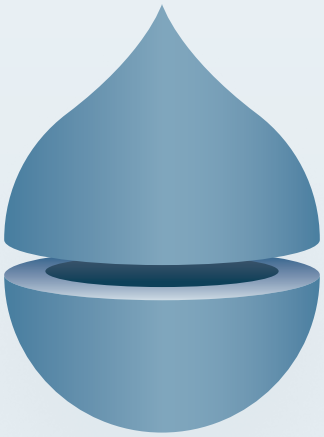


Decoupled Drupal

Is it right for your business?

Table of Contents

What is decoupled Drupal?

In the “old days” of the web, a website lived in a silo. A web page looked at least generally like a page in a brochure or a magazine and was mostly used the same way, as a means of one-way, fairly static communication with the users of the site.

Today, we communicate with customers in numerous ways, across multiple devices. We allow users to interact with our site, push and pull information to social media, and respond to questions spoken into a phone. The web has changed, and so have websites.

We must alter the way that websites are built to adjust to this new omnichannel world where a web “page” might be an app on a phone and content might be reused across many devices and in many ways. We’re not just creating websites anymore.

Fortunately, Drupal was designed from the beginning to be much more than a simple blogging platform, and it’s been flexible enough to evolve as the needs and uses of a website have changed.

48%

browse on multiple devices

25%

browse on computer only

25%

browse on mobile only

Stats above from Google Data, “How People Use Their Devices” 2016

Typically, Drupal is used to deliver all the components of a website, an approach that can be called **traditional**, **monolithic**, or **full-stack** Drupal. In this scenario, Drupal provides the mechanism to create and store structured data, includes an editorial interface that allows editors to add and edit content and set configuration, and automatically provides the front-end markup that users see in their browsers. Drupal does it all.

“ The line between traditional and decoupled is not black and white. There are all sorts of in-between solutions... ”

Decoupled or **headless** Drupal is where website functions are separated across multiple web frameworks and environments. That could mean managing data creation and storage in a traditional Drupal installation, but using React and Node.js to create the page markup. It could also mean using a React app as an editorial interface for a traditional Drupal site.

The line between **traditional** and **decoupled** is not black and white. There are all sorts of in-between solutions, like creating a traditional Drupal website that also serves up data to one or more mobile web applications. Or using JavaScript on the front end of a traditional Drupal site to retrieve content from an external service and inject it into the Drupal-generated markup. This in-between state is sometimes called **progressively decoupled**.

Many organizations, including Netflix, have seen great benefits from a decoupled approach to content but knowing when and if decoupling is right for your website is key.

TECHNICAL ISSUES

	COUPLED	DECOUPLED
Page display management	Drupal	Javascript, Mobile Apps, OTT, Etc.
Page speed and performance	Average	Average to Excellent
Technical stack complexity	Average	Highly Complex
Front to Back-end connection and workflow	Highly Connected	Loosely Connected

EDITORIAL ISSUES

	COUPLED	DECOUPLED
Content management	Drupal	Drupal
Editorial control over content	Complete	Complete
Editorial control over layout/presentation	Complete	None to Partial (additional effort)
Content broadcasting capabilities (apps, etc.)	Yes	Yes
Suitability of content for broadcasting (apps, etc.)	Depends	Excellent
Content preview capability	Good	Possible (additional effort)

Should you decouple?

Just because we can build sites in creative and new decoupled ways doesn't mean that decoupling is the right solution. There are many factors that should enter into that decision.

Determining whether or not you should decouple is a matter of asking the right questions and understanding the benefits and challenges that come with it.

If the benefits apply to your situation, it's worth investigating. But, the challenges of decoupling may rule it out as a solution, at least for now.

DECOUPLING ADVANTAGES

- Support for multiple consumers
- Content reuse capabilities
- Less Drupal-specific knowledge required
- Expedited development
- Enhanced performance
- Reduced cost of future redesigns

DECOUPLING CHALLENGES

- Increased development cost
- Increased technical complexity
- Increased risk of over-engineering
- Increased task difficulty
- Mind-set shift required
- Less page control (layout, order, etc.)
- Multi-site difficulty
- Content model lock-in

The Benefits of Decoupling

SUPPORT FOR MULTIPLE CONSUMERS

More and more sites are delivering content to multiple consumers, mobile apps, TV, etc. Rather than building a website in a silo, we can create content that is intended from the beginning to be reused in many ways. We then treat the front end of the website as just another consumer of that content.

CONTENT REUSE CAPABILITIES

Content is expensive to create; decoupling is a way to reuse it, not just across platforms, but also from website redesign to redesign. By creating a physical separation between the content management and the website itself, it can be easier to create content that is, in fact, reusable, and not hard-wired to the current website design. When the content is only ever displayed on the website, it's likely to be polluted with assumptions about what the "page" will look like.

