Decoupled Drupal 101

Table of Contents

Introduction

In the web development world, few trends are spreading more rapidly than decoupled (or headless) content management systems. A decoupled architecture allows developers to utilize any technology to render the front-end experience ("the glass" where a user interacts with an application) in lieu of the theming and presentation layers that come with a coupled CMS out-of-the-box. In a decoupled Drupal architecture, the Drupal back end exposes content to other front-end systems, such as native mobile applications, conversational UIs, or applications built in JavaScript frameworks.

Organizations may select a decoupled Drupal approach for a few reasons. Some implement a decoupled strategy to leverage Drupal as a content repository to serve content to any device in a complex digital ecosystem. Others favor decoupling to allow front-end teams to use popular JavaScript frameworks while maintaining the back-end capabilities of Drupal. For example, digital agencies are taking advantage of decoupled Drupal to showcase creativity in the front end with JavaScript Model-View-Controller (MVC) frameworks, such as Angular or Ember.

In this guide to decoupled Drupal, you'll learn how a decoupled architecture works, when you should consider decoupling, and how developers and designers can leverage decoupled Drupal to deliver ambitious digital experiences.

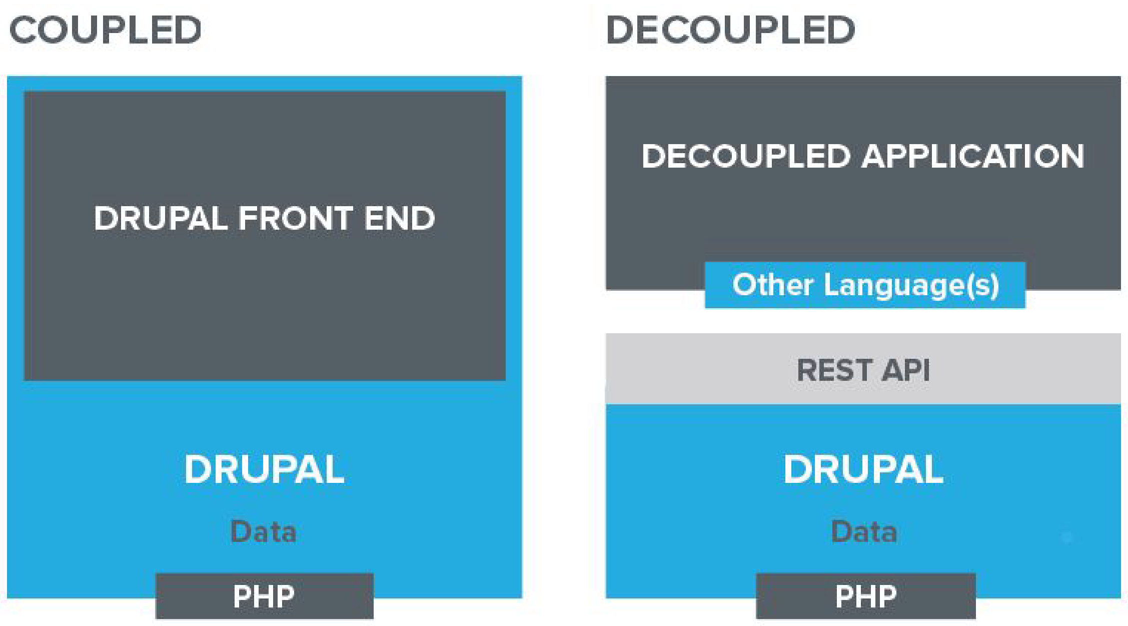
Why Decoupled Drupal?

- **Power a multitude of devices:** With its flexible APIs and web services, Drupal can be the brain behind all of your systems to deliver content everywhere.
- **Leverage other front-end technologies:** Drupal can function as a services layer to allow content created in the Drupal CMS to be presented through a JavaScript framework, such as Ember, React, and Angular.
- **Control all aspects of your media:** Decoupled Drupal can serve as a central repository to send video and data to the many outlets available in the current media marketplace.
- **Integrate with multiple systems:** Organizations can introduce Drupal to the back end in order to support existing technical systems.

How it Works

What is the difference between a traditional and decoupled architecture?

