How Can Advanced Drupal Multisite Management Transform Your Content Sharing Game?

Quick summary

Navigating the world of Drupal multisite configurations can be both challenging and rewarding. As Drupal developers, it’s essential to understand the nuances of managing content across various microsites. This comprehensive guide delves deep into the options available for Drupal website

development, enriched with practical use cases and advice, to help you make informed decisions for your complex Drupal projects.;

Introduction:

Navigating the world of Drupal multisite configurations can be both challenging and rewarding. As Drupal developers, it's essential to understand the nuances of managing content across various microsites. This comprehensive guide delves deep into the options available for Drupal website development, enriched with practical use cases and advice, to help you make informed decisions for your complex Drupal projects.

Understanding Drupal Multisite:

A Drupal multisite setup allows multiple websites to operate under a single Drupal installation, each with its themes and modules. While this offers tremendous flexibility, it also raises the question: How can we efficiently manage and share content across these diverse environments?

Let's explore the various content sharing options at our disposal:

1. Entity Share for Drupal Multisite

- What is Entity Share? — It is a collection of modules to syndicate content between Drupal websites in the client-server model. It equips JSON API for content sharing and offers UI to manage shared entities.
- It allows content sharing in one-to-many and many-to-many scenarios.
- The content hosted on the server website can be shared with the client website, provided they comprise the same structure (machine name, field machine name, and storage) with separate codebases and databases.
- The server website supports recursive pull/update requests by the client website. It's worth noting that, for now, the server website doesn't support push requests from the client.
- Following are some of the content entities that can be shared via Entity Share among Drupal site setups with client-server architecture patterns:
    - Nodes
    - Taxonomy terms
    - Medias
    - And other entities.
- Companion Module:
    - Entity Share Async (sub-module)
    - Entity Share Diff (sub-module)

- Entity Share Lock (sub-module)
- Entity Share Cron
- Entity Share Websub
- ECA Entity Share

Use Case: Some of the most common and widely used use cases for the Entity Share module are content sharing between staging and production environments, aggregating the content, and sharing taxonomy.

Consider a scenario with multiple educational websites requiring content sharing between the websites. Entity Share can efficiently distribute course materials and updates among multiple educational websites, leveraging the expertise of offshore developers.

Flowchart: Content sharing via Entity Share across sites.


Drupal Website
Server
Drupal Website 1
Client 1
Drupal Website 2
Client 2
Drupal Website 3
Client 3

Pros and Cons : While user-friendly and efficient for compatible Drupal versions, version incompatibility between client-server setups can be a significant hurdle.

Pros:

- The module is actively maintained and is used by 3000+ websites.
- The client site can pull data from an approved list of shared content.
- Maintains a versioned copy of the shared content and facilitates the identification of content changes.
- Facilitates Multilingual content sharing.
- Automatically creates referenced entities based on UUID during content/entity pulls.
- Provides an easy-to-use UI interface for pulling/synchronizing content.

Cons:

- The Entity Share module doesn’t support custom workflows as it retains the original state (published/unpublished) of the pulled content from the main/server website.
- It ensures content exposure with proper authentication.
- It affects the revision history of the node after pulling an edited entity on the client end; changed timestamps mess up the client-side revision history.
- Entity Share provides an additional layer of security by allowing administrators/editors to verify data before synchronization.
- It needs developers to step in. Subscribing to specific events is a must to change content states during pulls on the client site. Customizing functionality might require a developer’s touch.
- The module encounters challenges in displaying variations in reference fields within the revision (diff) module.

Tips:

- Regularly update the Entity Share module to ensure compatibility with the latest Drupal versions.
- Clearly define the use cases before implementing Entity Share. Understand the specific content-sharing needs and workflows within the Drupal site.
- It affects the revision history of the node after pulling an edited entity on the client end; changed timestamps mess up the client-side revision history.
- Monitor the performance of Entity Share, especially in scenarios involving frequent content pulls and synchronization. Optimize configurations

for efficiency.

