camptocamp

INNOVATIVE SOLUTIONS
BY OPEN SOURCE EXPERTS

GeoMapFish

Analysis Report

GMF3 Evolution

Version 1.0
18.02.2021

Table of Contents

Goal of This Document

Most of the NGEO/GMF components and services are based on the AngularJS framework which is reaching End Of Life at the end of December 2021.

The goal of this document is to explore and evaluate the alternative for the future of the client side of the GeoMapFish solution.

The backend side of the solution, notably all the layers administration and permissions, will be kept.

Impact of AngularJS reaching End Of Life

What is AngularJS?

AngularJS is a framework. It allows us to:

- Create UI components and declare their inputs and outputs
- Create JS services and instantiate them
- Wire services/values to components and services
- Proxy some browser features through services (http, location, ...)

AngularJS does not:

- Handle neither the CSS (gmf2 uses the latest version of bootstrap for that) nor the HTML itself, as a consequence there is no breakage to fear on new browser versions due to the obsolete AngularJS (contrary to what happened with ExtJS)
- Build the application (gmf2 relies on webpack for that)

Impact of EOL

- Limited regarding the stability of the platform: AngularJS relies on standard Javascript which will be supported by all browsers forever;
- High regarding the new investissements: creating new features on top of this obsolete technology is increasing technical debt.

Analysis of the Current Version of GMF2

Main Features

Appendix 1 lists the main features of GeoMapFish version 2.6.

Technical Insights

The client part consists of several parts:

- NGEO components and services;
- GMF components and services, often built on top of NGEO;
- GMF apps (desktop, mobile, ...) which are production-ready geoportals;
- API library for integration in foreign websites;
- Translations and build tools.

Since the beginning of the project in 2014, the code base has received continuous effort:

- Migration to the ES6 module standard;
- Migration to the de-facto standard bundler Webpack;
- Isolation of the functionality into modules;
- Migration from Less to the Sass CSS preprocessor used by bootstrap v4;
- Early use of the CSS variable standard to allow dynamic styling;
- Migration to the latest stable OpenLayers version;
- Migration to the latest stable jQuery version;
- Migration of AngularJS to the latest supported version and component architecture.

In a nutshell, the code base is in a very good state, except for the AngularJS dependency which was abandoned by Google and has become a legacy.

Possible Improvements

- Stop using non-essential features of the AngularJS framework (typically the http service, location service, ...);
- Drop entirely the AngularJS framework and port to another technological ground;
- Simplify code base: for example by merging the NGEO and GMF components and services together.

Statements of Requirements

The user groups expressed the need to remove the dependency on the obsolete AngularJS framework, while keeping all the existing functionalities.

Requirements

Functional

- Swiss local projection
- Application localized in 4 languages (en, fr, de, it)
- Same level of ergonomy
- Same level of performances
- Similar level of configurability by client projects

Technical

- The chosen dependencies should be supported on the long run, and widely used.
- Good test coverage
- Minimum browser: last 2 versions of Chrome, Firefox, Edge, Safari + Firefox ESR (no IE11)

Desirable Changes in the Configuration Possibilities

Ability to define extension points in HTML allowing use of Web Components provided dynamically by the client project (without recompilation of the GMF code).

Evaluation Criteria

In addition to matching the specified requirements, the solution will be evaluated according to:

- The cost of the migration
- The cost of maintenance
- The impact on current build tools (webpack, docker) / integration in client applications
- The ability to do an incremental migration
    - Creating production-ready releases during the migration

Ability to prioritize migration of some components before the others (to allow evolving them)