Drupal is a content management system that can back a variety of decoupled applications, such as mobile applications, conversational UIs like the Amazon Echo, and in-store screens. In a traditional or coupled CMS, both front-end and back-end responsibilities are contained within a single system. The Drupal back end is tightly coupled to the presentation, and this means that the layers that render and deliver data into templates cannot be separated from the PHP back end.



A **decoupled architecture** allows the developer to utilize another technology to render the front-end experiences in lieu of the theming and presentation layers that come with Drupal out-of-the-box. This process employs Drupal as a content repository, which exposes content and data for consumption by other applications. These applications could include:

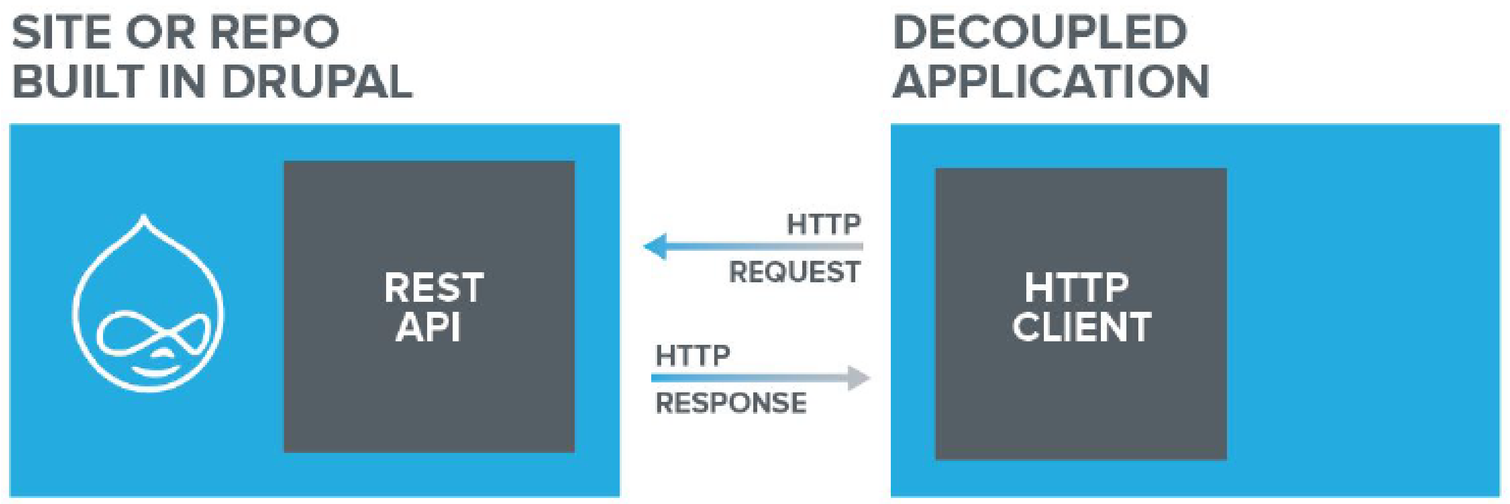
- **Native applications** are applications developed for one specific device or platform such as desktop or mobile. For example, mobile applications are often developed for a particular smartphone device. iOS and Android applications are developed in Swift and Java, respectively.
- **Single-page JavaScript applications** update pages dynamically through asynchronous requests back to the server and eschew full page refreshes. This means that web applications can call the server without the browser needing to reload the page.
- **Internet of things applications,** which include devices like the Amazon Echo, Apple Watch, or connected fitness trackers.

How does a decoupled architecture work?

Most interactions that take place on the web rely on a request/response paradigm where a user requests data and a system responds by gathering, formatting and rendering the appropriate content within a predefined HTML template. Instead of all of this taking place within a single system, a decoupled architecture distributes these responsibilities among multiple systems, as it enables a central content repository and a variety of applications to exchange data over a network.

In a decoupled architecture, the Drupal back end acts as the content repository. The Drupal repository and the decoupled application exchange data through standard HTTP methods. A Representational State Transfer (REST) API is the most common entry point for the Drupal repository, while an HTTP client in the decoupled application is responsible for interpreting the REST API's responses.

The **REST API** is a technique that is frequently used in the development of web services. A REST API provides developers with a set of functions that can execute HTTP requests and responses between an independent server and client. In Drupal 8, the REST API is made available out-of-the-box.



