WHITE PAPER

magnolia®

Building Acme's Composable DXP

Planning and implementing a DXP architecture with Magnolia

Contents

The term "Digital Experience Platform" (DXP) refers to an integrated technology stack that helps enterprises create and optimize engaging digital customer experiences throughout the customer journey. While the idea behind a DXP isn't new, the term itself is only gaining popularity now.

Many old DXPs rely on monolithic software suites that pose a barrier to change, preventing companies to adapt to customer expectations at the speed that is required to make their mark.

According to Gartner, modernizing the DXP stack "means abandoning monolithic technologies that cannot meet market demands. To future-proof the stack, a composable DXP must be adopted as a way to deliver composable user experiences."[1]

Because of the flexibility of "composability", the complexity of composable DXPs can seem hard to grasp. And while no two composable DXPs are identical, you can use a common approach to reduce complexity and build a DXP that meets your needs uniquely.

This white paper will demonstrate how to plan and implement a composable DXP using the example of Acme, a fictitious retailer. You can apply the same practices to your use case.

The composable DXP

The idea of composability is based on the combination and integration of a set of technology components. Each component augments the DXP by providing certain capabilities for creating and managing digital experiences.

This approach allows companies to determine the breadth and capabilities of their DXP flexibly. While some components are integral to a DXP, others are optional and can be added incrementally. Integral components include content and experience management, and delivery. Optional components complement the DXP and include personalization and optimization.

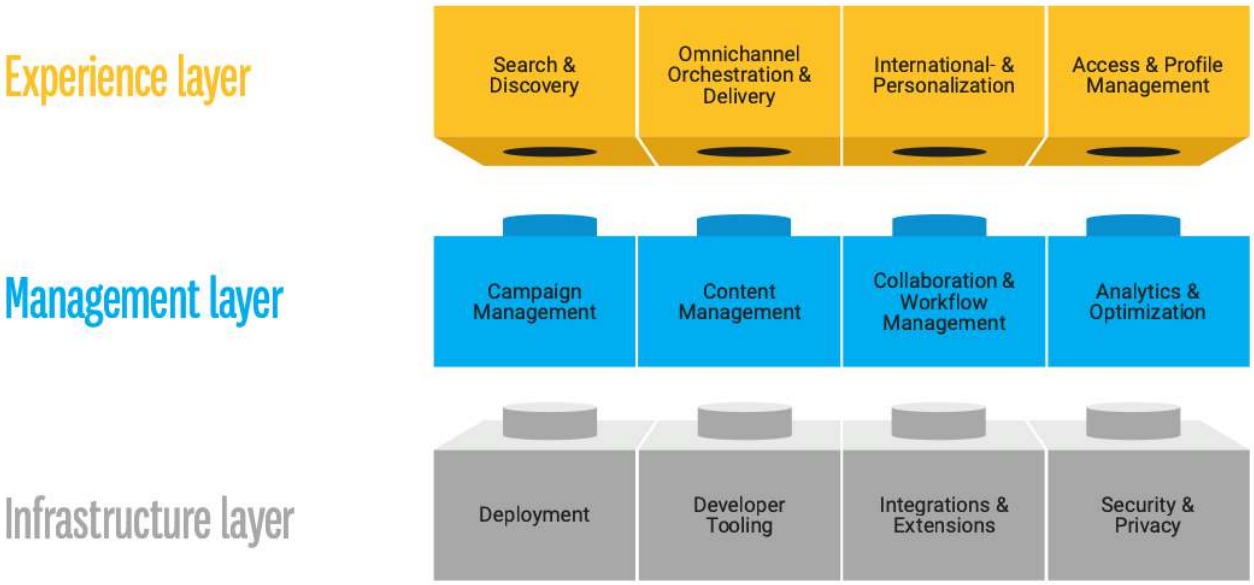
Another benefit of composability is the ability to adapt to changing requirements. Individual components can be replaced by technologies that are a better fit at any time.

[1] Gartner, Magic Quadrant for Digital Experience Platforms, 26 January 2021

A framework for DXP planning

Without a structured approach, planning a composable DXP can easily become an overwhelming exercise, either leading to the omission of required capabilities or the abandonment of the composable DXP in favor of an inflexible packaged DXP solution. In both cases, you risk finding yourself in a situation that is less than ideal, limiting your ability to deliver.

To help you overcome its complexity, we break the DXP down into 3 layers; each layer has 4 building blocks.



The framework identifies components to consider when building your DXP, and helps you assess which components already exist in your current technology stack, allowing you to plan for their integration. It might also highlight components that had not been considered, allowing you to close any gaps.

