Getting Started With

Java-Based CMS

BY RUSS DANNER

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 webpage 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 is a definition of the types of content you want to capture that's comprised of the fields for each type you want to capture along with any structural relationships between types. 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.

CONTROLLER

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 controller. A controller is code that can take content based on the model and prepare it for the view/presentation by applying business rules.

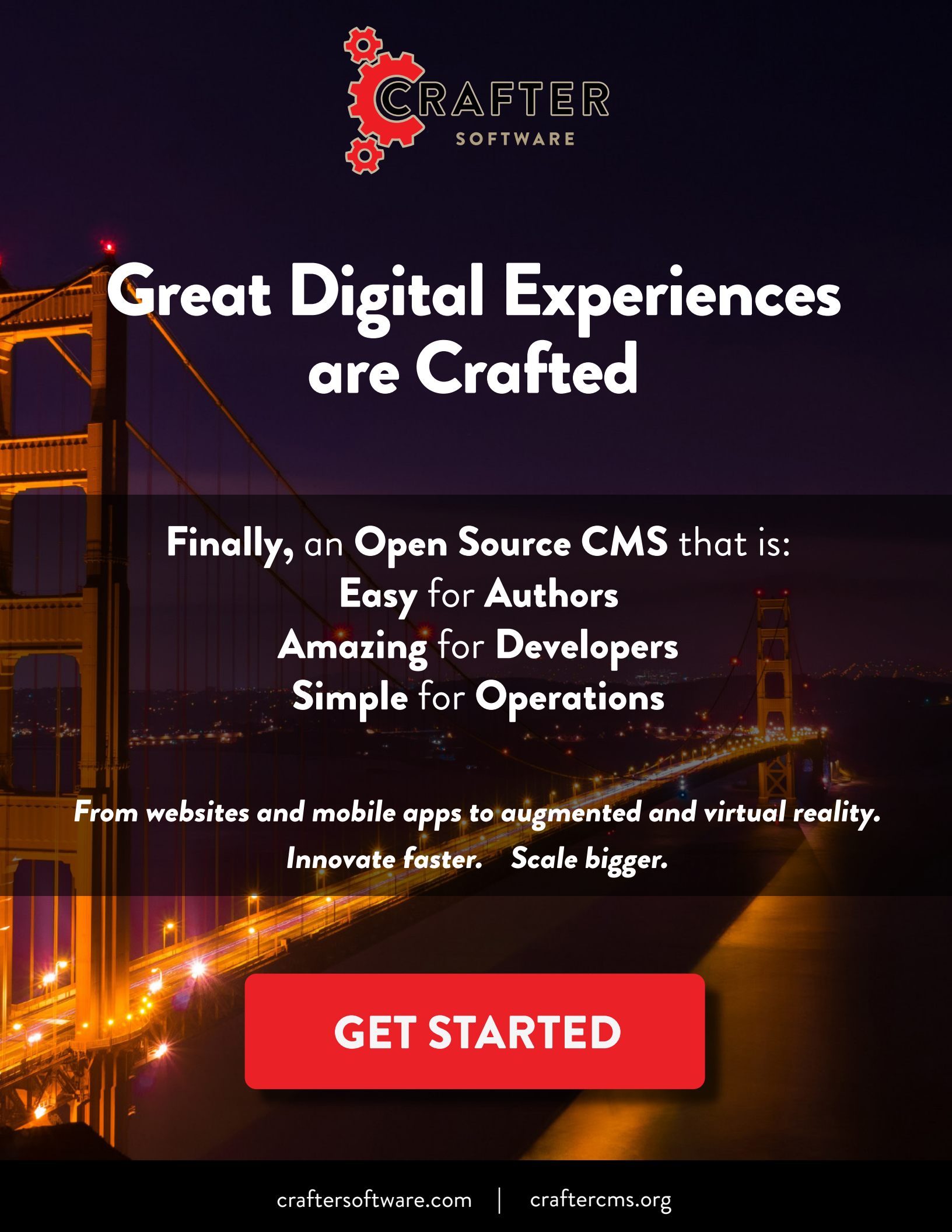
PRESENTATION

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 mobile app 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

