Front-end delivery: a Symfony application to deliver web pages

Overview

A Symfony front-end application will deliver the working web pages, the “frontend”. This application reads content in from Craft, which acts as a Headless CMS, and other data sources as required. Content is passed to the template layer, and web pages are rendered via the Twig templating system.

This work will be led by Studio 24, with support from W3C Systems Team and dedicated time from Jean-Gui to assist with work on the Symfony front-end application.

The front-end application work is divided into:

- Application architecture planning
    - Fetching data via GraphQL and other data sources
    - Integration with existing Symfony “Main” application
    - Review previous open source code
    - Caching strategy
    - Hosting architecture
    - Managing redirects
    - Review preview content solution
    - Serving images on production
- Setting up of environments and deployment scripts
- Symfony application architecture setup
    - Setup base application
- Twig template integration
    - Meta tags
    - Multilingual content
    - Icons
- Routes, controllers, and views
- Other front-end functionalities
    - Posting new comments
    - Logged in/out state

Managing work between Studio 24 and W3C

This work is led by Studio 24, though W3C will be available to assist with planning, reviewing decisions and undertaking development work alongside the Studio 24 team.

Communication will be undertaken on Basecamp as normal, with any decisions copied into this specification document.

Work tasks will be managed in JIRA, the standard agile task management tool used by Studio 24 on projects of this size. It is strongly recommended any W3C staff working on development work during this phase of the project also use JIRA. Studio 24 can make users available on our commercial JIRA account.

Ideally we'll have regular touchpoints, either daily standups or meeting regularly each week, to ensure the team working on the front-end integration work knows what we're all doing and can easily answer any questions quickly.

Application architecture planning

This work will be undertaken by Studio 24, assisted by W3C.

Fetching data via GraphQL and other data sources

Data populating the W3C website comes from multiple sources. The front-end application needs to fetch data using various Rest APIs but also using GraphQL when fetching content from the Craft CMS.

We need to ensure the right content is exposed via GraphQL and how authentication works for queries that need to post data (e.g. submitting comments via blog page).

We also need to identify data sources for other data, e.g. members. These data sources will be managed by W3C.

Integration with existing Symfony "Main" application

In this phase of work, there are two Symfony applications that power W3C web pages:

- Existing **"Main"** Symfony app (pages such as TR, Account pages, Groups)
- New Symfony **"Frontend"** app (new pages with content from Craft CMS such as Homepage, Business Ecosystems, News)

The current plan for the Beta site is to develop new pages in a new Symfony “frontend” application.

However, a lot of existing functionality already exists in an older Symfony application called “Main” which due to time constraints it is not possible to fully migrate to the new “Frontend” application in this phase of work.

We are therefore taking a pragmatic approach. Work in this phase is focussed on the new “Frontend” app. W3C will also need to make updates to the existing “Main” app to update the template and update functionality as required (for example to update how the TR listing page works).

The long-term aim is to use APIs to pull content into the frontend app and deliver all web pages via this application.

Integration points

This means we need a few integration points between the two Symfony apps, with the aim of reducing duplication where possible.

CSS

This can be loaded directly from the “Frontend” application.

Twig template

This may have to be copied between the two applications. We plan to use Symfony Bundles for this.

Common content

E.g. Primary navigation, Footer.

We propose to generate static versions of this common content in the “Frontend” application which can be read in via the “Main” application (and any other system that needs it).

This will be available via HTTP via a URL (e.g. w3.org/en/fragments/navigation.html) and can be accessed via HTTP requests. We recommend local caching to help minimise requests.

Review previous open source code

Studio 24 have undertaken headless CMS projects in the past, we have a range of open source code we maintain to assist building these projects. We will review what

code we can re-use to help increase efficiency within this project and how well that works with GraphQL. Where we need to add features we'll do this in the open source project which can then be used in the W3C project without any impact on the project budget.

Caching strategy

Full-page caching

By default we assume HTML pages delivered by the front-end system are fully cacheable and shared by all users. This means all public users see the same HTML web page content, including HTTP headers. Caching public pages increases performance dramatically and is the recommended solution wherever possible for w3.org web pages.

Pages are also assumed to be stateless and functionality should not be based on serving pages from one server (since production will be run from multiple webservers).

We have techniques for full-page caching in Symfony, though we recommend using Varnish Cache since this is available.

Any personalised content, therefore, needs to be either excluded from full-page caching (e.g. account section) or we use JavaScript solutions to generate personalised content (e.g. sign in / my account navigation).

Caching strategy

We need to confirm the caching strategy, caching lifetime and what personalised content is required.

Clearing the cache

We also need to explore how to clear the cache on content updates. It has been suggested using tags to help clear content. See FOSHttpCacheBundle.

API caching

In previous projects we have also cached API requests to increase performance. However, with the use of Varnish Cache this is not recommended since it increased complexity for little gain.

Hosting architecture

See Hosting architecture.

Review preview content solution

In previous Headless CMS projects supporting a preview for page content that is not yet published is challenging and not easily possible.