When a decoupled application issues an HTTP request, the REST API answers with the requested content. Imagine the decoupled application is a spacecraft communicating with its Earth-based mission control (in this case, the Drupal repository). The HTTP call and response are like radio transmissions that move back and forth between the spacecraft and base. The REST API is the transmitter for the base. The HTTP client is the spacecraft's radio.

An HTTP response can be served in JSON, XML, in addition to other representations. In the figure above, the REST API and HTTP client act as the mediators in the decoupled architecture, allowing both back-end and front-end developers to work with their preferred frameworks.

What is the difference between headless and decoupled?

In this guide, we have chosen to use the terms headless and decoupled interchangeably. The distinctions between headless and decoupled are pretty nuanced and can be difficult to define. However, a good rule of thumb is this: all headless CMSes are decoupled, but not all decoupled CMSes are headless.

Simply put, a **headless architecture** is the clean division between the back end and front end. This pertains to the interaction between one CMS and one front end.

A **decoupled architecture** includes this implementation but also extends to service-oriented architectures. This means that the front end can be selective in its communication with the various components of the CMS. For example, the front end could pull requests from the back end's user service or content service independently. In a headless architecture, the front end cannot segment requests based on these distinct services.

What are Drupal Web Services?

Another term synonymous with the decoupled movement is the ability for organizations to be **API-first**. Traditionally, being API-first translates to building your API first and then developing your web application around it. However, organizations can have various definitions of what being API-first means for them. For Drupal, it means making the content stored and managed by Drupal available for other software.

In order to make Drupal the best open-source and API-first CMS, the Drupal community has been working for many years to develop the most robust offering of web services available. This is important because not all decoupled applications communicate via a REST API. In order to make Drupal truly API-first, web services need to extend functionality beyond the reach of the REST API.

Today, Drupal offers a range of web services modules, which enable anyone to consume data from Drupal with ease. These include contributed modules in addition to modules that are now available in Core (the standard base package that defines the latest release of Drupal):

Core RESTful Web Services: With the release of D8, Drupal now offers a REST API out-of-the-box. This includes operations to create, read, update and delete (CRUD) content entities and to read configuration entities. There are also four primary REST modules in core, including Serialization , RESTful Web Services , HAL, and Basic Auth. Core REST requires limited configuration while providing a wide range of features.

JSON API: JSON API is increasing in popularity due to its adoption by developers from both the Ruby on Rails and Ember communities. JSON API is a specification for REST APIs using the JSON format and offers functionality beyond the core REST API. The JSON API module is available for Drupal 8.

GraphQL: Originally developed by Facebook, GraphQL is a query language developed for client-tailored queries. GraphQL enables clients of decoupled Drupal to easily pull data from the back end, allowing them to retrieve custom sets of data through a single request. Drupal 8 now supports the GraphQL module.

What software development kits does Drupal offer?

In addition to a robust collection of web services, Drupal now offers an ecosystem of software development kits (SDKs), which help to accelerate the development of applications in other technologies. SDKs can help to extend Drupal's reach outside of PHP.

Previously, consuming Drupal content required some understanding of the REST API implementation details. This changes with Drupal's Waterwheel module, which helps developers build Drupal-backed applications in JavaScript and Swift, without needing extensive Drupal expertise:

Waterwheel.js helps JavaScript developers query and manipulate Drupal data. Waterwheel.js can be leveraged universally to reuse code across both server and client. On the server side, Waterwheel.js can be used to make API calls within Node.js during server-side execution of a JavaScript framework. On the client side, it can be used to issue asynchronous requests as needed by the JavaScript framework.

Waterwheel.swift provides support for developers building Swift applications. This includes the ability to consume Drupal 8's core REST API in addition to offering session management and authentication capabilities for Swift applications. Waterwheel.swift supports iOS, macOS, tvOS, and watchOS out-of-the-box to simplify communication between Drupal and Apple-driven applications.

When should you decouple?