2. Scalable Drupal Views Configurations for Drupal Website Development

What is Drupal Views? — It is a versatile module that can address the need for content sharing while offering extensive flexibility. It fetches and displays content from the site's database.

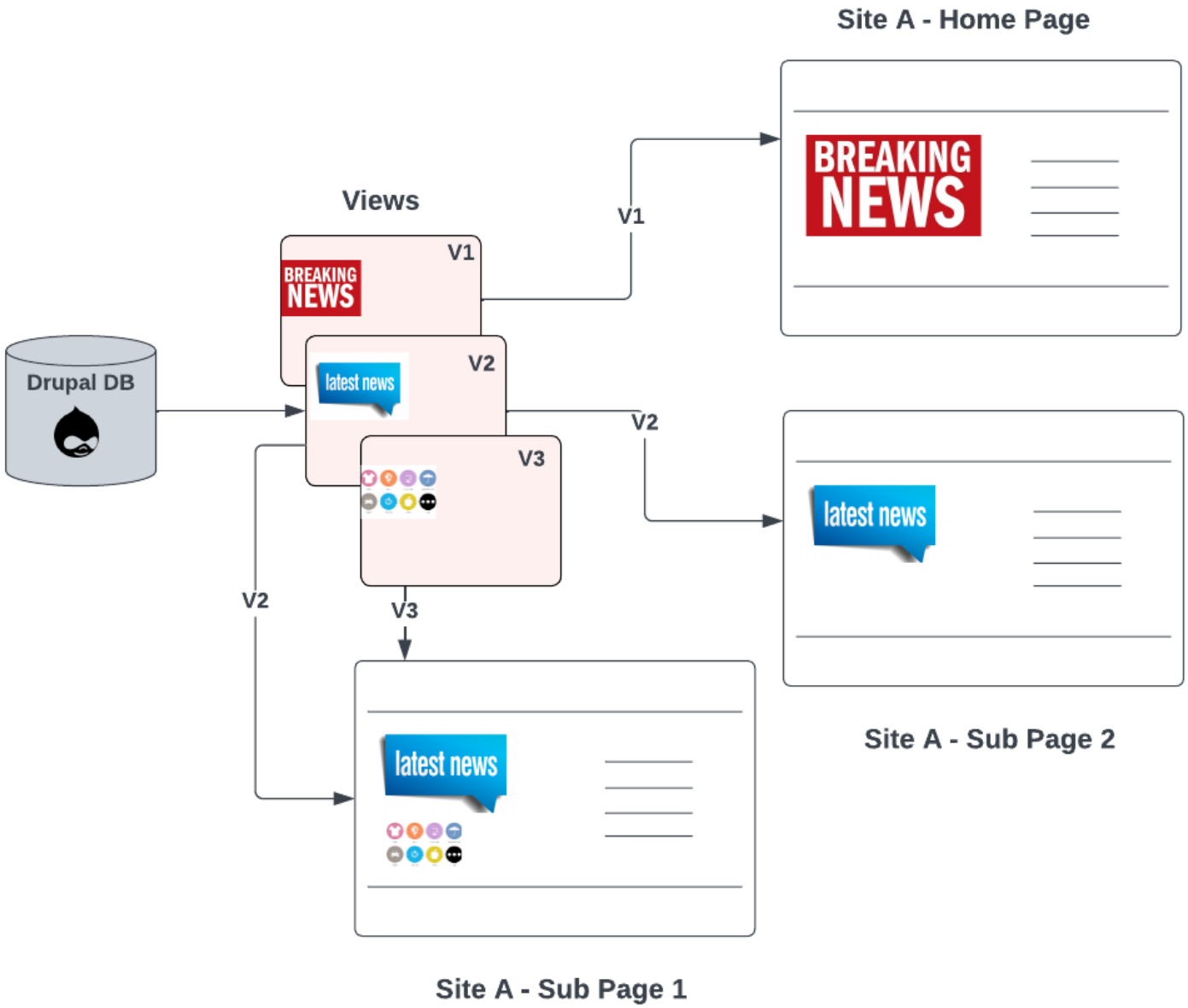
- It provides a user-friendly interface and supports creating lists, posts, galleries, and more.
- It outperforms direct queries with caching and offers various display options, making it a go-to tool for presenting dynamic content on the Drupal site.
- It enables the creation of various content displays, offering extensive flexibility.
- By default, Drupal Views are specific to the Drupal instance.
- Format: Drupal Views serves data in various formats based on configuration and requirements.
    - **HTML**: Primarily serves content in HTML format for webpage rendering.
    - **RSS Feeds:** Can provide data in RSS feed format for syndication.
    - **JSON/XML**: Supports JSON or XML output to integrate with other systems or create web services.
- Export and import views between Drupal instances with Drupal's configuration management system to reuse views across different instances. It's essential to ensure that both instances have consistent content types, fields, and other dependencies for the views to function correctly.
    - To do this:
        - In the Drupal admin interface, navigate to Configuration -> Development -> Configuration synchronization.
        - Export the view configuration, which generates a YAML file.
        - Using the same interface to import the view configuration on the target Drupal instance.