While the framework is not exhaustive, it covers the vast majority of use cases and provides almost unlimited flexibility through integration. Using a framework like this, helps you to set yourself up for DXP success from the start.

Acme's use case

We are going to use Acme, a fictitious retailer, as an example to demonstrate how to build a composable DXP with Magnolia. Their use case is similar to a lot of use cases that we come across in our daily work with our customers and prospects.

The status quo

Acme is a clothing retailer that conducts its main business online, but also owns brick-and-mortar stores. Acme's challenge is to build a new platform to seamlessly deliver engaging customer experiences across various digital channels at speed.

Its current content management system (CMS) was identified as a limiting factor because it requires developers to work with its templates rather than integrating with a front-end framework like React, which Acme developers have been requesting for a while. Re-using content on channels beyond the website has also proven difficult.

Acme is now looking for a headless content management system (CMS) that sits at the center of its platform integrating its existing DX ecosystem, most importantly its commerce solution.

The CTO wants to ensure that the new platform is flexible and can integrate with new technologies easily to avoid getting stuck in the future.

The company has been migrating to the cloud over the last couple of years and would like to run the platform on state-of-the-art cloud architecture.

Acme also has the following technical requirements for their new Digital Experience Platform:

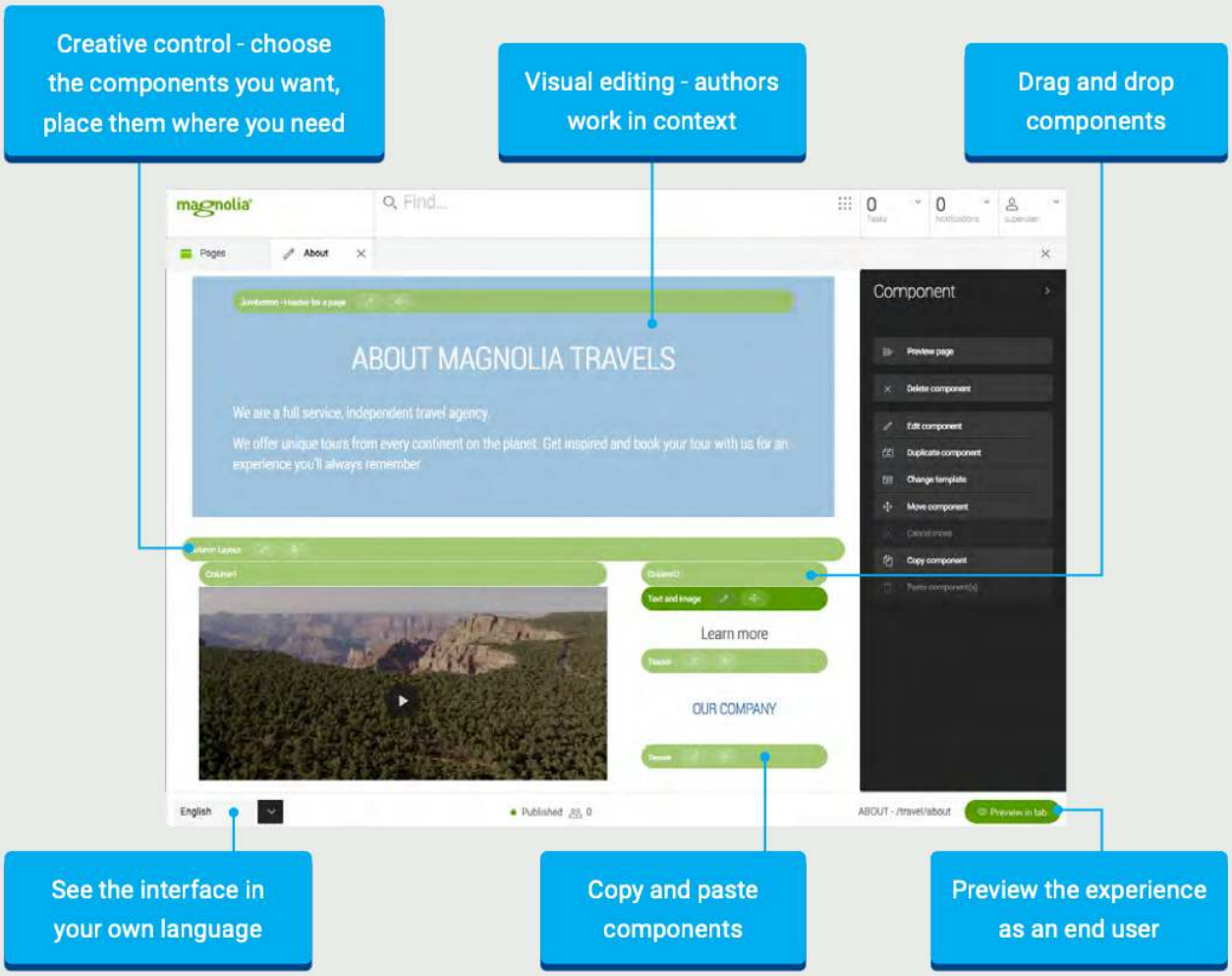
Visual editor to manage experiences	Our content authors are used to working in a WYSIWYG editor. The new platform should allow them to do that, while having a modern look and being comfortable to use.
Headless content delivery with the Jamstack	Our front-end developers want to develop digital experiences in React. We want to enable this without impacting our content authors. We would also like to improve our website's performance using Netlify.
Fashion blog	We want to host a fashion blog featuring topical products.

