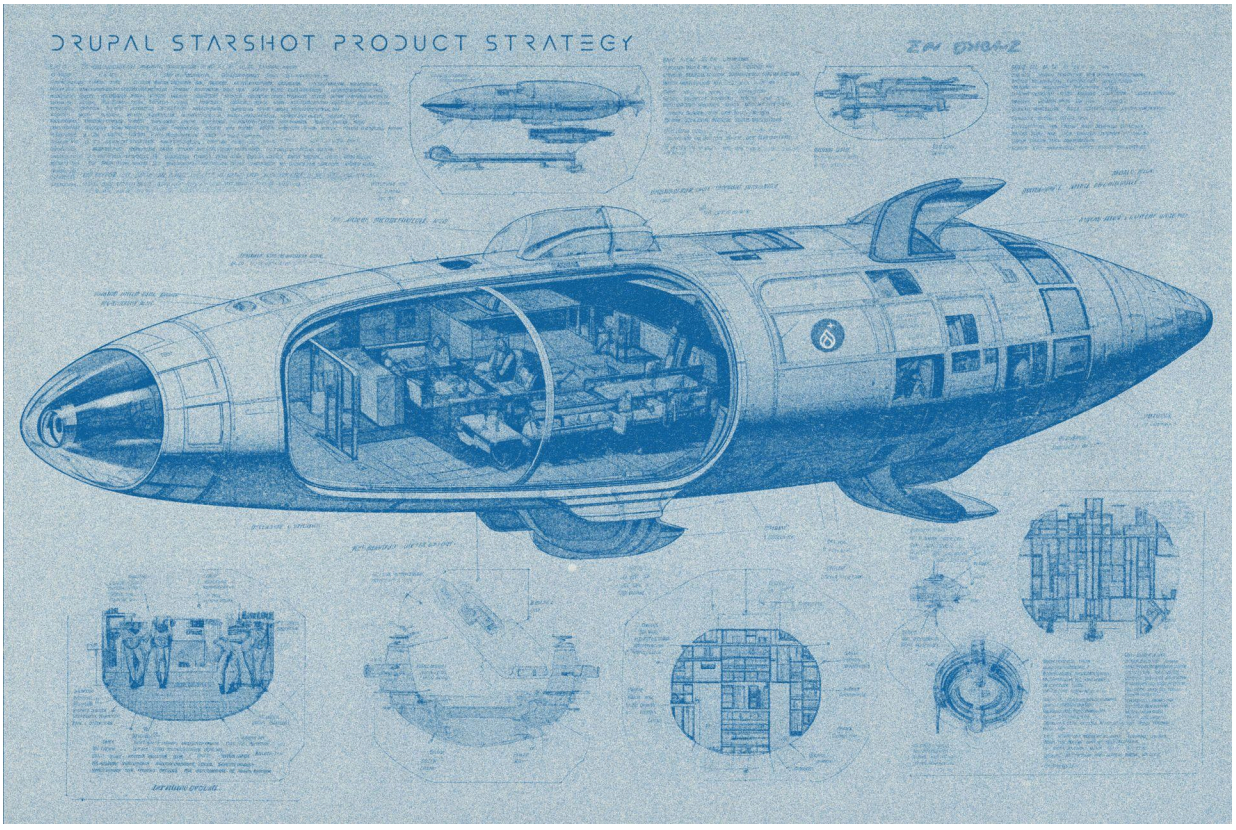
Proposal: Advanced Search

Drupal CMS

Authors (alphabetically)

- 1x: Baddý Sonja Breidert
- 1x: Christoph Breidert
- 1x: Artem Dmitriiev

Last review date: 2024-08-18



Requirements

This document outlines a proposal for the search functionality included in Drupal CMS.

The target audience[1] for this proposal includes the following personas:

- **Ambitious Marketer**: Possesses some technical knowledge but primarily seeks user-friendly marketing tools.
- **Ambitious No-Code Developer**: Technically skilled, with a preference for configuring applications rather than writing code.
- **Ambitious Developer**: Holds advanced or expert-level proficiency in coding and working with web frameworks.