Use Case: Imagine you run a news website using Drupal. With Drupal Views, you can create a "Latest News" page that dynamically pulls and displays headlines, images, and summaries from articles. You customize the layout, filter by categories, and even add a sidebar block for "Featured Stories." Drupal Views provides the ability to present and organize content dynamically.

Flowcharts:

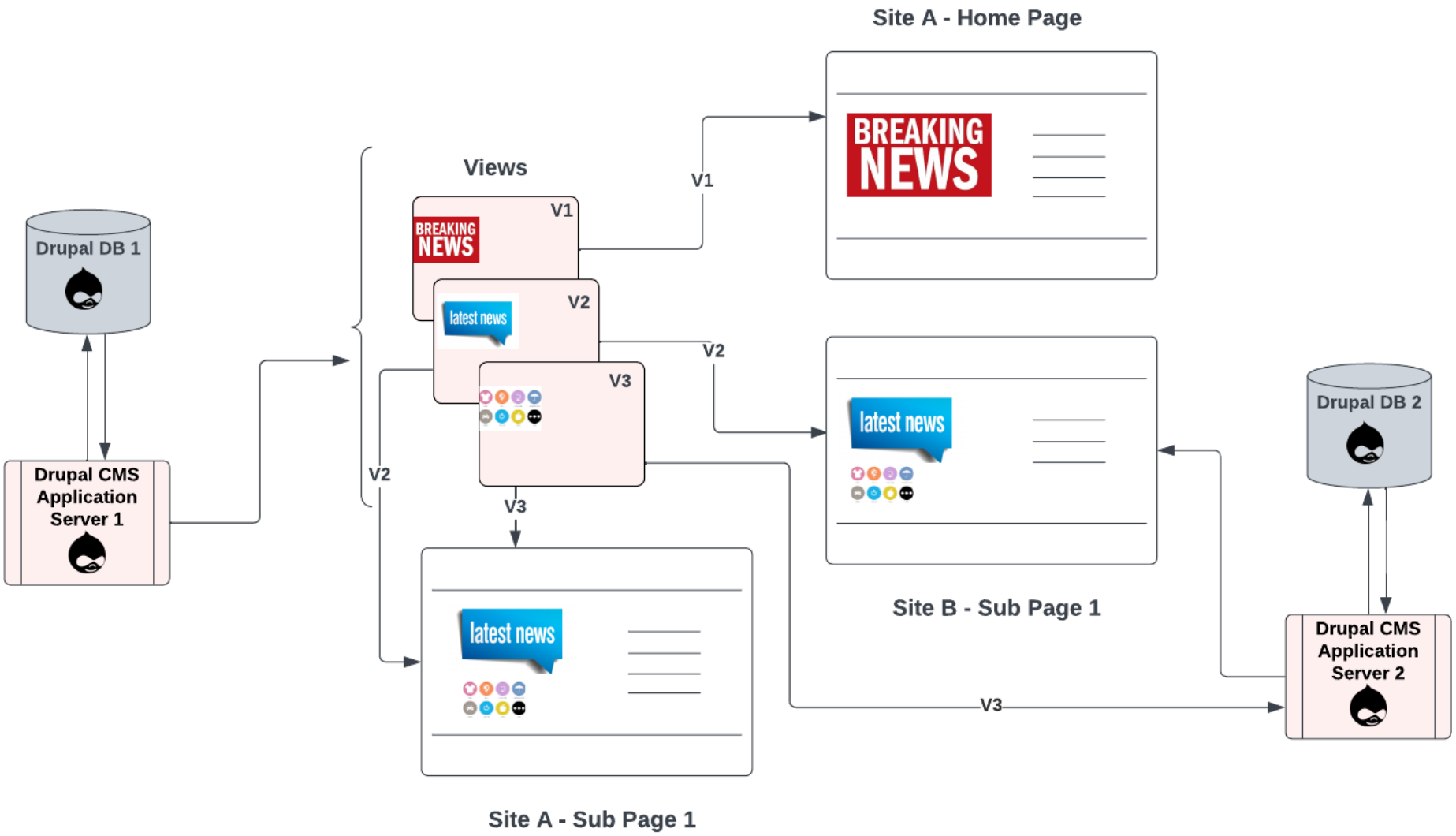
Scenario 1: For Single Drupal Instance Content Sharing via Drupal Views