Product reviews	We developed a custom solution for product reviews and would like to leverage the solution on the product pages. The solution offers APIs to create and query product reviews.
Digital signage in stores	We would like to populate the screens in our stores with content from the CMS. This content should be easily created in the CMS and dynamically displayed.
Access though SSO	We want our content authors to log into the platform with their existing Acme account.
commercetools	We manage our products and shop accounts in commercetools and want to continue to do so. The platform has to integrate with commercetools and should allow our e-commerce team to easily use product data in our digital experiences.
Salesforce Marketing Cloud	We store customer profiles and newsletter subscriptions in Salesforce Marketing Cloud. We would like to integrate marketing forms on our website and store all customer data in Salesforce.
Google Analytics	Web analytics are key to our organization. We report on our key metrics on a regular basis and expect our teams to leverage data when creating experiences.
Cloudinary	We use a lot of images on our website and in our store. We implemented Cloudinary as our Digital Asset Management (DAM) system and the new platform has to integrate with it.

Content Management

Magnolia provides the Content Management building block via its core product, offering traditional web content management as well as headless content delivery. This approach allows developers to consume presentation-independent content from a central repository via API while creating the front end in React.

When working with Magnolia in a headless scenario, its Visual SPA Editor is a game changer for author/developer collaboration. Despite the decoupled front end, content authors can create content and design experiences using a WYSIWYG editor in Magnolia's AdminCentral. For Acme this means that developers work in React, while authors work in the CMS.



The following paragraphs explain the concept of the Visual SPA Editor and how it is able to render and edit the front end. For a detailed step-by-step guide, explaining how to set up a headless project using the Visual SPA Editor, please consult Magnolia's headless documentation. The examples below are part of the minimal-headless-spa-demos project within this documentation.

Similar to Magnolia's page editor, the Visual SPA Editor relies on templates and dialog definitions. To enable authors to manage a single-page application or another custom front end, the front end has to provide Magnolia with a list of page components and their editable properties. Instead of using freemarker to render a page, it uses its dedicated SPA renderer. The SPA template file and renderer are defined in the page template.

react-minimal-lm/templates/pages/basic.yaml (Source)

```
title: 'React: Basic'
templateScript: /react-minimal-lm/webresources/build/index.html
dialog: spa-lm:pages/basic
renderType: spa
class: info.magnolia.rendering.spa.renderer.SpaRenderableDefinition

areas:
  main:
    title: Main Area
    availableComponents:
      Headline:
        id: spa-lm:components/headline
      Image:
        id: spa-lm:components/image
      Paragraph:
        id: spa-lm:components/paragraph
      Expander:
        id: spa-lm:components/expander
      List:
        id: spa-lm:components/list

  extras:
    title: Extras Area
    availableComponents:
      Headline:
        id: spa-lm:components/headline
      Paragraph:
        id: spa-lm:components/paragraph
      List:
        id: spa-lm:components/list
```

Each page or component has a dialog definition, configuring the fields authors can edit. The YAML configuration is the same for traditional and headless delivery. The below example shows how to configure the “title” field.

spa-lm/dialogs/pages/basic.yaml (Source)

```yaml
label: Page Properties
form:
  properties:
    title:
      label: Title
      $type: textField
      i18n: true
```

The page’s actual code is created in the React front end, using the properties defined in the component dialog. Magnolia’s React library processes the template and its content to render the page, making it available for visual editing by authors.

react-minimal/src/pages/Basic.js (Source)

```jsx
import React from 'react';
import { EditableArea } from '@magnolia/react-editor';

const Basic = props => {
  const { main, extras, title } = props;

  return (
    <div className="Basic">
      <div className="hint">[Basic Page]</div>
      <h1>{title || props.metadata['@name']}</h1>

      <main>
        <div className="hint">[Main Area]</div>
        {main && <EditableArea className="Area" content={main} />}
      </main>

      <div className="Extras" >
        <div className="hint">[Sercondary Area]</div>
        {extras && <EditableArea className="Area" content={extras} />}
        {/* <button>Contact</button> */}
      </div>
    </div>
  )
};

export default Basic;
```