Note, that the developer persona has been included to clearly define the audience for this proposal.

The personas outlined above have the following requirements for search functionality in a Drupal installation:

- **Ambitious Marketer**: Desires the ability to independently configure multiple search functions (e.g., site search, event search, product search) without developer assistance. They also want to create engaging and dynamic search experiences.

[1] Compare https://dri.es/introducing-drupal-starshot-product-strategy

- **Ambitious No-Code Developer**: Shares the same goals as the ambitious marketer but is also capable of handling more complex tasks. They understand how to extend search functionality and can collaborate effectively with developers.
- **Ambitious Developer**: Requires intuitive configuration options and clear extension mechanisms and APIs to easily enhance search capabilities.

This proposal aims to address these requirements with a comprehensive solution.

Approach

Given that the data structure for the content is still under development, our approach is based on the current available information.

To provide a standard search experience, we've included only nodes, focusing on their properties: title, type, and rendered content. This setup facilitates a keyword-based search that prioritizes results with the keyword in the title, while still enabling the discovery of nodes where the keyword appears within the content. Additionally, search results can be refined by type, allowing for more precise filtering through facets.

Delivering a simple yet well-configured search function in Drupal CMS is an effective starting point. As the content structure evolves, this search functionality can be easily extended by marketers, no-code developers, and developers alike.

For selecting the appropriate modules, we conducted an extensive review of various resources. Our primary consideration was the number of installations of the selected modules on Drupal.org. We also evaluated case studies, award-winning Splash-Award projects, blog posts, and podcasts.

Our research consistently shows that the ***Search API***[2] and related modules are the best choice for creating engaging search experiences in Drupal. We found no significant evidence of Drupal core search functionality being used in ambitious projects.

However, we acknowledge the growing popularity of external search services like Algolia, Coveo, and SearchStax. These services typically require developers to build JavaScript-based search applications that interact directly with the search service. Such applications offer complete customization and extensibility for developers. A key advantage of these services is their compatibility with headless Drupal solutions, where the frontend consumes content from a Drupal API, and the search functionality is powered by the external search service's API.

To reinforce Drupal's value proposition as a ***Headless CMS***, we have also explored integrating headless capabilities into our advanced search proposal.

[2] Compare https://www.drupal.org/project/search_api

Proposal

The proposal is split into frontend and backend. Based on the same backend recipe different frontend recipes can be installed.

Backend

We propose leveraging the *Search API* and ***Facets***[3] module family to configure search functionality in Drupal CMS. These modules have become the de facto standard for advanced search solutions in Drupal.

Search API allows for the indexing of any content into various search backends (e.g., Database, Solr, Elastic). It is fully configurable, enabling precise control over what data is indexed, with support for multiple search backends and indexes. Both the input (data indexing) and output (data retrieval) processes can be modified. The entire functionality is accessible through the UI and can also be extended via API.

Facets enables the creation of dynamic search filters that allow to refine searches and narrow the search results

Drupal CMS should include a standard configuration for a default site search, complete with a facet for available content types and a configuration for autocomplete using the module ***Search API Autocomplete***[4].

The configuration will be provided as a recipe (***Search backend***), and the details of the recipe are described below.

This out-of-the-box setup of the backend will allow the target personas to explore and modify the default search or replicate its configuration to create customized searches.

Frontend

The primary challenge for search in Drupal CMS is providing a versatile frontend.

A common option is to build frontends using a combination of *Views*, *Exposed filters*, *Blocks*, and *Ajax* from Drupal's core. However, solutions built with this setup require a lot of knowhow of Drupal's internal mechanics in order to create or modify searches and it can be cumbersome to add styles and Javascript functionality to enhance a default display.