- Utilizing Drupal Views within a site (Site A)


Site A - Home Page
Views
V1
V2
V3
Drupal DB
BREAKING NEWS
latest news
V1
V2
V2
V3
Site A - Sub Page 2
Site A - Sub Page 1

Scenario 2: For Multiple Drupal Instance Content Sharing via Drupal Views

- Utilizing Drupal Views for communication between multiple sites (Site A and Site B):
  - **Site A:**
    - Multiple views are created and shared within the site.
  - **Site B:**
    - Frontend Drupal view sharing.
    - Backend Drupal view sharing and integration with Site B’s configuration settings.



Pros and Cons: Offers excellent flexibility and performance but can be complex to manage during major Drupal updates.

Pros of Drupal Views:

- Versatility:
  - It creates diverse content displays like lists, tables, galleries, etc.
  - Adaptable to different content types, including nodes, users, and custom entities.
- User-Friendly Interface:
  - Views UI provides an intuitive, graphical interface for easy configuration.
  - Reduces the need for coding, making it accessible to a broader audience.
- Dynamic Content Presentation:
  - It enables dynamic content updates without manual intervention.
  - It supports contextual filters for personalized content displays.
- Performance Optimization:
  - It implements caching layers for improved performance.
  - Optimizes database queries, enhancing overall site speed.
- Integration Capabilities:
  - It integrates seamlessly with other Drupal modules.
  - It supports export/import configurations for ease of deployment.

Cons of Drupal Views:

- Learning Curve:
  - It may have a learning curve, especially for complex configurations.
  - Advanced features require a deeper understanding of Drupal’s architecture.
- Performance Considerations:
  - Improperly configured views can impact site performance.
  - Large datasets may require careful optimization to avoid resource-intensive queries.
- Security Considerations:
  - In specific configurations, exposed JSON: API endpoints might pose security risks.
  - It requires diligent security practices, especially for sites handling sensitive data.
- Complexity for Novice Users:
  - The novice users may find advanced features overwhelming.
  - Misconfigurations can lead to unintended results.
- Customization Limitations:
  - While powerful, some particular customizations may require additional development.
  - Advanced theming may be necessary for unique design requirements.