There is a lot of hype around decoupled architectures, so before embarking on a project, it is important to make a balanced analysis. If you choose to use Drupal as the content repository to serve multiple applications, a decoupled architecture could be right for you. A decoupled approach could be the right solution in any of the two following scenarios:

1. A traditional Drupal site powering one or more sites or applications
2. A Drupal site without a front end powering multiple sites or applications

However, a decoupled architecture might not be suitable for standalone sites and applications. Implementing a decoupled architecture in these instances can duplicate workflows and obstruct the benefits of a coupled Drupal architecture. Taking a decoupled approach for the “sake” of going decoupled without a clear and justifiable need can complicate a project that would otherwise be perfectly suited to a traditional Drupal implementation.

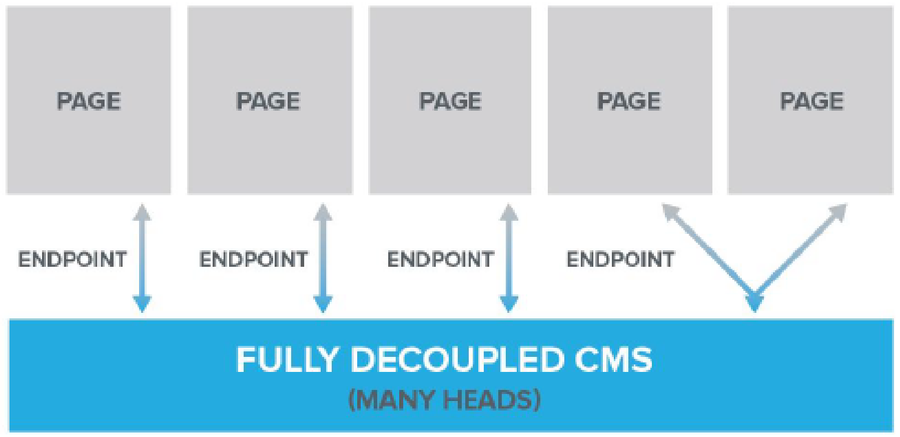
What are the differences between fully and progressively decoupled Architectures?

Retaining a coupled CMS architecture is a perfectly valid option for those building standalone sites and apps. However, if you suspect that your needs go beyond Drupal’s typical offerings, you could select either a **fully decoupled** or **progressively decoupled** strategy.

A **fully decoupled** architecture separates the Drupal front end from the back end in its entirety. There are several advantages associated with a fully decoupled Drupal architecture. Preston So explains both the rewards and risks inherent to fully decoupled Drupal in his blog, The Risks and Rewards of Fully Decoupling Drupal :

– **Separation of concerns :** Utilizing Drupal strictly as a content repository can facilitate a separation of concerns, where responsibilities of functionality and data model are dedicated to the back end, while presentation and aesthetics are devoted to the front end. With a fully decoupled architecture, the handling of content is confined to the back end. This is separated from the front end, which only addresses the presentation and delivery of that content to the end user.

– **Pipelined development:** This separation also extends to both back-end and front-end teams and allows developers to work at their own independent development velocities. Front-end developers are free to control markup and rendering, while back-end developers can focus their efforts on developing a robust RESTful API.



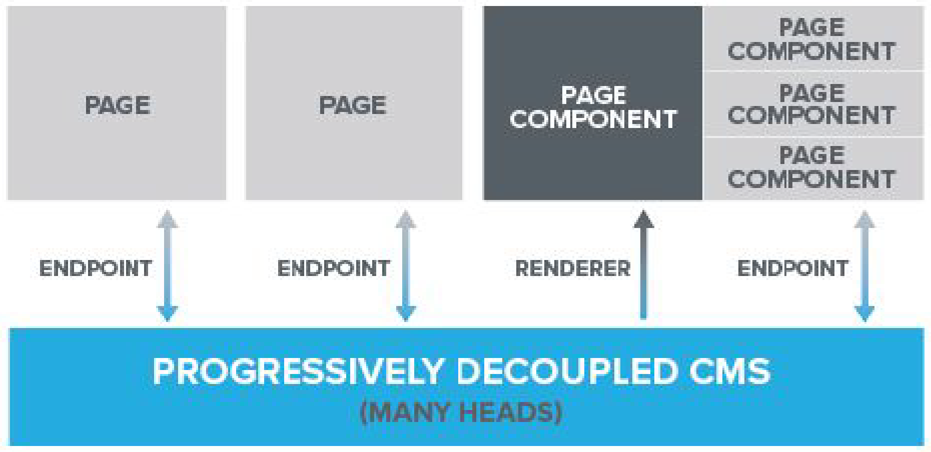
Risks of Fully Decoupled

When the entire front end is controlled by a JavaScript framework, technical teams cannot take advantage of the Drupal capabilities many developers have come to know and love. This strategy of fully decoupling negates Drupal capabilities like in-place editing and display management; it also introduces additional points of failure. Additional risks include:

