BROUGHT TO YOU IN PARTNERSHIP WITH

Getting Started With Java-Based CMS

WRITTEN BY SERGE HUBER
CHIEF TECHNOLOGY OFFICER (CTO) AND CO-FOUNDER AT JAHIA

CONTENTS

Content Management Systems

A content management system (CMS) is a software application that facilitates the creation, management (creation, editing, versioning, workflow, etc.), and publishing of content. Historically speaking, content management systems were invented so that non-technical content creators could quickly collaborate to produce and publish content without worrying about the technological complexities of displaying that content in a web page or application. Today, the CMS is just as important to developers as it is to non-technical content managers. The line between what is code and what isn't is more blurred than ever before. Today, almost every application has content in it. Developers are looking for solutions that allow them to craft and deploy content-enabled apps, websites, and other digital experiences quickly without having to reinvent the wheel.

Apps That Benefit from Content Management

As previously mentioned, nearly every application today has content in it. Here are a few classes of applications that get a huge benefit when the content they require is supplied by a CMS.

1. Websites and portals.
2. Native mobile and single-page apps.
3. Virtual and augmented reality experiences.
4. Kiosk-based applications.
5. Learning management applications.
6. Internal and external business applications (e.g., banking and insurance).
7. Retail and customer loyalty applications.
8. Health and fitness devices, applications, and support systems.
9. Game and entertainment experiences.
10. Many others; nearly every kind of app can benefit from externalized content.

The Pillars of CMS

At a very high level, there are four concepts that serve as pillars in the content management world.

CONTENT MODEL

A content model documents all content types associated with a brand and defines the relationship between those content types. With a content model, a content manager can visualize the purpose of each piece of content, enabling the organization of a website's content ecosystem. This includes multiple content types like web pages, blog posts, PDFs, images, and unpublished documents. Content modeling is the act of defining the content model. Content modeling is typically done as an upfront activity when setting up a content management system but may require modification over time.

CONTENT

Content is text along with images, video, and other media. Ideally, content is captured in a presentation-less format.

jahia

What Is The Right Web CMS For Your Organization?

The Forrester Now Tech Q4 Web CMS Overview Report

DOWNLOAD NOW!

Stack-up & Stand Out

Unite Content, Data, and Applications into
Your Stack to deliver a better customer experience.

- One hub for your content and customer data
- Modernize legacy applications to deliver personalized online experiences
- Flexible and continuous cross-channel management
- Open and modular to fit your needs
- Built to integrate with your best-in-class Martech stack

Visit us at www.jahia.com
Contact us at marketing@jahia.com

GENEVA | PARIS | LONDON | BOSTON | TORONTO

RENDERER OR PROCESSOR

It's common to want to apply some business logic to the content prior to providing it to the presentation, or "view layer." This is the purpose of a renderer, which is coe that can take content based on the model and prepare it for the view/presentation by applying business rules.

PRESENTATION

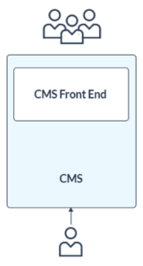
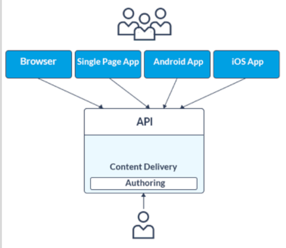
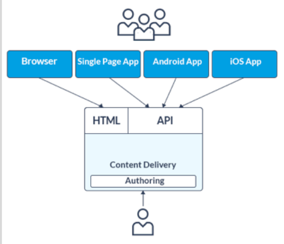
Presentation is the overarching term for the technology that presents the content. This technology may be a template, or it may be a remote client such as a smartphone or a single-page application.