LESS DRUPAL-SPECIFIC KNOWLEDGE REQUIRED

It can be difficult to find skilled developers to build and manage a website. It may be easier to find generalist JavaScript developers than expert Drupal developers. The cleanly separated front end of a decoupled site is one way to ensure the front-end team doesn't have to know anything about Drupal. From their standpoint, Drupal is just another provider of the JSON they already know how to consume.

EXPEDITED DEVELOPMENT

Some large sites have large development teams, and it can be easy for big teams to get in each other's way as they build out a site. The front-end team can be blocked by back-end work that isn't completed. A decoupled site allows you to have a clean separation of duties, so, once the API has been determined, the front and back ends can proceed in parallel to build the site.

ENHANCED PERFORMANCE

A modern JavaScript front end can be fast. The JavaScript frameworks used on many decoupled projects these days are multi-threaded and asynchronous, making them very speedy. A decoupled site is not automatically faster. You still need to pay attention to performance issues, but the nature of the frameworks makes that easier.

REDUCED COST OF FUTURE REDESIGNS

Once you have decoupled your site, you could launch a brand new design without making any changes to the back end, assuming you have a well-designed API (meaning an API that doesn't include any assumptions about what the front end looks like). Building a new front end is much less work than rebuilding the whole site, and it doesn't require some of the time-consuming tasks of a full replatforming, like migration.

Challenges of Decoupling

INCREASED DEVELOPMENT COST

It almost always costs more to decouple than to build a traditional site. Decoupling requires additional infrastructure and the recreation of solutions provided by traditional Drupal. We're still, as a community, figuring out the best practices so there may be some trial and error in the process of finding the best solutions.

INCREASED TECHNICAL COMPLEXITY

If you only need a website, decoupling is a convoluted way to accomplish it. The infrastructure stack is larger and more complex. The cost of building that more complicated infrastructure only makes sense when you are building more than just a website, like an API to serve multiple consumers.

INCREASED RISK OF OVER-ENGINEERING

You don't have to decouple to support other applications. Full-stack Drupal can be used to create a full-featured website that also provides APIs to other consumers. The website can use all the built-in features of Drupal, while mobile apps consume and re-distribute content from the site.

INCREASED TASK DIFFICULTY

Some tasks are particularly tricky in a decoupled environment, like previewing content before publishing it. In a truly decoupled environment preview makes no sense anyway. There is no concept of a page on an Alexa app or a smart refrigerator displaying a recipe, or even on a smartphone app, or a Roku TV. Content in a truly decoupled application could look totally different than the webpage, and you can't preview it because you can't be sure how it will be used.

MINDSET SHIFT REQUIRED

Many businesses have page-centric assumptions embedded deep into their content and processes. It might be uncomfortable for both editors and management to shift to a mindset where editors create content that might be deployed in many different combinations and environments, that could be reused in many ways. Editors won't be creating "pages", they will be creating reusable, well-structured content.

LESS PAGE CONTROL

Expanding on that point, many editors are used to being able to control things like the layout of the page. They expect to be able to rearrange the page and dictate where in the page their content will appear. Even controlling the URL of a story is problematic in a decoupled world. Content is placed by the apps that consume it, not by editors. There are ways to give editors tools to indicate preferences, and ways to design front ends to respect those preferences, but it adds complexity to the system.

MULTISITE CHALLENGES

Businesses with many properties have a special problem. Decoupling content for re-use across multiple sites implies a unified content model across all those sites. Many multi-site organizations already find it challenging to standardize their sites. But, without a single content model, there won't be a consistent API across the organization. If the content models aren't consistent, there will be multiple decoupled systems, which are even more complicated than a single decoupled system.

CONTENT MODEL LOCK-IN

Once you are decoupled, you no longer control who is consuming your APIs or how they're being used. If you make changes to your content model, you may break things outside of your website. You need to be aware of the dependencies you've created by serving an API.

How about some real-world examples?