A configuration file in the React application maps React components to Magnolia components.

react-minimal/src/magnolia.config.js (Source)

```
import Basic from './pages/Basic';
import Contact from './pages/Contact';
import Headline from './components/Headline';
import Image from './components/Image';
import Paragraph from './components/Paragraph';
import Expander from './components/Expander';
import List from './components/List';
import Item from './components/Item';

const config = {
    'componentMappings':{
        'react-minimal-lm:pages/basic': Basic,
        'react-minimal-lm:pages/contact': Contact,

        'spa-lm:components/headline': Headline,
        'spa-lm:components/image': Image,
        'spa-lm:components/paragraph': Paragraph,
        'spa-lm:components/expander': Expander,
        'spa-lm:components/list': List,
        'spa-lm:components/listItem': Item
    }
};

export default config;
```

Based on the above configuration, Acme's content authors can use the Visual SPA Editor to manage experiences delivered from its React front end.

Another capability that improves how content is managed is content modelling using Content Types and Content Apps. Authors can easily manage content in a structured way and make it available via Magnolia's REST or GraphQL APIs. Acme can use it to create a list of stores and, for example, build a store finder on top.

Acme's developers can create Content Types using low-code practices in Magnolia's Light Development. This file-based configuration allows developers to make changes without backend development or a restart, and enables the use of tools such as Git for configuration management. Light Development also offers efficient ways to create content dialogs, API endpoints, and page templates, reducing development effort and shortening time-to-market.

The following example shows how Acme can create a Content App to manage stores, their opening hours, as well as job openings.

Acme first creates three Content Types configurations:

<light-module-name>/contentTypes/store.yaml

```yaml
datasource:
  workspace: stores
  autoCreate: true

model:
  nodeType: store
  properties:
    - name: address
    - name: zipcode
    - name: city
    - name: state
    - name: country
    - name: services
      type: service
      multiple: true
    - name: jobs
      type: reference:job
      multiple: true
    - name: openHours
      type: reference:openhours
  subModels:
    - name: service
      properties:
        - name: description
        - name: openHours
          type: reference:openhours
```

<light-module-name>/contentTypes/openhours.yaml

```yaml
datasource:
  workspace: store-open-hours
  autoCreate: true

model:
  nodeType: store-open-hour
  properties:
    - name: weekDays
    - name: weekDaysHours
    - name: saturday
    - name: saturdayHours
    - name: sunday
    - name: sundayHours
    - name: additionalInfo
```

<light-module-name>/contentTypes/job.yaml

```yaml
datasource:
  workspace: store-jobs
  autoCreate: true

model:
  nodeType: store-job
```

```
  properties:
    - name: description
      type: richText
    - name: start
      type: Date
    - name: isPublic
      type: Boolean
```

Acme then creates a Content App to allow content authors to manage data of each Content Type easily:

<light-module-name>/apps/stores-app.yaml

```
!content-type:store
name: stores-app
label: Stores
```

<light-module-name>/apps/open-hours-app.yaml

```
!content-type:openhours
name: open-hours-app
label: Open hours
```

<light-module-name>/apps/jobs-app.yaml

```
!content-type:job
name: jobs-app
label: Jobs
```

To enable Acme to consume store data from their React front end via REST API, they create an endpoint:

<light-module-name>/restEndpoints/stores.yaml

```
class: info.magnolia.rest.delivery.jcr.v2.JcrDeliveryEndpointDefinition
workspace: stores
depth: 10
nodeTypes:
 - store

references:
  - name: jobsTypeReference
    propertyName: jobs
    referenceResolver:
      class: info.magnolia.rest.reference.jcr.JcrReferenceResolverDefinition
      targetWorkspace: store-jobs
  - name: openHoursTypeReference
    propertyName: openHours
    referenceResolver:
      class: info.magnolia.rest.reference.jcr.JcrReferenceResolverDefinition
      targetWorkspace: s-opetoren-hours
```

The REST endpoint will respond on http://acme.com/.rest/stores/.

For their fashion blog, Acme can use Magnolia's Stories App to manage content. It provides an interface to combine fixed metadata, such as a headline, a date, and the author's bio, with a variable number of content blocks including text, images, and products. The configuration of a Stories App is very similar to the configuration of a Content App.

To enable content authors and marketing teams, Magnolia also offers capabilities known from traditional web content management including multi-site management, translations, and publication workflows, allowing authors to create and manage complex experiences efficiently.