Foundational CMS Capabilities

Between commercial and open-source CMS solutions, there are over 1,000 platforms available on the market. With so many CMS platforms, the possible feature set is huge. There are a lot of important capabilities, like analytics and marketing automation, that won't matter much if you don't get the basics right. Here's a list of foundational capabilities to evaluate first when choosing a CMS:

Ease of use	A CMS must make it easy for non-technical individuals to create, manage, and publish content or it won't properly provide the value it's designed to deliver.
Content modeling	Content modeling is a fundamental capability of a CMS. Make sure you can easily define the structure of your content types and the associations (relationships) between them.
Templating	Templates allow you to define the presentation of your content independent of the content itself. Updating a single template changes the presentation for all the content objects that use that template.
Search and query	Today's use cases require dynamic content. A CMS must provide powerful query and search capabilities to the content delivery components of the system.
Publishing	Publishing is a category of capabilities that defines how a CMS makes content available to users. There are two main approaches to publishing: • **Push**: Content is "transferred" from authoring systems to delivery systems to be leveraged by content consumers. • **Pull**: Content is requested from the delivery components of the CMS by content consumers. Almost all CMS technologies support the pull model for publishing. Requesting an HTML page is an example of this model. Push-based publishing is more commonly supported by decoupled CMS platforms (and will be described later).
Library services	Library services are capabilities that support the management of your content. These services include: • Content locking (check in/check out) • Versioning • Activity auditing
Workflow	Workflow enables your team to collaborate by breaking down the content creation and review process into steps/tasks that can be assigned to users or groups of users for completion.

CMS Architectures

Coupled / Monolithic	In a coupled system, the underlying store for your content serves both authoring and delivery. In a coupled system, the process of making content live is typically a matter of setting a flag in the database.	
Headless	Headless CMS technology provides content to the presentation tier (or content consumer) as a service, typically via a RESTful interface in JSON or XML format. This is known as Content-as-a-Service (CaaS). A headless CMS can either be coupled or decoupled, and should support headless/CaaS-based content delivery regardless of whether it is coupled or decoupled.	 Browser Single Page App Android App iOS App API Content Delivery Authoring
Decoupled /Head Optional	A decoupled system puts authoring and delivery in separate applications and potentially on separate infrastructure. In a decoupled system, the process of making content live is done through a publishing mechanism where content is pushed from the authoring platform (and underlying content repository) to a content delivery infrastructure.	 Browser Single Page App Android App iOS App HTML API Content Delivery Authoring

IS A HEADLESS CMS A DECOUPLED CMS?

While headless CMS architectures do "decouple" content and presentation, they do not dictate anything about the publishing capabilities of the CMS. While decoupling content and presentation is certainly one of the most important things you can do architecturally, as we can see above, it's not the only major architecture decision you need to consider. Therefore, it is important to maintain the traditional use of the term "decoupled CMS" as it relates to the separation of authoring and delivery capabilities.

COUPLED VS. DECOUPLED: MAKING A CHOICE

So which approach is the right architecture? The reality is that there is no single right or wrong answer. The answer depends on the alignment with your requirements, your business processes, and your business goals.

Our analysis reveals that a coupled architecture can work well for web apps, mobile apps, and other content-backed digital experiences that need to be set up and put online in short order, and that do not need to scale quickly or publish content beyond the website itself. On the other hand, we see that a decoupled architecture is ideal for websites/content back ends that require high levels of availability and performance, need a lot of tailored functionality, must be integrated with third-party business systems, and must publish to one or more digital channels beyond the website itself.

The Importance of Separating Code from Content

Developers have a strong handle on how to manage and deploy code assets. Yet at some point in our application build, we've all said, "What about this text? What about these images? Where do these belong?" That's pretty universal. Nearly every single application today has content in it. Whether it's a web app or a native app, it's full of strings, images, icons, media, and other classes of content.

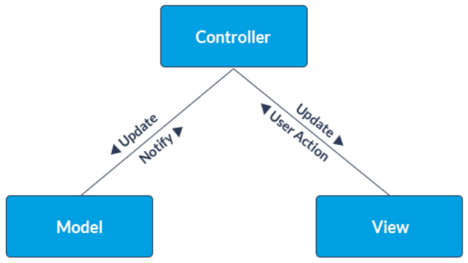


Figure 1: Model View Controller Pattern

This content doesn't really belong in our code base — because it's not code. These non-code assets make us, as developers, pretty uneasy. We know that at some point, a business user is going to ask us to make a change to one of those strings, and we're going to spend hours of build and deploy cycles to handle a 30-second code change. We know that at some point, we're going to need to translate that content. We know that at some point, we're going to replace this UI with another one. We know all these things — and we know leaving that content, even if it's abstracted into a string table or a resource bundle, is going to come back to haunt us. Developers need to update it. Developers and systems folks need to deploy it. No matter the abstraction, it's part of the build.