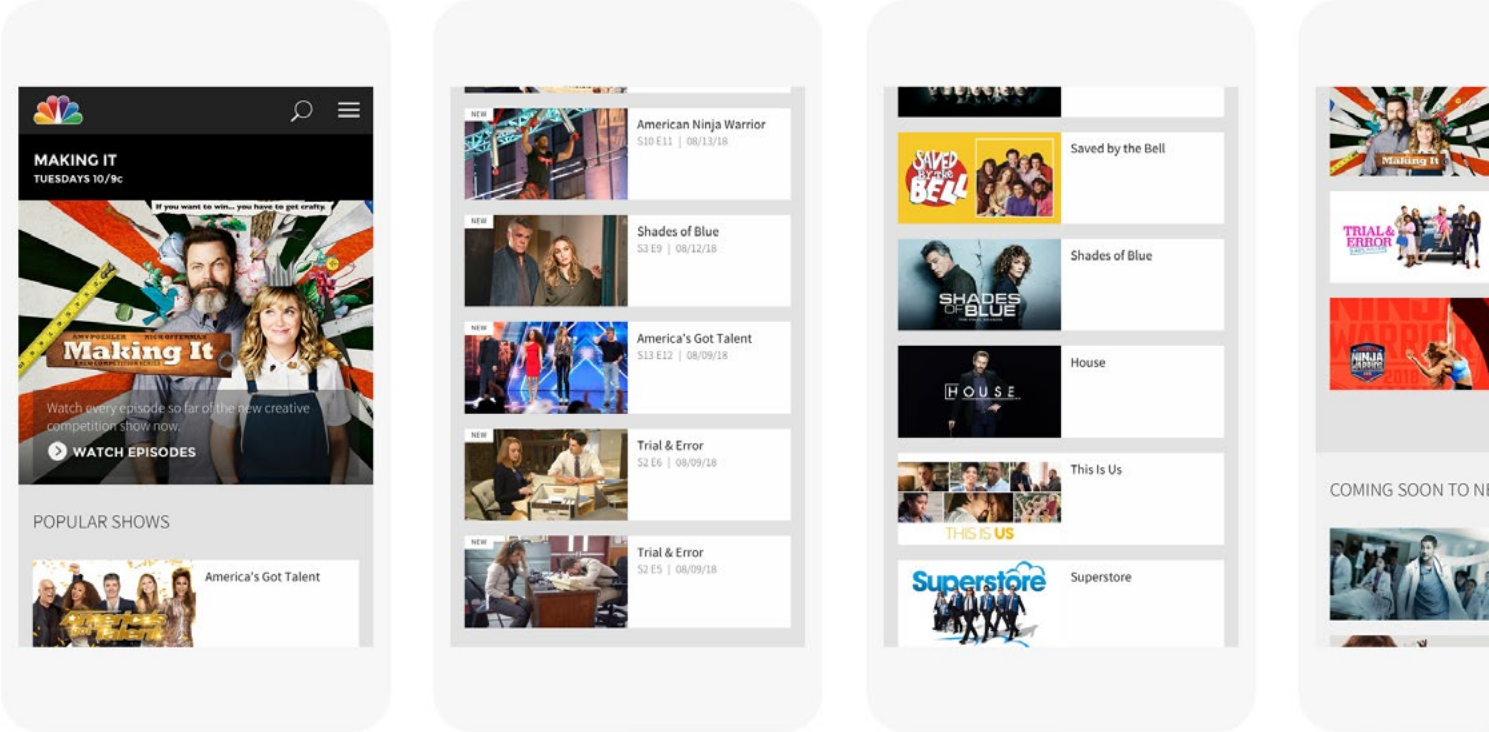
The following pages contain examples of ways we at Lullabot have implemented fully or partially decoupled web architectures using Drupal.

NBC

NBC.com is a Drupal 7 site that started life as a Drupal site that output content in a typical way using Drupal's built-in theme system. It was later retrofitted into a fully decoupled architecture. This transition to a React-based app was done in stages by utilizing the site's CDN to move URLs over one-by-one to the new front end. One goal of this re-architecture was to decouple content and presentation. For instance, fields which indicated very specific layout options such as "this is blue" were abstracted to represent more generic settings, like "which audience is this for?" This abstraction made it possible for different front-end apps to react to that information in any way that was appropriate.