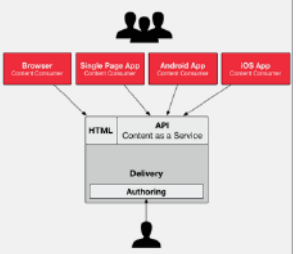
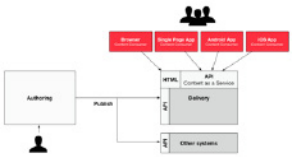
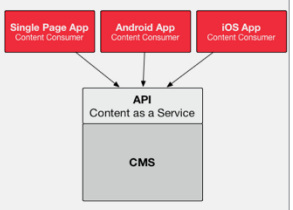
Continued on next page


CRAFTER
SOFTWARE
Great Digital Experiences are Crafted
Finally, an Open Source CMS that is:
Easy for Authors
Amazing for Developers
Simple for Operations
From websites and mobile apps to augmented and virtual reality.
Innovate faster. Scale bigger.
GET STARTED
craftersoftware.com | craftercms.org

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 capability to the content delivery components of the system.
Publishing	Publishing is a category of capabilities that define 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	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.	
Decoupled	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.	
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).	

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. And 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 separation of authoring and delivery capabilities.

A headless CMS can either be coupled or decoupled. Further, any CMS worth its salt today (decoupled or not) must support headless/CaaS-based content delivery.

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 context: alignment with your requirements, your business process, 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 on line in short order and that do not need to be able to scale quickly or to 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. Be it a web app or a native mobile app, it's full of strings, images, icons, media, and other classes of content.

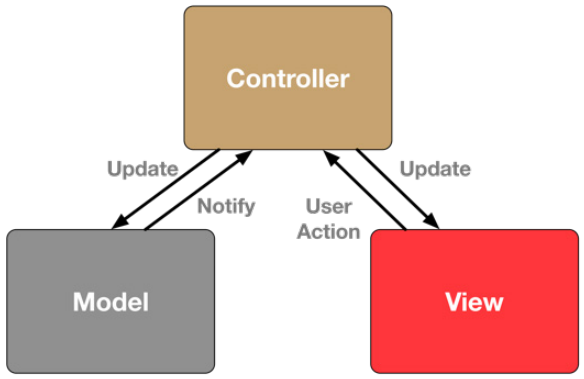


Figure 1: Model View Controller Pattern

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 content change. We know that at some point, we're going to need to translate that content. We know 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 because no matter the abstraction, it's part of the build. Developers need to update it. Developers and systems team members need to deploy it.

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 if the app is 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 matter. Architecture matters. Making the right choices up front can save countless man 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 databases and JCR repositories rely on clustering and replication for reliability and scalability. These technologies can be difficult and expensive to scale out. For high transaction throughput applications, make sure you have a plan to properly scale out the backend or consider a CMS with a "shared nothing" 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 your CMS is capable of managing a full range of content and provides the freedom to use any UI framework.

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. • Content preview. • Drag and drop content and template layout. • Workflow and other dashboards.
Library services	Believe it or not, some of the most widely used CMS platforms don't provide proper library services like locking and versioning. This really should disqualify them — and over time, it will. 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 or application.
Decoupled architecture	Today's content, multi-channel, scalability, and legal/data security demands are driving modern CMS platforms toward decoupled architectures.
Freedom of UI framework	Modern CMS is "unopinionated" about the presentation technology. JavaScript frameworks and other presentation technologies change quickly. Platforms that enforce a particular approach cannot keep up.
Cloud-class scalability	Today's use cases demand scale. You need to be able to quickly and easily spin up additional capacity. "Shared nothing" architectures offer the most straightforward scalability.
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, modern CMS offer topologies that allow for geo-distribution. "Shared nothing" architectures offer the most straightforward approach for these needs.

Developer-friendly	No matter how close the fit a CMS is, the ability to adapt the CMS is crucial. You may need to connect with other systems in your enterprise (e.g. security, CRM, ERP, etc.) or you may need to implement a custom business rule or feature. The more developer-friendly a CMS, 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 the 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, as well.
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 CRAFTER CMS

Crafter CMS is an open source, Java-based CMS that was designed to handle innovation, scale, and distribution challenges. In this section, we'll walk you through getting Crafter CMS installed and up and running with two types of digital content experiences: a traditional website, and a catalog of products that can be accessed by an external native application or single page application (SPA) via APIs.

CRAFTER CMS BACKGROUND/ARCHITECTURE

- 100% open-source, Java-based CMS.
- Allows you to create or manage any kind of content.
- Decoupled CMS (disconnected global delivery).
- Dynamic/personalized delivery of every request at speed.
- API-first CMS (content as a service).
- Git-based content and code repository.
- Share-nothing delivery architecture.
- Front-end agnostic (bring your favorite UI framework or use as a headless CMS).
- Equal support for all three CMS stakeholders: content authors, developers and system administrators.