Smart developers separate code from content. They make sure that the content in the application is completely independent of the build-and-deploy cycle of the application itself. Where appropriate, they make sure non-technical business users have access to the externalized content and can update it and publish changes at any time.

CAN I USE ANY WEB CMS TO SUPPLY CONTENT TO MY APPS?

Maybe. A lot depends on the app's needs and the CMS. If the app is web-based or a native app, and the CMS supports APIs/headless mode, then the answer is yes.

But there's a catch. In a few words, the ability to build the app against a CMS alone is not sufficient. Technology choices like programming language and architecture matter. Making the right choices up front can save countless hours and dollars of rework later.

Technology	Choose a CMS that aligns architecturally and that ideally aligns technology-wise with your existing technology investment and skills.
Architecture	CMS technologies based on RDBMS database and JCR repositories rely on clustering and replication. For content delivery, these systems rely on clustering and replication to handle high throughput. For high transaction throughput applications, make sure you have a plan to properly scale out the back-end, or consider a CMS with a "sharding" content delivery architecture. Decoupled CMS platforms are more likely to support a shared-nothing topology.
Purpose	Many CMS technologies were created to manage web pages. Make sure the CMS is either designed to manage the content you intend to put in it, or that it's appropriately content agnostic enough to meet your needs.

What Does a Modern CMS Provide?

Ease of use	Modern CMS platforms are very easy to use. They typically include features such as: • In-place/in-context editing. • Site/web project creation and management. • Content preview. • Drag-and-drop content and template layout. • Workflow and other dashboards. • Multilingual management and translation workflows.

Library services	Believe it or not, some of the most widely used CMS platforms don't provide proper library services like locking and versioning. Developers have become familiar with extremely powerful version control platforms like Git. A modern CMS provides its users with solid library services.
Channel agnostic and multi-channel	Modern CMS is multi-channel. It can manage any kind of content and it can provide content to any kind of consumer.
Decoupled architecture	Today's demands regarding content, multi-channel, scalability, and legal/data security are driving modern CMS platforms toward decoupled architectures.
Freedom of framework	Modern CMS is "unopinionated" about the presentation technology. JavaScript frameworks and other presentation technologies change quickly.
Cloud-class scalability	Today's use cases demand scale. You need to be able to quickly and easily spin up additional capacity.
High-performance content delivery	A modern CMS should have a strong focus on performance. It should not unnecessarily require cluster nodes for basic needs. Usually this implies either a caching technology or some kind of static generation and delivery.
Geo-distributed and disconnected content delivery	Content performance (how quickly an app, a web page, or a video loads) has a huge impact on the bottom line. Countries now enforce strict rules about the flow of data in and out of their borders. Some locations that benefit from access to content are not constantly connected to the internet. For these and many other reasons, a modern CMS offers topologies that allow for geo-distribution.
Developer-friendly	No matter how close the fit a CMS is, the ability for a CMS to adapt is crucial. You may need to connect with other systems in your enterprise (security, CRM, ERP, etc.) or you may need to implement a custom business rule or feature. The more developer-friendly a CMS is, the better.
Strong content modeling	Content modeling is a fundamental capability of a CMS. Make sure you can easily define the structure of your content types and the associations (relationships) between them.
Powerful templating	Templates are the heart of a CMS's presentation capability. Powerful template engines provide you with a lot of built-in capabilities but don't limit you too much. Be wary of CMS technologies that enforce specific limitations on layout and design.