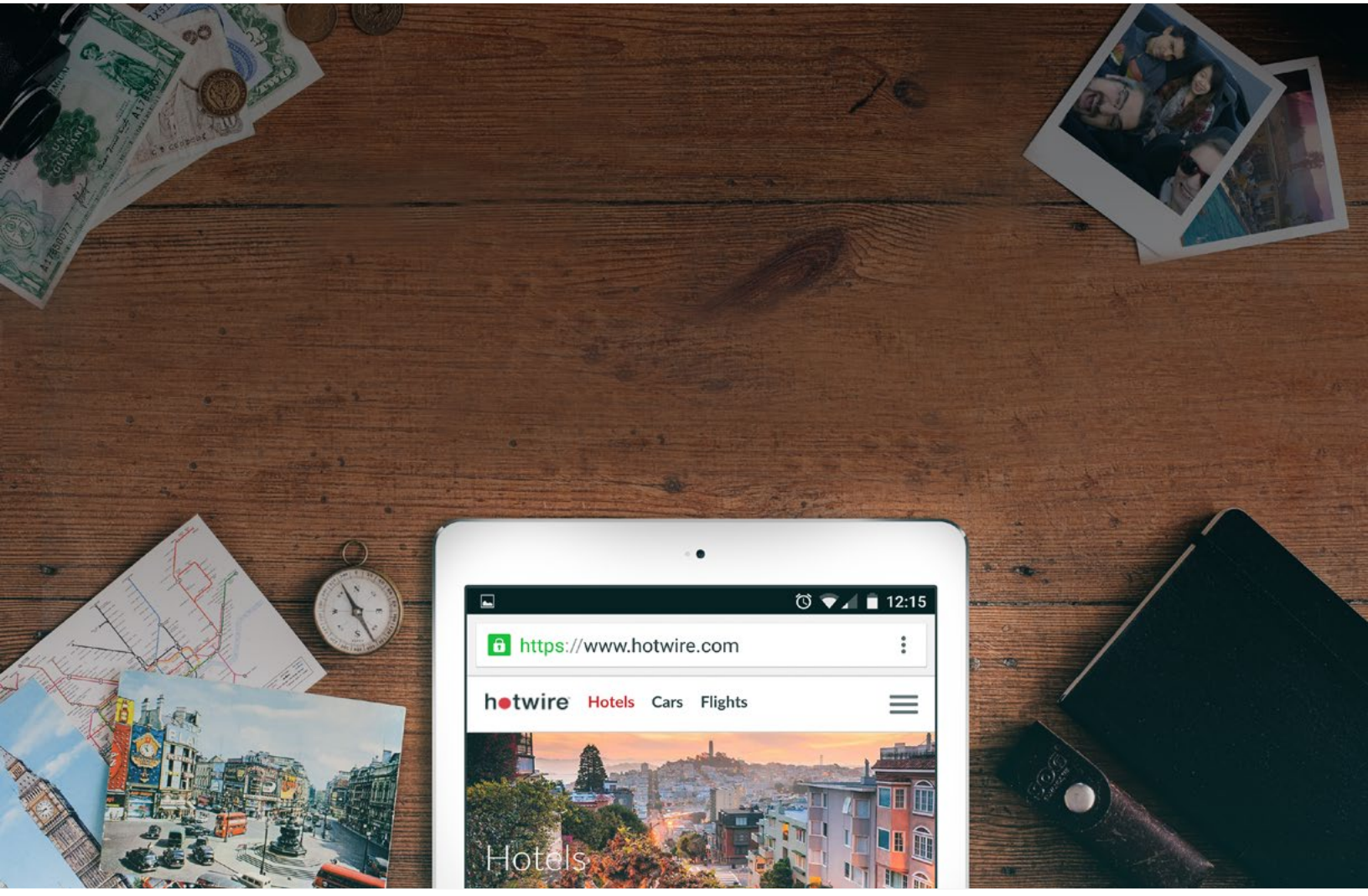
The API is based upon the RESTful Drupal module. However, the front-end app does not directly consume data from Drupal. Instead, a Node.js instance sits in front of the API. This architecture is designed to allow other data from outside Drupal to be aggregated in. It also improves performance, a concern because the site was not originally designed to be API-first. The back-end APIs are now consumed by more than 20 apps for various streaming service providers and televisions.

A preview system allows content editors to view what the site will look like at any date in the future by appending a date to the URL. The front-end app then has the responsibility of passing this date to any APIs it calls, which in turn utilizes the SPS module to get as-yet unpublished data. This functionality is protected by a token authentication system (OAuth2 bearer token) to prevent unauthorized users from accessing future data, as well as allowing penetration of caches to view the content.



