Consult our software solutions specialists now to discuss how our .NET development services will help you overcome your software technology challenges. Our featured technology used to drive innovation is the Sitecore xDB system and headless content delivery, in addition to several other technologies like EPI server. We also focus on Martech technologies to empower our partners to reach their full potential. Martech is the area of customer relationship management (CRM) concerned with creating, managing and using digital tools that make it possible for marketers to automate tasks and make data-driven decisions. An important goal of martech is to help marketers find and nurture customers, personalize customer interaction and measure the effectiveness of campaigns.



CHAPIN INDUSTRIES

Florida, USA

www.chapinindustries.com



HIGH SCORE LABS

Dnipro, Ukraine

contact@highscorelab.com

hsl.chapinindustries.com