Powerful development capabilities	Today, innovation is the name of the game. CMS technologies need to make it easy for content managers, but they also have to make it easy for developers. Integration with IDEs, development process, and CI and the ability to easily workflow code from development to production is a must for any application, website, or other digital experience that's going to change frequently.
API and headless support	API support is a must. A modern CMS will expose its capabilities as APIs so that you can integrate and build automation around the CMS. More importantly, the CMS needs to make the content easily consumable via APIs.
Powerful search and query	Field schemas and types, full text indexing, the ability to handle binaries, boosting, fuzzy searching, faceting, and synonyms are just a few examples of search/query capabilities a modern CMS needs to provide. The heart of dynamic behavior is the ability to perform real, sophisticated queries quickly. If a CMS doesn't have a solid answer for search, you will quickly find its limits.
Personalization	Today's users are used to customized interfaces that adapt to them based on factors like device, location, demographics, usage history, and preferences. A modern CMS needs to provide a solid answer for driving the right content to the right person at the right time in the right context.
Push-and-pull publishing	A modern CMS can get content delivered wherever it needs to, whenever it needs to. Traditional CMS platforms tend to rely on consumers coming to them and requesting the content. A modern CMS can meet this need but is just as comfortable pushing the content into a third-party system.
Library services that work for developers	A modern CMS is as good for developers as it is for authors. That means integrating with developer tools and process — and it also means enabling versioning and workflow that's appropriate for code artifacts.
Workflow and scheduling	For many, content authoring and management responsibility is distributed across a team. In order to truly facilitate delegating work, a process or workflow is needed to ensure quality and accuracy of content. A modern CMS provides easy, powerful workflow to help facilitate collaborative content development as well as scheduled publishing so that content can be published at a time in the future once approved.

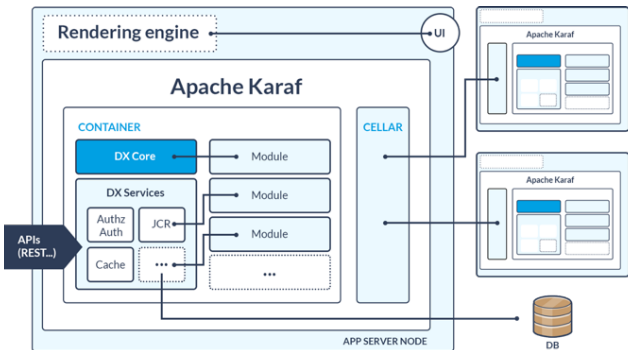
Putting a Modern CMS Into Practice With Jahia DX

Jahia is an open source, Java-based Digital Experience Platform. In this section, we'll walk you through getting Jahia DX installed and up and running with a digital content experience: **A demo website (web project)** that is prepopulated with content to discover and learn web content editing basics.

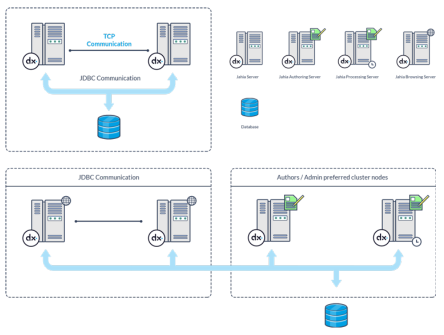
JAHIA DX ARCHITECTURE

- Leverages several open-source technologies (Apache Karaf, Tomcat, Jackrabbit, Red Hat).
- Clustering and scaling features using Apache Karaf Cellar.
- High-performance front-end and back-end caches.
- Option to choose among decoupled, headless, or hybrid CMS.
- Rendering engine that provides dynamic and personalized delivery of > every request at speed.
- Support for all three CMS stakeholders: content authors, developers, and system administrators.

Jahia DX's fragment-based HTML cache makes sure that content is only regenerated when needed thanks to smart dependency analysis and scoped invalidation. Only modified content (including any of its dependencies) is invalidated so that your site keeps humming even under high traffic.