Due to these limitations, many have turned to external search services like Algolia, Coveo, or SearchStax. These services typically require developers to create Javascript-based search applications that interact directly with the search service. Such applications can completely be

[3] Compare https://www.drupal.org/project/facets
[4] Compare https://www.drupal.org/project/search_api_autocomplete

modified and extended by developers. However, this approach is unsuitable for the target audience because it excludes ambitious marketers and no-code developers.

To overcome this 1xINTERNET contributed a module called ***Search API Decoupled***[5], which exposes the full functionality of *Search API*, *Facets,* and *Search API Autocomplete* as an easy to use endpoint. Furthermore, a Javascript library called ***Search API Decoupled Client***[6] was contributed to provide an out-of-the-box search experience.

Search API Decoupled ships with a UI sub-module to place search widgets such as search input, results, pagination, facets, autocomplete, and search summaries using the layout feature from Drupal core. For each of these widgets implementations are provided in the library from *Search API Decoupled Client*.

By utilizing these modules, ambitious marketers and no-code developers can create flexible and responsive search experiences, while ambitious developers can extend the search functionality with custom widgets (e.g., proximity facets, hierarchical facets, custom result displays, advanced pagination) or even build standalone search applications. Examples with videos showing the functionality can be found in this blog post.

A positive side effect of this approach is that the search configuration can also be reused when installing a future headless setup in Drupal CMS.

For the frontend we propose to create two recipes that build on the backend recipe for search:

- ***Views search frontend***
- ***Javascript search frontend***

Recipes

Below recipes will be created that provide the proposed functionality. The recipes will be kept up to date with Drupal CMS.

Search backend

A single recipe for the search backend will be provided.

Search API

Server

- Name: Database (database)
- Server configuration based on Database backend (as it doesn't require any additional dependencies).

[5] Compare https://www.drupal.org/project/search_api_decoupled
[6] Compare https://www.drupal.org/project/search_api_decoupled_client

Index

- Name: Default (default)
- Nodes of all content types will be indexed.
- The following fields will be indexed:
    - Title
    - Content type
    - Rendered node (view mode *Search index* from core rendered for anonymous users)
- Keyword search will be configured for the title field and the rendered node with view mode *Search index*.

Note: That the view modes *Search index* and *Search result highlighting input* must have sensible fields visible. Changing this configuration will not be part of the recipe. However, the recipe will activate the view modes.

Facets

- A facet for content types (the bundle) will be created.

Views search frontend

A recipe that depends on the search backend recipe will be provided for a traditional search frontend.

Views

- Exposed filter for search input
- Exposed filter for search facet
- Content with pagination
- Search page (`/search`)

Javascript search frontend

A recipe that depends on the search backend recipe will be provided for a decoupled search frontend.

Search API Index

- Rendered node (view mode *Search result highlighting input* from core rendered for anonymous users)

Search API decoupled

- Configuration of endpoint (`/api/search/default`)

- Configuration of searched and returned fields
- Configuration of query parse mode

Search API decoupled UI

- One column layout
- Search input widget
- Search summary widget
- Facet widget
- Search result widget
- Pagination widget
- Static search route enabled (`/search-api-decoupled/default`)

Search API decoupled client

The client itself has no configuration, the Javascript library is pulled with composer.json when the module is installed.

Results

Upon installing Drupal CMS with both frontend recipes enabled, two similar looking search pages with the same functionality will be available.

A search database and index configuration will be available.

All functionality is minimal and can easily be copied, changed, or extended.

Contribution

We need to contribute the following functionality to *Search API decoupled client*:

- Add missing implementation of configuration options.
- Fix pagination bug
- Add configuration for a search page

Timeplan

- Create concept / Specification document (CW 33)
- Present concept to track leads and collect feedback (CW36)
- Create recipes (CW 33 - CW 35)
- Create test environment for review (CW 35 - CW 36)
- Create community survey and collect feedback (CW 37 - CW 38)
- DrupalCon CW 39