Tips:

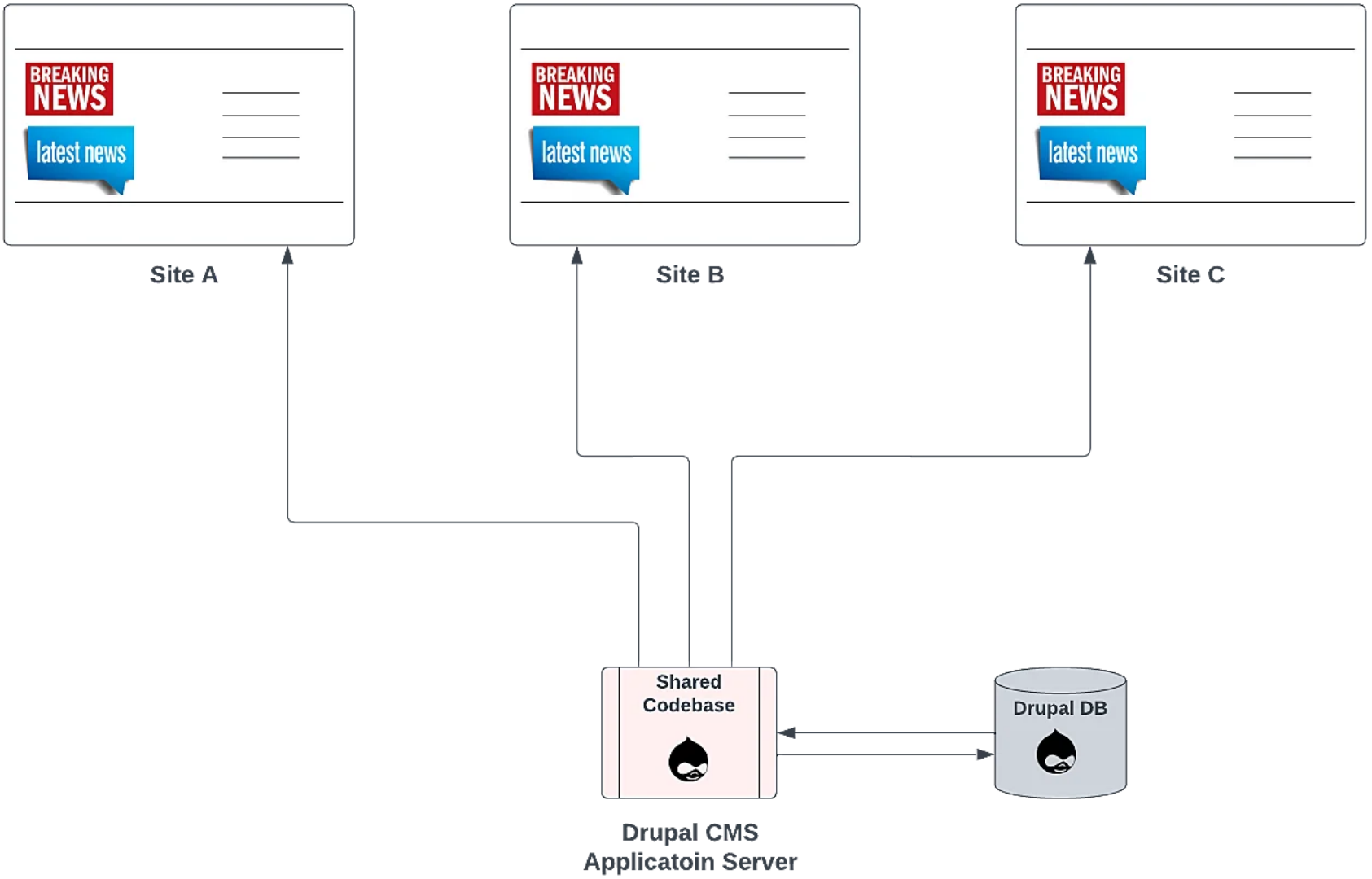
- Utilize the caching feature of Drupal Views to enhance site performance.
- Optimize queries to enhance performance. Use the Views UI or SQL previews to inspect generated queries, identifying and addressing potential bottlenecks.
- Keep abreast of module updates and enhancements. Views evolve, and newer versions may offer improved features, security patches, and optimizations.