Crafter CMS is a true decoupled content management system, yet it supports dynamic and personalized content delivery.

INSTALLING AND STARTING CRAFTER CMS

1. Download the Crafter CMS install zip file from http://craftercms.org/v/craftercms-bundle. The zip file will install a fully functional Crafter Studio instance and a Crafter Engine in "Preview Mode."
2. Unzip the contents in any directory.

STARTING CRAFTER CMS USING THE STARTUP SCRIPT

To start Crafter CMS Server: From the command line, navigate to the INSTALL_PATH/crafter directory and execute the startup script:

- Unix/Linux systems: startup.sh
- Windows: startup.bat

Note: It takes a few moments for Crafter CMS to start up.

STOPPING CRAFTER CMS USING THE SHUTDOWN SCRIPT

To stop Crafter CMS Server: From the command line, navigate to the INSTALL_PATH/crafter directory and execute the shutdown script:

- Unix/Linux systems: shutdown.sh
- Windows: shutdown.bat

GETTING STARTED

Now that Crafter CMS is installed and started, let's get a few projects set up.

- Open Crafter Studio
- In your browser, go to http://localhost:8080/studio.
- Log in with the following:
  - username: admin
  - password: admin
- After logging in, you should be redirected to the My Sites screen. You're now ready to create your first project!

WORKING WITH CRAFTER CMS

Understanding the User Interface and content editing	http://docs.craftercms.org/en/3.0/content-authors/index.html
Architecture	http://docs.craftercms.org/en/3.0/developers/architecture.html
Working with content modeling	http://docs.craftercms.org/en/3.0/developers/content-modeling.html
Templating	http://docs.craftercms.org/en/3.0/developers/templates.html

CREATING A TRADITIONAL WEBSITE PROJECT

A very typical use of a CMS is the creation and management of a website. Using Crafter CMS, we can create and manage beautiful responsive websites based on the front end framework and design of your choosing. To illustrate this point, let's leverage an out-of-the-box editorial website blueprint to quickly get up going.

After logging in, you'll see the MySites screen (below). Click on **Create Site**.

Give the site a friendly name for the *Site Id*, give it a description, and then choose a blueprint. We're going to be using the "Website_Editorial" blueprint. Blueprints offer you a starting point for your website. New blueprints can be created and installed on the system. As you are entering the site id, spaces are removed and uppercase letters are converted to lowercase letters.

Click on **Create** and wait for the system to create your site based on the blueprint. It's creating configurations, site content, and permissions based on the template provided by the blueprint. When it's done, you will be taken to the home page of your site:

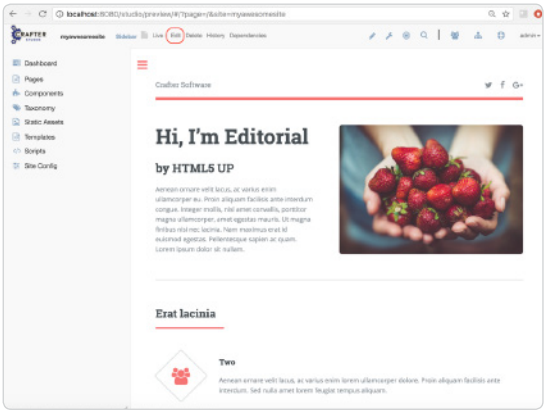


Figure 2: Once the blueprinting process is complete, the information architecture, content, and code of a project can be modified. Figure 2 illustrates a website based on editorial blueprint ready to be edited.

Now that the site has been created from the blueprint, you can modify it by editing existing pages or by creating new pages and components.

CREATING A HEADLESS CONTENT MANAGEMENT PROJECT

An increasingly common use case for CMS is delivering content via APIs (Headless/Content as a Service). This approach allows applications of any kind, running on any platform and any technology, to easily access content managed by the CMS. In recent years, native mobile applications, single page web applications, game consoles, and even business applications with frequently updated application/business rule content have moved to this approach. To illustrate this point, let's leverage an out-of-the-box headless_store project blueprint to quickly get us going.

After logging in, you'll see the MySites screen (Below). Click on **Create Site**.