Omnichannel Orchestration & Delivery

Acme has decided to deploy its website through Netlify's globally distributed network with automated pre-rendering for maximum performance. Magnolia's extension for Netlify allows Acme's content authors to build Netlify sites directly from Magnolia's AdminCentral.

The extension configures Netlify sites for deployment and adds a field "Build on Netlify" to the publication dialog allowing authors to choose which Netlify sites to build. When a publication gets approved, Magnolia instantly triggers a build for each selected Netlify site.

The extension is available in the Magnolia Marketplace and easily installed as a Maven module:

```
<dependency>
  <groupId>info.magnolia.cdn</groupId>
  <artifactId>magnolia-netlify-integration</artifactId>
  <version>${netlifyIntegrationVersion}</version>
</dependency>
```

The module configures a REST client to trigger a build via the Netlify API. The API token can either be set in the Maven module itself or in a separate module.

/decorations/netlify-integration/config/config.yaml

```
apiToken: your_api_token
```

You can find out more about Magnolia's partnership with Netlify in our blog Netlify and Magnolia partner to deliver best-practice headless CMS frontends.

Besides the web channel, Acme has several screens in its brick-and-mortar stores to display promotional content. Magnolia's Digital Signage App allows Acme to create dedicated content for these screens or to reuse any existing content such as images and videos. Authors can conveniently manage content for this channel in AdminCentral and rely on the usual translation and publication workflows.

This extension is available as a Maven module.

Integrations & Extensions

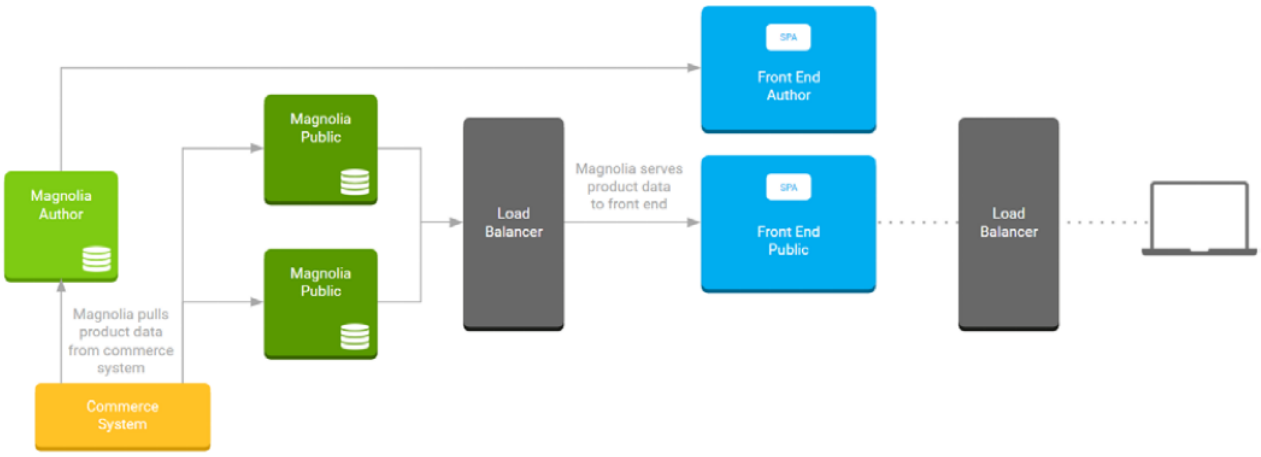
Commerce integration

Acme has an established ecosystem of digital experience (DX) technology that has to be integrated in its DXP. With Acme being a retailer, its commerce solutions, commercetools, is one of the most critical components for its business.

To integrate content and commerce in one platform, Magnolia provides a Connector Pack for Commerce. The out-of-the-box integration enables Acme's developers to make external product data available in Magnolia.

While the actual data is stored and managed in commercetools, content authors can seamlessly use product information on product and category pages, or teasers, as if the content was native to Magnolia. Authors can conveniently select specific products from the component dialog filtered by category.

As Acme has chosen a headless architecture, its front-end developers can consume product content through Magnolia's REST API to display the product data in the front end.