3. Domain Access Module in Drupal with Domain Source:

- What is Domain Access? — Drupal provides a suite of modules to manage multiple websites (domains) within a single Drupal installation.
  - It has a single shared database.
  - The approach allows us to share users, content, and configurations across a group of sites.
  - With Domain Source Module: It helps to achieve more flexibility and control over how different domains share or isolate content and configurations.

Use Case: Good for scenarios where you want to run several websites on different domains but share Drupal codebase and database.

Flowchart: Content Sharing via Domain Access with Domain Source



Pros and Cons:

- **Pros:**
  - **Simplicity:** Straightforward setup and configuration, making it easier for users who prefer simplicity.
  - **Integrated Solution:** A bundled set of modules designed to work seamlessly together.
  - **Community Support:** A well-known project, it often benefits from community support and updates.
  - **Domain Source:** Adds more flexibility and customization options with fine-grained controls.
- **Cons:**
  - **Less Customization:** Limited flexibility for highly customized scenarios; might not accommodate unique requirements.
  - **Complex Requirements:** You may need help with intricate setups where detailed control over domain-specific configurations is necessary.
  - **Dependency on Project Development:** Relies on ongoing development and updates from the Domain Access Project maintainers.
  - **Domain Source:**
    - **Potential for Misconfigurations:**There is more room for misconfigurations due to the extended customization options, which may impact the system.

Tips:

- **Define Specific Requirements:** Clearly outline the project's requirements for managing multiple domains. Understand the level of customization and control needed over content, configurations, and user experiences.
- **Choose Domain Access Project if :**Opt for Domain Access Project if the goal is to manage multiple domains with a straightforward setup and you don't require highly customized configurations.
- **Perform Thorough Testing :** Conduct thorough testing on a staging environment before deploying Domain Access configurations to live sites.

4. Custom API Development in Drupal via Drupal Core API:

- What is Custom API? — A tailored approach using Drupal's Core API, this method suits organizations looking for flexibility, complete control, and customization.It comes in headless, decoupled, and hybrid approaches, enabling content sharing across