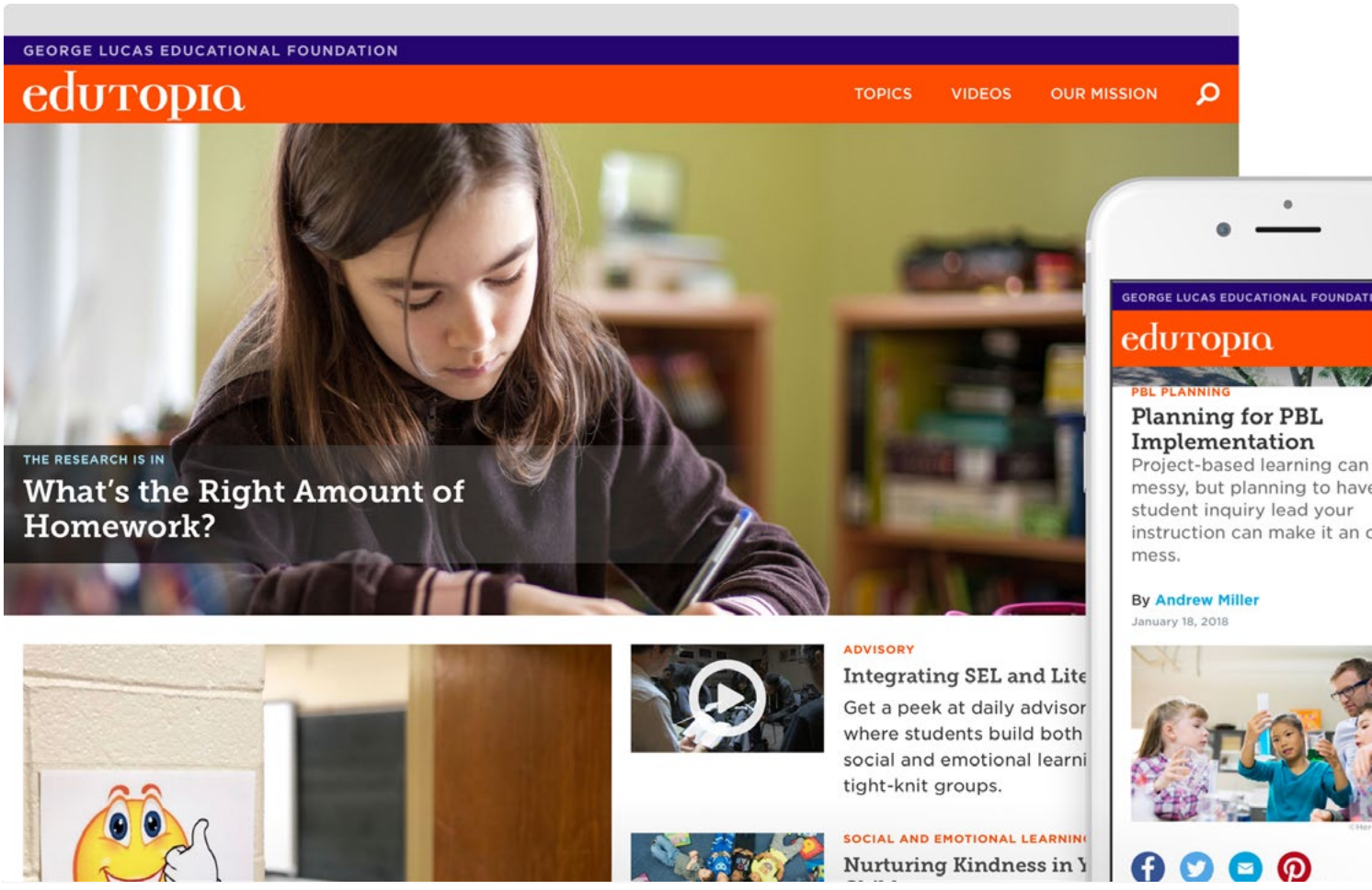
Hotwire

The travel site Hotwire has a fully decoupled hotel search service. We architected and implemented a Drupal platform to provide API data to a front-end JavaScript application. Searches for hotels were handled with a legacy system, but Drupal provided editors with a way to add and manage supplementary metadata about the retrieved hotel, such as descriptions and photos. This data was then augmented into search results via a REST API.