As described in our documentation, Maven is the easiest way to install the module.

```
<dependency>
  <groupId>info.magnolia.ecommerce</groupId>
  <artifactId>magnolia-ecommerce-commercetools-connector</artifactId>
  <version>${ecommerce.version}</version>
</dependency>
<dependency>
  <groupId>info.magnolia.ecommerce</groupId>
  <artifactId>magnolia-ecommerce-templating</artifactId>
  <version>${ecommerce.version}</version>
</dependency>
<dependency>
  <groupId>info.magnolia.ecommerce</groupId>
  <artifactId>magnolia-ecommerce-ui</artifactId>
  <version>${ecommerce.version}</version>
</dependency>
```

Magnolia's connector for commercetools makes the configuration simple:

```
class: info.magnolia.ecommerce.common.EcommerceDefinition
type: commercetools
enabled: true
implementation:
  products:
    all: info.magnolia.ecommerce.commercetools.products.All
    byId: info.magnolia.ecommerce.commercetools.products.ById
    byCategoryId: info.magnolia.ecommerce.commercetools.products.ByCategoryId
    searchByText: info.magnolia.ecommerce.commercetools.products.SearchByText
  categories:
    all: info.magnolia.ecommerce.commercetools.categories.All
    byId: info.magnolia.ecommerce.commercetools.categories.ById
    byParentCategoryId: info.magnolia.ecommerce.commercetools.categories.ByParentCategoryId
    byProductId: info.magnolia.ecommerce.commercetools.categories.ByProductId
connections:
  connectionName:
    enabled: false
    authUrl: https://demo.commercetools.com
    parameters:
      clientId: client_id
      clientSecret: client_secret_or_path_to_password_manager
      apiUrl: https://demo.commercetools.com
      projectKey: project_key
```

Integration of external APIs

Acme would like to leverage a custom-built solution for product reviews to enrich its product pages. Customers can review and rate products, and see other customers' reviews, too.

The review system offers APIs to write and read product reviews. Magnolia's REST client, Multisource, allows Acme to easily read product reviews from this third-party system via API using YAML configuration.

Magnolia's declarative-rest-demo project provides the following example, that Acme can copy:

```
baseUrl: http://openlibrary.org/api
restCalls:
  searchByIsbn:
    method: get
    entityClass: java.lang.String
    path: /volumes/brief/isbn/{isbn}.json
    defaultValues:
      isbn: 978-3-16-148410-0
```

To write data to the review system, Acme needs a custom Java-based REST endpoint.

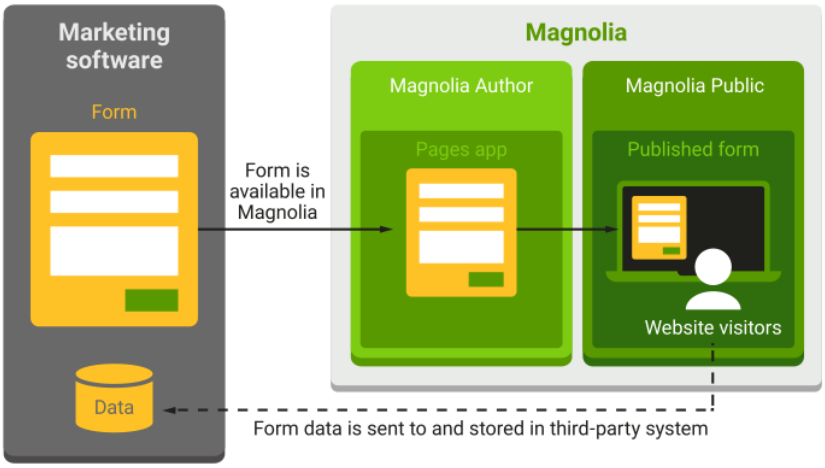
Marketing integration

Salesforce Marketing Cloud is Acme's CRM and marketing automation system. It is the system of record for all customer data. All new customer registrations and newsletter sign-ups are to be stored in Salesforce Marketing Cloud through an integration with its marketing forms.

Using the Connector Pack for Marketing Automation, Magnolia retrieves marketing forms and their fields from Salesforce via API.

Authors can choose a form in Magnolia and arrange its fields in any order. To place an individual form field on a page, authors select it from a simple dialog. The style of the form is defined by the Magnolia theme.

When the form is submitted, its data is directly written to the CRM.



As described in our documentation, Maven is the easiest way to install the module.

```
<dependency>
  <groupId>info.magnolia.marketingautomation</groupId>
  <artifactId>magnolia-marketing-automation-ui</artifactId>
  <version>${marketing-automation.version}</version>
</dependency>
<dependency>
  <groupId>info.magnolia.marketingautomation</groupId>
  <artifactId>magnolia-marketing-automation-salesforce-connector</artifactId>
  <version>${marketing-automation.version}</version>
</dependency>
```