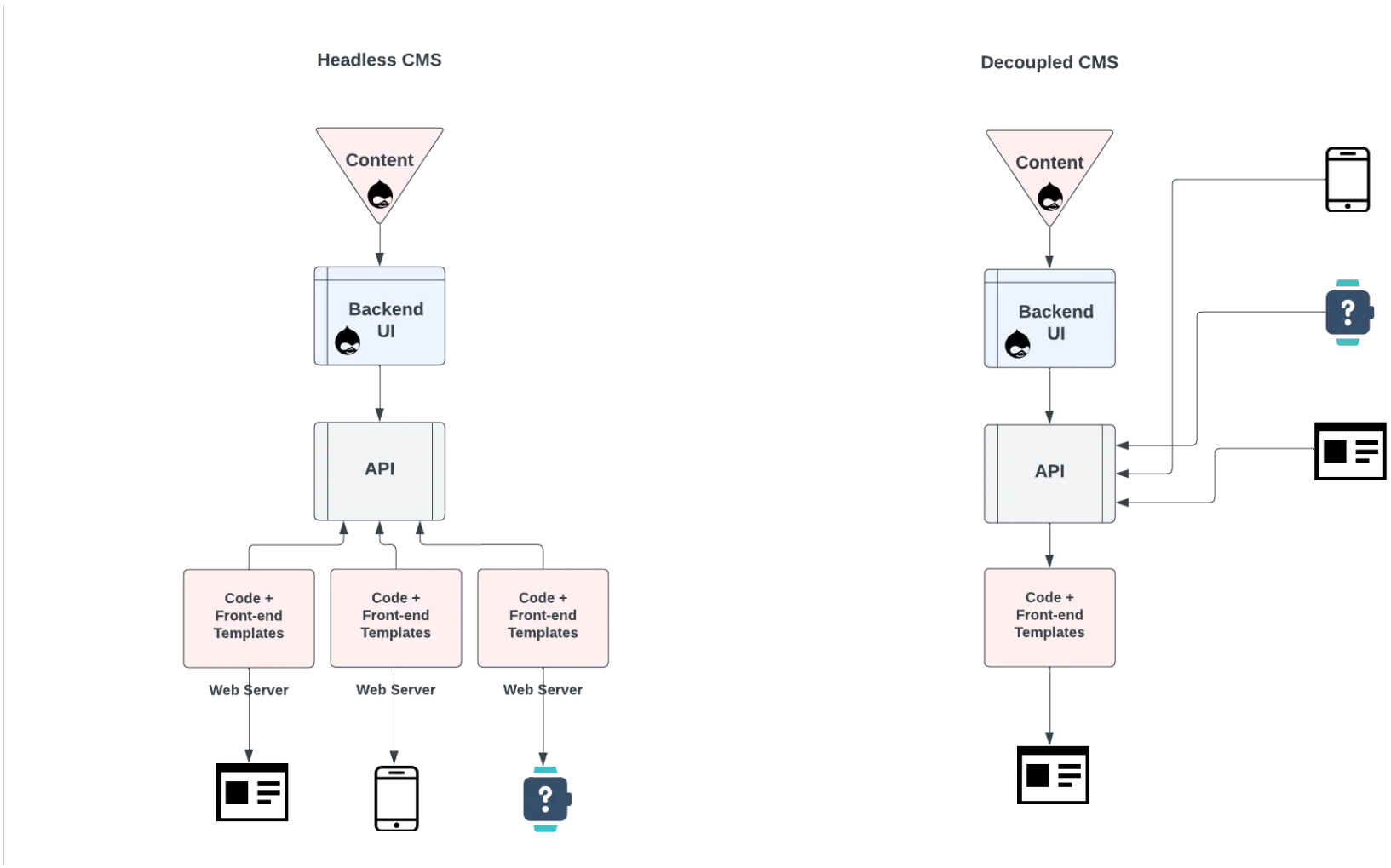
various platforms and devices, a boon for offshore developers tackling diverse project requirements.

- There are many ways to go about depending on the project requirement and flexibility needed:
  - **Headless**: In a headless Drupal architecture, the back end (where content is managed) is separated from the front end (where content is displayed to users).
    - **Implementation**: The Drupal back end serves as a content repository and exposes content through web services like JSON/ REST API. The front end is built using a separate technology, such as a JavaScript framework (e.g., React, Vue.js), which communicates via API with Drupal's back end to fetch and display content.
    - **Content Sharing Benefit**: Headless Drupal makes it easier to share content across different platforms and devices. The content can be accessed and displayed by various applications or websites, not just the traditional Drupal-powered front end.
    - **Advantages:**
      - Offers maximum flexibility in choosing front-end technologies.
      - Enables independent development and scalability of the back end and front end.
  - **Decoupled:** Decoupled Drupal involves separating the back end from the front end.However, connections or interactions between the two layers might still exist in a decoupled pproach, such as an optional front end.
    - **Implementation:** While the front end is decoupled and can be built with different technologies, there may be some dynamic elements or features powered by Drupal's presentation layer. This can be achieved through API calls or other methods.
    - **Content Sharing Benefit:** Decoupled Drupal offers flexibility by allowing the use of different front-end technologies while still leveraging Drupal's content management capabilities. Content sharing can occur seamlessly, with the front end fetching data from Drupal's back end.
    - **Advantages:**
      - Offers flexibility for using different frontend technologies.
      - Allows for independent development and scaling of the back end and front end.
  - **Hybrid:** This approach incorporates traditional (monolithic) Drupal and decoupled Drupal elements. It balances the benefits of a fully integrated system and the flexibility of a decoupled architecture.
    - **Implementation:** Certain sections or components of the website may be built with a traditional Drupal front end, while others are decoupled.This approach can compromise content authoring capabilities and flexibility in front-end technologies.
    - **Content Sharing Benefit:** Content sharing in a hybrid approach allows for a mix of traditional and decoupled components. This can be useful when specific site sections require the dynamic capabilities of a JavaScript framework (e.g., React, Vue.js) while other sections remain tightly integrated with Drupal's presentation layer.
    - **Advantages:**
      - Allows for a gradual transition from a traditional to a decoupled architecture.