Local architecture (simplified)



Physical architecture

INSTALLING AND RUNNING JAHIA DX

To make the installation as simple as possible, you can use a Docker container to deploy DX and its dependencies. If you don't have Docker installed on your system, you can get it here: docker.com/get-started

To run a DX Docker image:

After installing Docker, open a Terminal window on Mac or a Command Prompt on Windows (cmd.exe) and type the following commands:

```
docker pull jahia/dx-dev:7.3.0.2
docker run -p 8080:8080 jahia/dx-dev:7.3.0.2
```

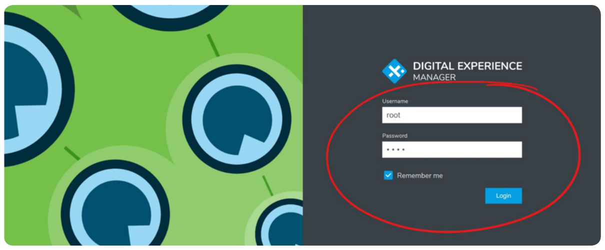
The initial startup of DX takes a few minutes. When DX finishes starting, you should see a message that looks like this:

```
___________________

D E V E L O P M E N T M O D E A C T I V E

In development mode, Digital Experience Manager
will allow JSPs to be modified, modu les to be
re-deployed and other modifications to happen
immediately, but these DO have a performance
impact.

It is strongly recommended to switch to
production mode when running performance tests
or going live. The setting to change modes is
called operatingMode in the jahia.properties
configuration file.

___________________

Modules:
    Started: 65
___________________

Digital Experience Manager 7.3.0.2 [Hit-Girl] -
Enterprise Distribution - Build 59171.4436 is now
ready. Initialization completed in 268 seconds

___________________
```

LOGGING INTO DX

You are now ready to connect to DX. Simply open a supported browser (see the Supported stack for a list of supported browsers) and enter the following URL: http://localhost:8080

The login page will be displayed:



Username
root
Password
••••
☑ Remember me

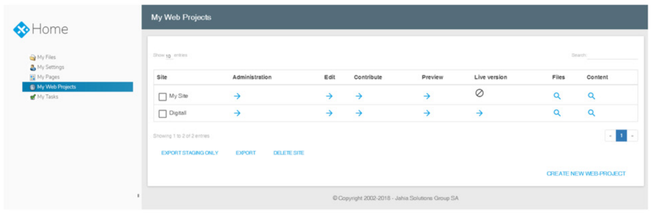
Login using the following default credentials:

Username: root

Password: root

Important: *Docker images are not intended to be put in production as-is. Exposing the images to the outside world without any further configuration could expose you to security issues.*

The dashboard home page will be displayed:


Home

My Files
My Settings
My Pages
My Web Projects
My Tasks

My Web Projects

Show 10 entries   Search:

Site	Administration	Edit	Contribute	Preview	Live version	Files	Content
☐ My Site	→	→	→	→	⊘	🔍	🔍
☐ Digitall	→	→	→	→	→	🔍	🔍

Showing 1 to 2 of 2 entries

EXPORT STAGING ONLY   EXPORT   DELETE SITE

CREATE NEW WEB-PROJECT

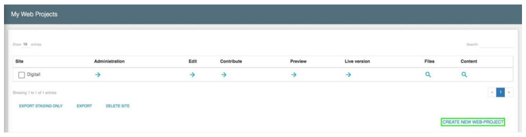
© Copyright 2002-2018 - Jahia Solutions Group SA

CREATING A WEB PROJECT

Web projects are virtual web sites that you can edit in DX. One single DX server can host several web projects for different teams and can handle separate domain names if needed.

To create a web project:

1. In your dashboard, click **My Web Projects** in the left menu, and click **Create New Web Project** in the main pane.


My Web Projects

Show 10 entries   Search:

Site	Administration	Edit	Contribute	Preview	Live version	Files	Content
☐ Digitall	→	→	→	→	→	🔍	🔍

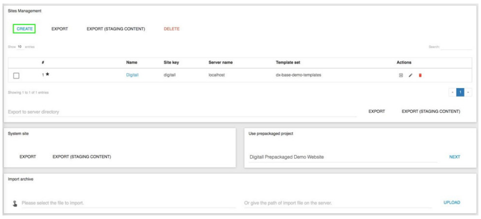
Showing 1 to 1 of 1 entries

EXPORT STAGING ONLY   EXPORT   DELETE SITE

CREATE NEW WEB-PROJECT

2. Do one of the following:

   - To create a new web project from scratch, click **Create**.