- No cross-site scripting protection or input sanitization (esp. when not using a framework)
- No layout and display management
- No previewable content workflow
- No modules affecting the front end
- No system notifications, errors or messages
- No BigPipe progressive loading, or advanced caching strategies
- No accessible markup or user experience benefits

Benefits of Progressively Decoupled Drupal

An increasingly popular solution to mitigate the risks of fully decoupling Drupal is a **progressively decoupled strategy**. Unlike a fully decoupled architecture, progressively decoupled implementations insert a JavaScript framework into a Drupal site's front end. JavaScript frameworks continue to consume the REST API; however, certain areas of content can be controlled and rendered by Drupal while others can still take advantage of client-side rendering. Progressively decoupling enables the front-end experience of JavaScript while editorial and technical teams can both continue to take advantage of valuable Drupal capabilities. The rewards of a progressively decoupled build mitigate the risks inherent to a fully decoupled architecture.



Why should you decouple?

At Acquia, our team sees an enormous variety of enterprise Drupal use. Today, digital experiences need to be omnichannel, interact with numerous applications, and offer an API-first architecture. Below are a few examples of how organizations can leverage decoupled Drupal to build ambitious digital experiences:

Powering a multitude of devices: Web services can send data to any digital device, not just to computers and smartphones. Coupled with Drupal's ability to integrate with other systems, API-first Drupal can serve as a powerful brain for complex systems. An API-first approach positions Drupal at the center of many technical ecosystems and simplifies the distribution of content between numerous applications.

However, many of these applications have their own unique requirements. Many marketers have been conditioned to conceptualize content in the form of HTML pages, but page-centric architecture does not apply to all of the content opportunities available today. For example, an Amazon Echo (a zero user interface , which is any device without a visual display or screen) and a traditional website do not understand content in the same way. Digital teams can accommodate a variety of channels by treating content more like data and transitioning towards models of structured content. Structured content refers to the strategy of organizing digital content into independent types with constituent fields and references to other content. Treating content like structured data better enables marketing teams to create once and publish everywhere, across numerous touchpoints and channels. Creating structured content allows organizations to better scale their marketing efforts and to interact with a customer on their preferred device, whether that be the web, a mobile application or Amazon Echo.

Drupal is the best CMS for structured content because it allows for the rich customizability of content types, fields and metadata out-of-the-box. Additionally, Drupal provides excellent tools to manage field entities, powerful taxonomy and tagging capabilities, and full-featured metatag controls.

Leveraging other front-end technologies: Recent developments on the web have created demand for more flexibility in the client. JavaScript frameworks are increasing in popularity due to the promise of increased productivity and maintainable code. Many exist, some with variations on the Model-View-Controller (MVC) model. Some of the most popular include Ember, React, and Angular. Drupal can function as a services layer to allow content created in the Drupal CMS to be presented through a JavaScript framework. This use case is especially important for digital agencies, who can choose to take advantage of other front-end technologies besides those native to Drupal. Drupal's robust collection of web services and flexible APIs provide digital agencies with both technical and creative freedom.

Integrating with multiple systems: Drupal is the CMS of choice for the enterprise. For companies with a standing web presence, this often means converting complex web properties to Drupal websites. In some cases this requires organizations to introduce Drupal to the back end in order to support existing technical systems.

Controlling all aspects of media: Drupal as a services layer can serve as a central repository to send video and data to the many outlets available in the current media marketplace. This of course includes native iOS and Android apps, as described in other examples. But it can also include sending data to live broadcast television, posting to YouTube, and accessing future outlets as they become available.

For more information on how decoupled Drupal can fit into your digital environment, read API-First Drupal: On the Road to Publish Once, Access Everywhere.

LET'S TALK