Edutopia

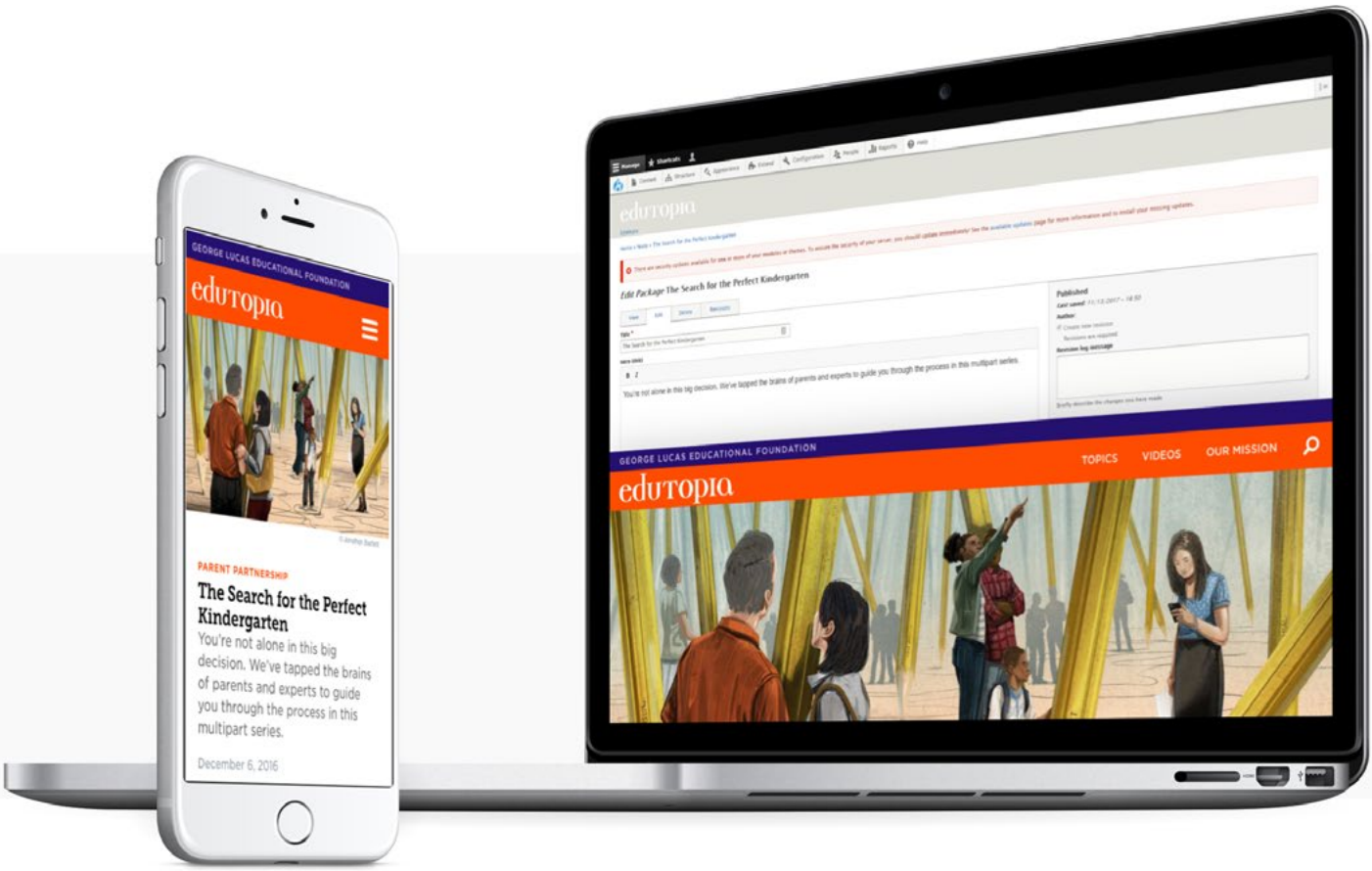
The nonprofit George Lucas Educational Foundation was created by George Lucas to identify and spread innovative, replicable and evidence-based approaches to helping students from preK to grade 12. As usage on mobile devices and apps continue to drive expectations, the Foundation wanted more control over the structure and performance of the front-end user experience. At the same time, they needed Drupal, a mature CMS with a strong API foundation, to serve content.



Previews can be problematic in decoupled applications, and the Edutopia editorial team requires real-time previews of content as it goes through the production process. We built a system to author or revise content in Drupal and then view it in their front-end React application with the click of a link. Landing pages can also be previewed, allowing editors to experiment with potential layouts and different combinations of components before making them live.

We also shifted from loading full HTML pages with every click to loading only the data that is needed for an application. A Redux datastore holds the page data in the browser so that when a page is accessed more than once, the page load is instantaneous. This helps to create a snappy feeling for the website and opens up new options for the future, such as offline reading and personalization.