Craft CMS does support draft entries in the GraphQL API, so this is easier to achieve for the W3C site. This work involves reviewing how to retrieve draft entries and how we can display these on the front-end application so CMS editors can preview content from the CMS.

This may require authentication on the front-end and a small custom plugin in Craft to update preview links to point to the front-end site. See Craft CMS docs.

Serving images on production

The production site will run on multiple webservers, therefore, we need a strategy for serving images from one static location. This will need reviewing with W3C.

Setting up of environments and deployment scripts

This work will be undertaken by Studio 24, assisted by W3C.

- Shared GitHub repo (led by W3C)
- Setup of deployment script (led by W3C)
- Branching strategy for different environments
- Continuous Integration (review security, code standard, automated tests that can be setup, review Symfony Insights). Ideally use GitHub Actions.
- Explore whether it's feasible to set up integration testing (e.g. Behat)
- Setup of required environments

Environments and URLs

Expected URLs at this time are:

- Development
    - www-dev.w3.org
    - Development site intended for testing code during development
    - Use Deployer for deployment

- Staging
    - www-staging.w3.org
    - Staging site intended to test & QA web pages and functionality by the client before they go live
    - Use Deployer for deployment

- Production
    - beta.w3.org - production URL - the live site
    - To be replaced with www.w3.org on full launch
    - W3C to setup deployment (likely based on Puppet)

Disabling search indexing of non-production site

Development and Staging environments must set the X-Robots-Tag header to “noindex” to ensure pages are not indexed.

While Production is in Beta this must also set the noindex header.

Preview URLs

We may need “preview” URLs setup depending on how we manage viewing draft content in the CMS on the front-end. This will be confirmed in the architecture setup work.

Symfony application architecture setup

This work will be undertaken by Studio 24.

Setup base application

This part of work requires an outline application architecture to be created, to enable future work. At least one page URL should render content from Craft CMS as part of this work.

Twig template integration

This work will be undertaken by Studio 24.

Templates of the Design system that are necessary for delivering the views listed in the routes, controllers, and views section above will be converted into Twig templates.

The Twig templates will be built according to the Twig coding standards and the DRY principle to ensure productivity, maintainability and quality of the templates.

Meta tags in the document head

The following metadata will be populated with data when building the views:

- Document title

- Meta description
- OpenGraph metadata
- Twitter metadata (where not covered by OpenGraph data)
- Robots metadata
- Home link
- Canonical URL link (we will review whether we should point to www.w3.org from beta site for canonical links)

Reference:
https://developer.twitter.com/en/docs/twitter-for-websites/cards/guides/getting-started

Internationalization on the front-end

Localized messages

Some text on the website will not be managed in the CMS, e.g. pagination text “Next page.”

We recommend the use of language files (e.g. YAML) to manage different translated messages for use across the site. Messages need to be identified by short, clear keywords. We do not recommend using the actual English text as the message key, since this can cause confusion if the text subsequently changes.

In Symfony these are managed via the Translation component.

Also see Mozilla’s Localization content best practices.

Page language attribute

HTML documents generated by the front-end app will bear a language attribute with a value that corresponds to the language of the current page content.

In-page translated content language attribute

Sometimes, a piece of content in another language than the main language of the page will be displayed. In that case, a language and direction attribute will be added to the HTML element that contains that piece of content.

Links to translated content

Any links to translated content will have the hreflang attribute.

Bi-directional templates

Similarly, all HTML documents will have the direction attribute set to ‘rtl’ or ‘ltr’ (right-to-left or left-to-right) depending on the current language of the page. The CSS rules applicable to the current language direction will take effect automatically.

Links to localized websites

Visitors of the website will see a choice of languages to switch between localized websites. When clicking a language link, the visitor will be redirected to the homepage of the website in the target language.

Icons

Icons of the Font Awesome 5 icon collections will be used. When an icon is picked in the CMS, its slug will be used as its identifier. In the front-end app, we will need to implement a Twig function that:

- Takes a Font Awesome 5 identifier slug
- Fetches content from the corresponding svg file
- Adds an optional <title> tag to it
- Adds optional classes to it top-level <svg> tag, and its <path> element

Routes, controllers, and views

This work will be undertaken by Studio 24 and W3C. Along with templating, it forms the bulk of the front-end integration work to create working web pages.

Here is the list of routes that need to be covered and processed by the front-end application.

Homepage

- Display page content

Data sources: Craft CMS, its content need to be fetched using its GraphQL API + other sources?

View: TBC

Pages

- Display page content

Data source: Craft CMS, each page need to be fetched using its GraphQL API

Views: There are two types of page templates: the default page template, and the landing page template

Business ecosystem pages

- Display page content

Data sources: The bulk of the content needs to be fetched from Craft CMS using its GraphQL API. Pieces of content are also fetched from other sources (TBC):
    - Groups
    - Member organizations
    - Testimonials?
    - Evangelists
    - Champions

View: Each business ecosystem entry uses the business ecosystem template.