The configuration using Magnolia's connector for Salesforce is simple:

```
class: info.magnolia.mkautomation.definition.DefaultMarketingFormDefinition
enabled: true
type: salesforce
implementation:
  forms:
    all: info.magnolia.mkautomation.salesforce.forms.All
    byId: info.magnolia.mkautomation.salesforce.forms.ById
    byName: info.magnolia.mkautomation.salesforce.forms.ByName
    submitBy: info.magnolia.mkautomation.salesforce.forms.SubmitBy
  leads:
    searchBy: info.magnolia.mkautomation.salesforce.leads.SearchByQuery
connection:
  baseUrl: https://mianlien.my.salesforce.com/services
  authUrl: https://login.salesforce.com/services/oauth2/token
  credentials:
    username: email@email.com
    password: password_or_path_to_password_manager
  clientId:
3MVG96_7YM2sI9wT5awWympuHDV97wt6AsbvT1ILMqMWXfUgSfB0e4SztxStXnyjskXpQC571mKsUx2qddJiw
  clientSecret: client_secret_or_path_to_password_manager
selectedForms:
  - Lead
  - Account
  - Contact
formSelectorId: marketing-automation-ui:fieldsSelectionForm
```

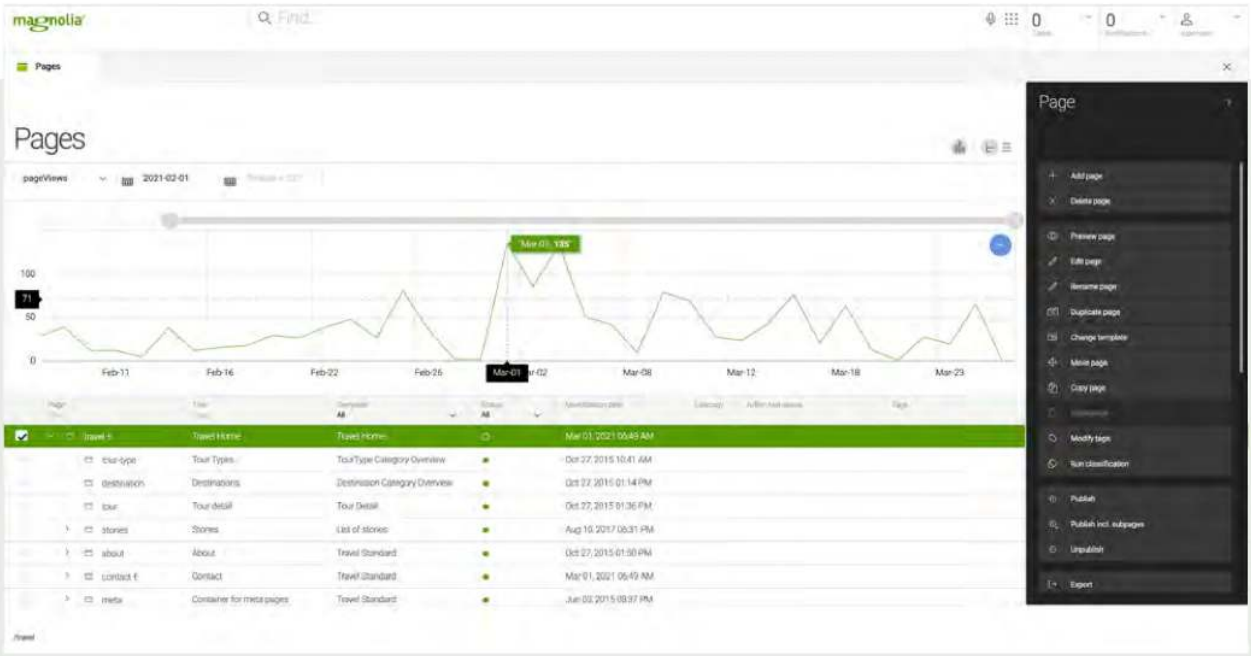
Analytics integration

The integration of analytics systems in Magnolia operates in two directions. Magnolia's Marketing Tags App is based on JavaScript tags and enables Acme to track data in Google Analytics, such as enhanced visitor and usage information.

By using the Connector Pack for Analytics Acme can display analytics data in Magnolia, saving authors time and effort to check this data in another system.

The Connector Pack for Analytics provides the Analytics App to create custom dashboards and charts, called widgets. Users can choose between several grid layouts and use the convenient drag-and-drop functionality to place their widgets. They can leverage commonly-used metrics that Magnolia provides out of the box, or create advanced custom widgets including a preview.

By loading analytics data for content on a specific page, the Analytics Connector Pack also enables authors to check how individual pages perform. Furthermore, authors can filter data by typing a date range or by selecting a specific range in the dynamic, clickable diagrams.