Sites Management

CREATE   EXPORT   EXPORT (STAGING CONTENT)   DELETE

Show 10 entries   Search:

	#	Name	Site key	Server name	Template set	Actions
☐	1 ★	Digitall	digitall	localhost	dx-base-demo-templates	

Showing 1 to 1 of 1 entries

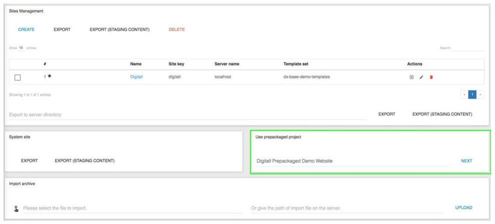
Export to server directory   EXPORT   EXPORT (STAGING CONTENT)

System site
EXPORT   EXPORT (STAGING CONTENT)

Use prepackaged project
Digitall Prepackaged Demo Website   NEXT

Import archive
Please select the file to import.   Or give the path of import file on the server.   UPLOAD

   - To create a project from a prepackaged site, in **Use prepackaged project**, select a project and click **Next**.


Sites Management

CREATE   EXPORT   EXPORT (STAGING CONTENT)   DELETE

Show 10 entries   Search:

	#	Name	Site key	Server name	Template set	Actions
☐	1 ★	Digitall	digitall	localhost	dx-base-demo-templates	

Showing 1 to 1 of 1 entries

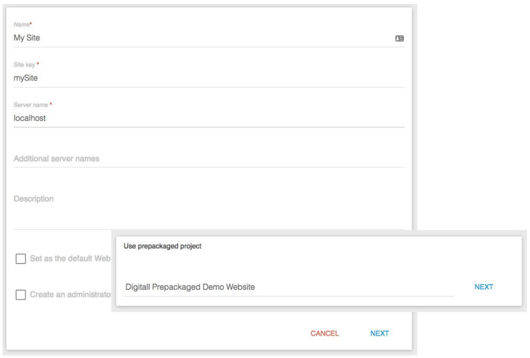
Export to server directory   EXPORT   EXPORT (STAGING CONTENT)

System site
EXPORT   EXPORT (STAGING CONTENT)

Use prepackaged project
Digitall Prepackaged Demo Website   NEXT

Import archive
Please select the file to import.   Or give the path of import file on the server.   UPLOAD

3. Set the properties for your new web project and choose modules if needed in the dialogs that follow.


Name*
My Site

Site key *
mySite

Server name *
localhost

Additional server names

Description

☐ Set as the default Web
☐ Create an administrato

CANCEL   NEXT

Use prepackaged project
Digitall Prepackaged Demo Website   NEXT

You can also create an empty web project and use DX as a headless CMS to build a single page application, a progressive web application, or a native mobile application that will use DX's GraphQL API to access headless content. To learn more, check out Headless React and GraphQL app tutorial in Jahia's resource center.

SERGEHEADSHOT (1)

Written by **Serge Huber,** *Chief Technology Officer (CTO) and co-founder at Jahia*

Serge Huber is the Chief Technology Officer (CTO) and co-founder at Jahia. Serge has more than 30 years of experience in developing digital experience solutions in various technologies, including AR and VR, and is constantly striving to find new ways to build high-quality and high-performance software. He has experience in building high visibility mission-critical applications for large customers (such as HomeAway, BNP Paribas, Sodexo, and the European Parliament). He now oversees the future technical development of Jahia's software, and manages the interaction with open-source communities such as the Apache Foundation. He is a VP and the initiator of Apache Unomi and a committer on the Apache Jackrabbit Project. He still considers himself primarily a geek and enjoys talking at events such as Java One, ApacheCon, UX+Dev Summit, and more.

DZone
A DEVADA MEDIA PROPERTY

DZone communities deliver over 6 million pages each month to more than 3.3 million software developers, architects, and decision makers. DZone offers something for everyone, including news, tutorials, cheat sheets, research guides, feature articles, source code, and more. "DZone is a developer's dream," says PC Magazine.

Devada, Inc.
600 Park Offices Drive
Suite 150
Research Triangle Park, NC

888.678.0399   919.678.0300

Copyright © 2019 DZone, Inc. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by means electronic, mechanical, photocopying, or otherwise, without prior written permission of the publisher.