Blog

View page

- Display page content
    - Display comments (Nice-to-have is to display W3C user avatar, W3C can provide an API endpoint to provide this based on email address)
    - Comments form
    - Submit comments form to Craft CMS (which needs to use spam filtering)
- Possible use of microdata to mark up content (see schema.org Blog)

Data source: Craft CMS

View: Blog articles follow the blog template.

Listing page

- Listing page to display paginated results of content (ordered by most recent)
- Listing page to display paginated results of content (by year)
- Listing page to display paginated results of content (by category)
- Data feed (RSS 2.0) for listing pages

Data source: Craft CMS

View: Blog listings implement the news listing template, in which posts are listed from most recent to oldest.

Search page

- Search content by keywords (match against title and/or content)
- Display listing page with paginated results

Data source: Craft CMS (note: this includes comments related to individual blog articles)

View: Blog search implements the news listing template, in which posts are listed in order of relevancy.

News

View page

- Display page content
- Possible use of microdata to mark up content (see schema.org NewsArticle)

Data source: Craft CMS

View: News articles each follow the blog template.

Listing page

- Listing page to display paginated results of content (ordered by most recent)
- Listing page to display paginated results of content (by year)
- Listing page to display paginated results of content (by category)
- Data feed (RSS 2.0) for listing pages

Data source: Craft CMS

View: News listings follow the news listing template, in which posts are listed from most recent to oldest.

Search page

- Search content by keywords (match against title and/or content)
- Display listing page with paginated results

Data source: Craft CMS

View: News search implements the news listing template, in which posts are listed in order of relevancy.

Press releases

View page

- Display page content
- Possible use of microdata to mark up content (see schema.org NewsArticle)

Data source: Craft CMS

View: Press release articles each follow the blog template.

Listing page

- Listing page to display paginated results of content (ordered by most recent)
- Listing page to display paginated results of content (by year)
- Data feed (RSS 2.0) for listing pages

Data source: Craft CMS

View: Press release listings follow the news listing template, in which posts are listed from most recent to oldest.

Events

It is intended to house all event content within the events section, which includes talks, workshops, conferences, meetings.

View page

- Display page content
- Possible use of microdata to mark up content (see schema.org Event)
- Possible download event as an iCal file (data about a single event)

Data source: Craft CMS

View: Events articles each follow the event entry template.

Listing page

- Listing page to display paginated results of content
- Current and future events are listed in chronological order according to their start date from closest in the future, to furthest in the future.
- Past events are automatically listed in an archive, which list events in reverse chronological order from most recent to oldest.
- Listing page to display paginated results of content (by year)

- Listing page to display paginated results of content (by category)
- Data feed (RSS 2.0) for listing pages

Data source: Craft CMS

View: Events listings follow the events listing template.

Staff Alumni

A page listing W3C staff alumni.

Data source: The Craft CMS

Views: A single page using the People listing template

Newsletters

Newsletters are intended to house simple HTML content to be sent out via the current W3C newsletter system.

Content is generated by a script managed by W3C which pushes content to Craft via the API. Content in Craft for newsletters would need to be edited in HTML only.

View page

- Display page content

Data source: Craft CMS

View: A very plain HTML document which does not use the main W3C template. Global CSS will not be applied to newsletter.

This would require some inline CSS styles to be created in order to allow W3C to send an HTML email alongside a text email. We'd need a simple bare bones HTML template for email sending.

Listing page

- Listing page to display paginated results of content (ordered by most recent)

Data source: Craft CMS

View: Default template, displaying a simple list of newsletters.

400 and 500 errors

400 and 500 errors will be set to return error views in keeping with the styles of the W3C website and present the user with a generic message. Content on these pages will be editable via Twig templates.

Common error pages we need to account for:

- 403 - Not authorised (cannot login)
- 404 - Page not found
- 410 - Gone (examples: https://www.w3.org/2001/03/webdata/xsv and https://www.w3.org/2017/11/tidy)
- 50x - Server error

Other front-end functionalities

Posting new comments

Users can leave comments on blog posts. The front-end app needs to process comment posting by users. This involves creating a custom commenting form, processing submissions and pushing the new comments to the Craft CMS using GraphQL mutations.

If a user is already logged-in then their name and email should be automatically captured for the blog comment form. Ideally this data would be read via JavaScript from the user menu AJAX endpoint.

Logged in/out state

Some page content will be personalised to users who are logged in to w3.org, namely the ‘My account’ link in the page header.

As we are implementing full-page caching, this content cannot be rendered by the PHP scripts of the Symfony application.

A JavaScript snippet will be created to make an AJAX call to a W3C application via a specific API endpoint. The request will return a JSON object bearing all the user and account information required to create the markup of the ‘My Account’ link.

W3C will develop the AJAX endpoint to return user data for this feature.

The AJAX endpoint will ideally contain data for:

- Current user email
- Current user name
- Current user avatar image URL
- User menu items (label & link)

,

Gravatar-like URLs

To display avatars next to blog post authors and commenters an endpoint will be needed to retrieve the avatar URL from an email address or username, similarly to what Gravatar does, but with W3C accounts instead.

W3C will develop the AJAX endpoint for this feature. We recommend Varnish caching is used on this endpoint to improve performance.