As described in our documentation, Maven is the easiest way to install the module.

```
<dependency>
  <groupId>info.magnolia.analytics</groupId>
  <artifactId>magnolia-analytics-ui</artifactId>
  <version>${analytics.version}</version>
</dependency>
<dependency>
  <groupId>info.magnolia.analytics</groupId>
  <artifactId>magnolia-analytics-amcharts</artifactId>
  <version>${analytics.version}</version>
</dependency>
<dependency>
  <groupId>info.magnolia.analytics</groupId>
  <artifactId>magnolia-analytics-google-connector</artifactId>
  <version>${analytics.version}</version>
</dependency>
<dependency>
  <groupId>info.magnolia.analytics</groupId>
  <artifactId>magnolia-analytics-adobe-connector</artifactId>
  <version>${analytics.version}</version>
</dependency>
```

Magnolia's connector for Google Analytics makes the configuration simple:

```
dataSuppliers:
  googleSupplier:
    class: info.magnolia.analytics.google.datasource.GoogleDataSupplier
    credentials:
      applicationName: Magnolia Analytics
      serviceAccountJsonPath: /google-analytics/analytics/google/private_key.json
    parameters:
      viewId: 191451330
      startDate: 7DaysAgo
      endDate: today
      dimensionName: ga:date
      metricExpression: ga:sessions
      metricAlias: sessions
```

DAM integration

Magnolia offers digital asset management (DAM) natively or through the Connector Pack for DAM. Acme uses Cloudinary and wants to integrate this system into its DXP. The integration will allow content authors to use images, videos and other digital assets from Cloudinary in Magnolia.

As described in our documentation, Maven is the easiest way to install the module.

```
<dependency>
  <groupId>info.magnolia.external.dam</groupId>
  <artifactId>magnolia-external-dam</artifactId>
  <version>{externalDamVersion}</version>
</dependency>
<dependency>
  <groupId>info.magnolia.external.dam</groupId>
  <artifactId>magnolia-external-dam-cloudinary</artifactId>
  <version>{cloudinaryVersion}</version>
</dependency>
```

The setup of the Cloudinary extension requires a few connection parameters, described in our documentation.

Deployment

Many enterprises are seeking to deploy their applications to the cloud, and so does Acme. Besides using Magnolia's Light Modules for low-code configuration, Acme wants to be able to create custom Java modules going forward and needs to ensure a reliable build pipeline consisting of a development, integration, and production environment. New features have to pass all stages of the pipeline before going into production.

Rather than self-hosting Magnolia, Acme decides to deploy Magnolia's PaaS solution, reducing the effort of managing infrastructure while maintaining maximum flexibility.

The solution can be managed in Magnolia's Cloud Cockpit, giving Acme full control over the environment. It also allows developers to copy content from the production to the integration environment for testing.

Security & Privacy

Magnolia's Single Sign-On (SSO) module offers an integration with identity providers such as Azure Active Directory (Azure AD), Auth0, AWS Cognito, Keycloak, and Okta, allowing Acme's employees to use their company account to log into Magnolia's AdminCentral automatically.

To set up SSO, Magnolia has to be configured as a client with the authorization service. To identify it as a valid client, Magnolia receives a client ID and a client secret, sometimes called app ID and app secret. Acme's IT team also has to define one or more callback URLs, also known as redirect URIs.

Our blog post "External User Management and SSO with Magnolia" explains SSO with Magnolia in more detail.

Summary

Acme reached the limit of what is possible with their current CMS and DX ecosystem. They defined technical requirements to achieve their DX vision while also providing flexibility to evolve their DXP in the future.

We used the DXP framework and mapped the required capabilities to building blocks. Acme's DXP will initially consist of these building blocks:

- Content Management
- Omnichannel Orchestration & Delivery
- Integrations & Extensions
- Deployment
- Security & Privacy

Acme's example is meant to demonstrate that this approach can be applied to various use cases, allowing you to build a flexible platform that meets your needs today and in the future.

magnolia®

Get in touch

To learn how Magnolia can help you launch great digital experiences faster, contact us at:

Switzerland - Headquarters
+41 61 228 90 00
contact@magnolia-cms.com

United States
(305) 267-3033
contact-us@magnolia-cms.com

Czech Republic
+420 571 118 715
contact@magnolia-cms.com

Spain
+34 662 63 43 36
contact-es@magnolia-cms.com

United Kingdom
+ 44 203 741 8083
contact-uk@magnolia-cms.com

Vietnam
+84 28-3810-6465
contact-apac@magnolia-cms.com

Singapore
+65 64 30 67 78
contact-apac@magnolia-cms.com

China
+86 2133 280 628
contact-apac@magnolia-cms.com

www.magnolia-cms.com

April 2021