- Preserves the benefits of a fully integrated system for certain sections while providing flexibility for others.

Use Case: An organization wanting to share its Drupal content with the web (different layouts), mobile apps, IoT devices, or other digital platforms.

Flowchart illustrating the API-based content sharing process



Pros and Cons: Highly flexible and powerful, but requires significant technical expertise and resources.

- **Pros:**
  - A Custom API allows you to design and implement endpoints tailored to the project's specific needs..
  - Utilizing Drupal Core API ensures integration with the broader Drupal ecosystem, leveraging the platform's strengths.
  - The API provides a unified data source, making managing and retrieving content and data from a single point

easier.
    - You have complete control over the data structure and format, allowing for customization based on the application's requirements.
    - Drupal Core API includes built-in security features and authentication mechanisms, ensuring secure data access.
    - Drupal's scalability extends to custom APIs, allowing for growth and increased demand for data access.
- **Cons:**
    - Developing a custom API may introduce additional development overhead, especially if the requirements are complex.
    - Custom APIs may have a learning curve, requiring developers to understand Drupal's API structure and best practices.
    - Custom APIs may pose maintenance challenges, mainly if there are frequent updates or changes in project requirements.
    - Depending on the implementation, there could be potential performance overhead associated with custom API calls compared to out-of-the-box solutions.
    - Creating a custom API tied to Drupal means a level of dependency on the Drupal platform. If there's a need to decouple completely, this may pose challenges.
    - There's a risk of over-engineering if the custom API includes optional features for the project, leading to unnecessary complexity.

Tips: Consider a headless Drupal setup if the project demands extensive cross-platform integration.

- **Standardization:** Consistent coding standards and practices enhance a development team's readability, maintainability, and collaboration.
- **Documentation:** Comprehensive documentation facilitates onboarding for new developers and ensures team members understand how to interact with the custom API..
- **Versioning:** API versioning ensures backward compatibility while allowing for future enhancements without disrupting existing implementations.
- **Security Measures:** Security is paramount in any API implementation. Protect against common vulnerabilities, such as injection attacks, and ensure data privacy.
- **Error Handling:** Robust error handling enhances the user experience and helps developers troubleshoot issues effectively.
- **Testing:**Rigorous testing ensures the reliability and stability of the custom API across various scenarios.
- **Scalability Considerations:** Design the API with scalability in mind to accommodate future data and user base growth.

Further Considerations:

- **Security:** When sharing content across sites, always prioritize security. Regularly update Drupal installations and modules.
- **Performance:** Consider the impact of content sharing on site performance. Efficient caching and database optimization are crucial.
- **Scalability:** Plan for future growth. Choose a solution considering the current needs and accommodate potential expansion.

Summary and Recommendations:
In summary, each content-sharing option comes with its unique set of features and considerations. For simpler site networks, Entity Share and Drupal Views are excellent choices. Domain Access or a Custom API approach may suit more complex, large-scale environments. Assess your team's technical expertise, the specific needs of the project, and long-term maintenance and scalability before making a decision.

For top-tier Drupal development, explore the services offered by August Infotech. Discover insights into offshore development companies and their expertise in white-label web development.