Give the site a friendly name for the *Site Id*, give it a description, and then choose a blueprint. We're going to be using the "Headless_Store" blueprint. Blueprints offer you a starting point for your project. New blueprints can be created and installed into the system. As you are entering the site id, spaces are removed and uppercase letters are converted to lowercase letters.

Click on **Create** and wait for the system to create your site based on the blueprint. It's creating configurations, site content, and permissions based on the template provided by the blueprint. When it's done, you will be taken to the preview of your project:

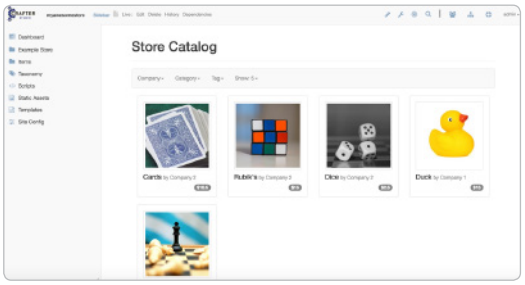


Figure 3: Once the blueprinting process is complete, the information architecture, content, and code of a project can be modified. Figure 3 illustrates a digital experience based on a headless catalog blueprint ready to be edited.

Note that unlike a website, you are not really looking at a webpage but rather a simple catalog of the products your project is managing. This allows your authors to quickly and easily preview and edit these items. This catalog or "console" also serves as a reference implementation for consumers of the APIs that your project will provide to external applications.

With a headless content project, the real rubber meets the road when you begin to define APIs and deliver the content through them. For many headless CMS technologies, you do not get to choose what your APIs look like or the format of the JSON that is returned. Crafter CMS allows you to define your API URLs and response data structures through simple Groovy scripting:

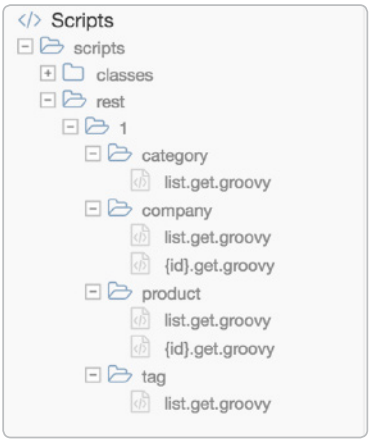


Figure 4: Crafter CMS gives developers complete control over RESTful response urls, formats and content via Groovy scripting.

```
import org.craftercms.blueprints.headless.
ProductSearchHelper

def id = pathVars.id

def products = new ProductSearchHelper(searchService)

        .filter("objectId: $id")

        .getItems()
if(!products.items) {
        return []
}

return products.items[0]
```

Once the script is complete, the service can be tested and published so that it is available to consumers.

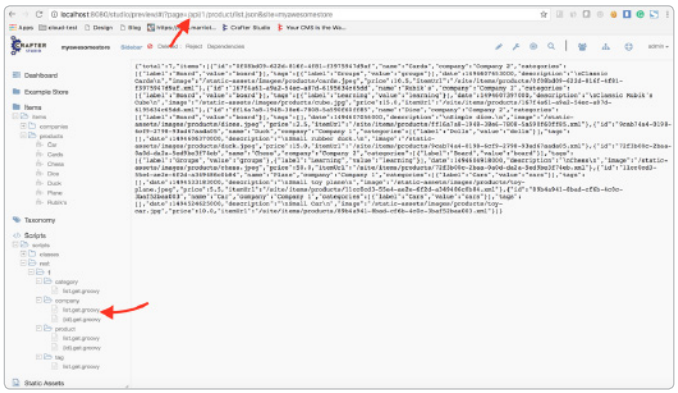


Figure 5: Once a RESTful service is created, it can quickly be tested and published from Crafter Studio. Figure 5 shows the output of a given RESTful service controller.

ABOUT THE AUTHOR

RUSS DANNER is VP Products at Crafter Software and is responsible for product management, product development and support, and client and partner success. Russ brings over 20 years of software architecture, design, and implementation experience. Prior to Crafter Software, Russ was Web Experience Management Practice Director at Rivet Logic and project lead for the open-source Crafter CMS project. Russ has also been active in the open source community since 2000 as a community leader, contributor, trainer, speaker, and user group organizer.

DZONE, INC.
150 PRESTON EXECUTIVE DR.
CARY, NC 27513

888.678.0399
919.678.0300

REFCARDZ FEEDBACK WELCOME
refcardz@dzone.com

SPONSORSHIP OPPORTUNITIES
sales@dzone.com