READ THE FULL CASE STUDY

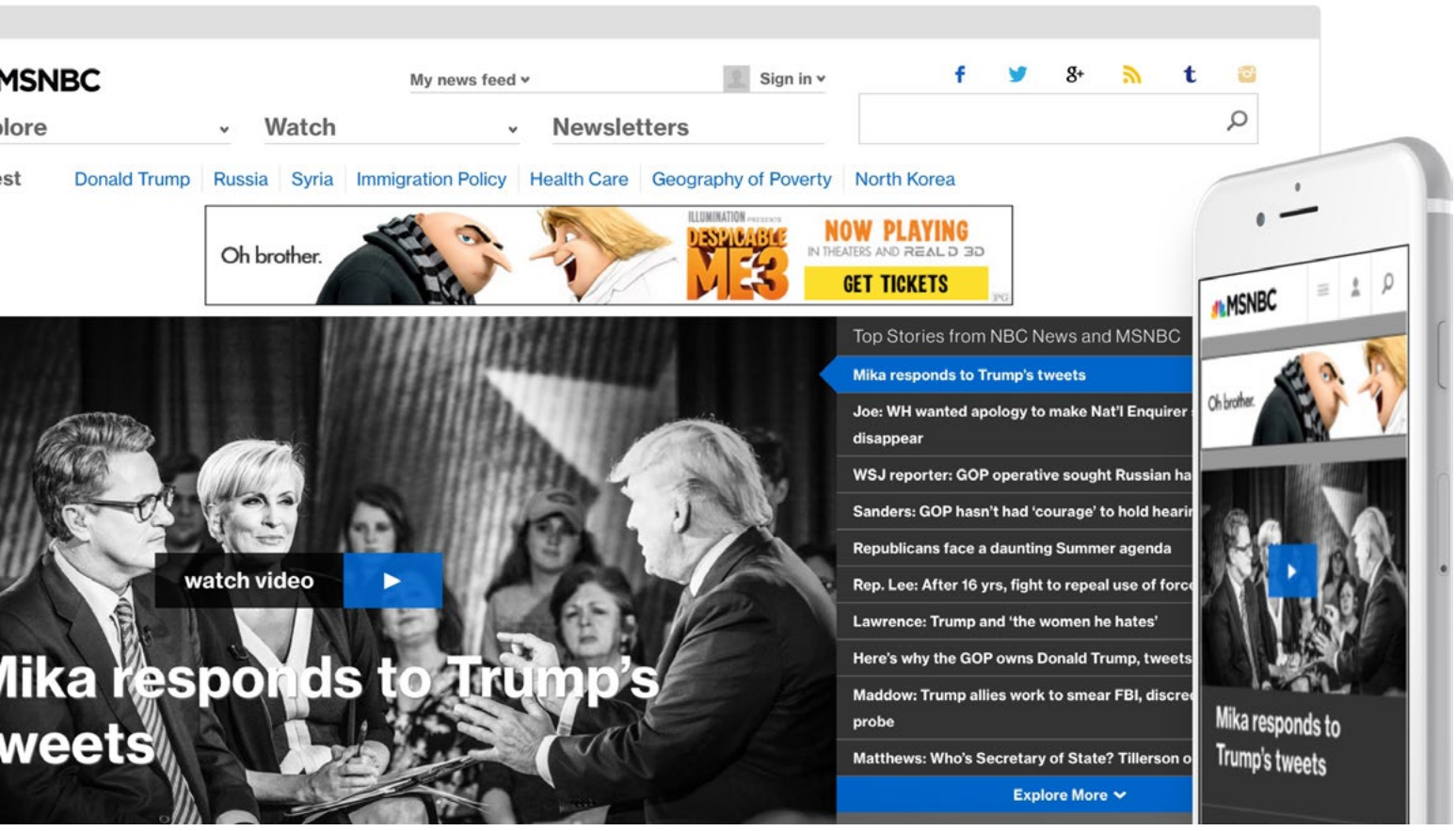


MSNBC

Although MSNBC used Drupal 7 to render the basic framework of its pages, there was a massive amount of third-party user data and user-generated content that needed to be woven into the Drupal pages. To accomplish this, third-party content was retrieved by client-side calls to the external API, then rendered in the browser.

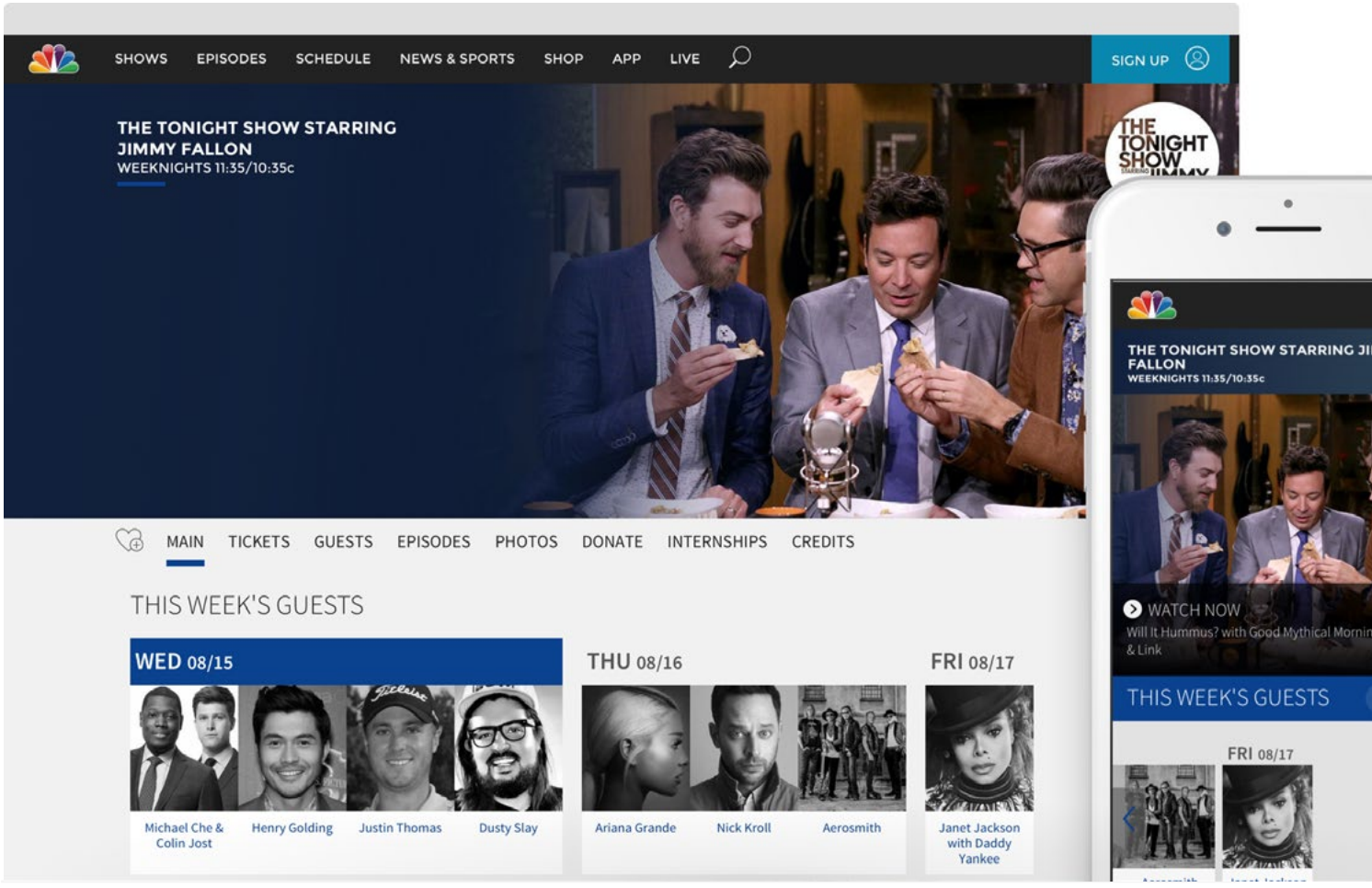
This was done by creating AngularJS blocks which could be placed anywhere on the site. These blocks contained the initial HTML/Angular template markup and JavaScript to populate themselves with the relevant remote data. Some blocks allowed users to dynamically select categories and otherwise filter the results from the remote services, while many others incorporated paging or scrolling and other interactions.

READ THE FULL CASE STUDY



The Tonight Show

Another decoupled project was a website for the then-new host of The Tonight Show, Jimmy Fallon. This project consisted of a distributed multi-national, multi-company team of Drupal, Node.js, Backbone, and editorial experts that created the decoupled site. We discussed how the project was built at DrupalCon, and a video for that presentation is available online. More information about The Tonight Show project is available in an article on Lullabot.com entitled Internal API Design for Distributed Teams.



Authored by Karen Stevenson, Director of Technology at Lullabot

Still unsure if decoupling is right for you?

Hopefully, this white paper is insightful and useful as you consider whether or not decoupled Drupal is what your business and your team needs. We know it can be a little overwhelming, so if you have any questions, please reach out! We're always happy to provide the answers you're looking for to make life easier for you.

Additional resources are provided on the following page.

About Lullabot

Lullabot is a strategy, design, and Drupal development company that has created some of the most high-profile and award-winning websites for large-scale publishers. As one of the first Drupal agencies, Lullabot is highly recognized for their body of work, authentic approach, and leadership in Drupal innovation, having contributed to more than 150 modules. Lullabot clients include NBC Universal, Martha Stewart Living, Syfy, Hotwire, GE, Principal Financial Group, Harvard University, and Verizon.

Phone: 1-877-585-5226 // **Email:** hello@lullabot.com // **Website:** www.lullabot.com

LULLABOT RESOURCES

- Beyond Decoupling, The Inherent Virtues of an API - a discussion of the importance of API design in a successful decoupled project.
- Decoupled Drupal Hard Problems: Routing - there are many tendrils that begin when we separate our routes and paths from a more traditional Drupal setup, especially if we need to think about routing across multiple different consumers.
- Decoupled Drupal Hard Problems: Image Styles - Our HTTP API serves an unknown number of consumers, but we don't want to expose all image styles to all consumers for all images.
- The Hidden Costs of Decoupling - Decoupled Drupal has been well understood at a technical level for many years now. While the implementation details vary, most Drupal teams can handle working on decoupled projects. But there are additional costs.
- Should you Decouple?
- Drupal JavaScript Initiative: The Road to a Modern Administration UI
- Will JavaScript Eat the Monolithic CMS?

OTHER RESOURCES AND TOOLS

Drupal core is enabling this activity through a couple of core initiatives:

- The API-first Initiative, which is focusing on providing the APIs needed for alternative front ends and other consumers of Drupal's content.
- The JavaScript Modernization Initiative, which is working on creating a decoupled JavaScript-based editorial interface.

Drupal and the Drupal community have numerous tools available to assist in creating a decoupled site:

- Contenta, a pre-configured decoupled Drupal distribution.
- Waterwheel, an emerging ecosystem of software development kits (SDKs) built by the Drupal community.
- JSON API, an API that allows consumers to request exactly the data they need, rather than being limited to pre-configured REST endpoints.
- GraphQL, another API that allows consumers to request only the data they want while combining multiple round-trip requests into one.