- Perenity of the new technical foundation

  Ideally 7-10 years support

  Community

  Ease to work with and to test

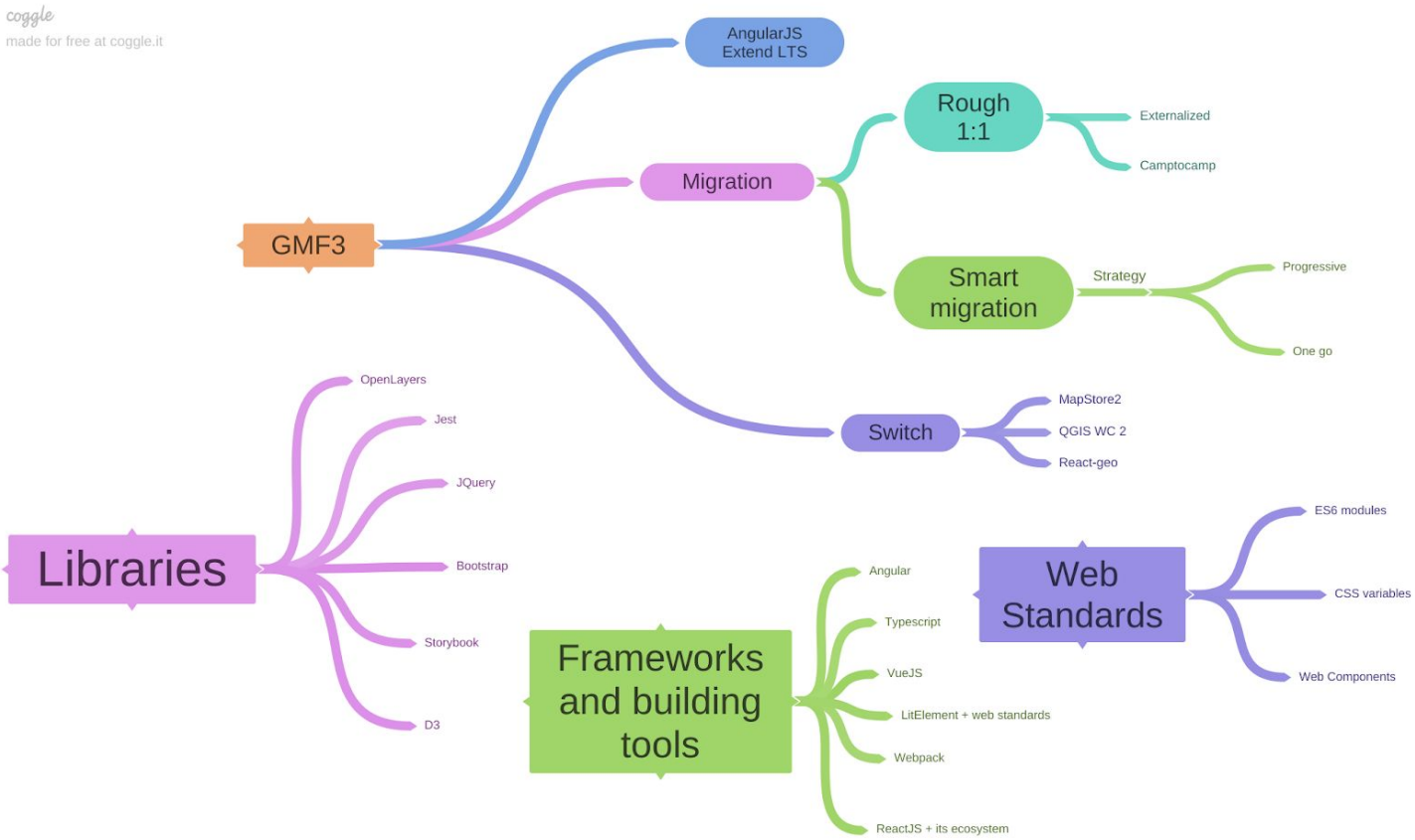
  Acceptance by users

  Maintenance of the old major versions

Evolving GMF2

Overview of the Possibilities

There are many possibilities to evolve GMF2. Some can be grouped. The diagram below shows some possible scenarios and choices.



Scenarios

Stay with AngularJS for the core

XLTS for AngularJS is an option to keep your application running and supported. It's not free, but compared to the cost of a migration or rewrite, it's an easy decision. XLTS.dev currently plans to provide support for AngularJS through at least 2026-2031. That's a lot of extra time.

Inside this scenario, it is still possible to do some evolution of the code:

- Replace non-necessary AngularJS dependencies (http service, location, ...)

- Add new components exposed as standard Web Components
- Add pure JS services
- ...

Rough 1:1 migration

This scenario considers the most basic strategy to migrate GMF to a new framework. It emphases on the short term cost and rapid deadlines, without simplification of the architecture. The migration would be a 1:1 migration (code, tests, examples and apps), it will be a pure mechanic work to ensure iso functionality. Externalizing such migration to reduce the cost would be possible (but bring risks as a counterpart), as it requires the least knowledge about GMF.

Depending on the chosen target framework, the time necessary to integrate GMF3 inside a GeoMapFish application will be more or less long. This work will need to be done by a GMF expert.

Progressive migration on new technologies

In this scenario, the migration would be progressive, the application will stay operational and continue to be deployed to production, there will be GeoMapFish releases coming regularly.

- It will be possible to continue to invest in new functionalities all along the migration process. For each release, the user group will have the possibility to influence which code modules need to be migrated;
- The AngularJS services will be rewritten to pure JS classes;
- Migration of the UI components to standard Web components;

In addition, this scenario will simplify the architecture, notably by merging the GMF and NGEO code where it makes sense. For example, the layertree code is very complicated in GMF2 due to the unnecessary use of the ngeo layertree component as a building block.

Notable elements of the target architecture

- The GMF code, containing only GeoMapFish specific code
- The isolated generic JS code, eventually developed and shared with others outside the project (ex: upstream in OpenLayers or in https://github.com/geoblocks/)
- Strongly typed (typescript), focused on API stability (document all breakage)
- Some framework or library (see comparison)
- Controlled extensibility (extension points are added to Views and Controllers)

- Catalog of components (using storybook or a simpler alternative)

Switch to another technical platform

In this scenario we create the application based on an existing web map client product. Here are the candidates:

- Mapstore2
- QGIS web client
- React-geo

This direction would quickly demonstrate a basic and generic working application.

Adding GMF specifics and customisable features may however be long and challenging.

Comparison Matrix

	Stay on AngularJS	Rough 1:1 Migration	Progressive Migration on New Technologies	Switch to Existing Client Library
Advantages	Ensure LTS with current solution	GMF users group are the owners of the solution and have full control over it. Best possible result / customizability if chosen framework allows it. Capitalize on existing code/knowledge. Best reactivity in case of issue. Fit to the GeoMapFish backend. The underlying architecture would stay the same. Actual GMF in a very good state for migration: modules, webpack, AngularJS 1.8 The Framework upgrade should be easier. Cheapest and fastest migration Could be externalized.	GMF users group are the owners of the solution and have full control over it. Best possible result / customizability. Capitalize on existing code/knowledge. Best reactivity in case of issue. Fit to the GeoMapFish backend. The underlying architecture would stay the same. Possible progressive migration and intermediate feature additions. Actual GMF in a very good state for migration: modules, webpack, AngularJS 1.8 The Framework upgrade should be easier.	Project with own and independent lifecycle Existing community / fundings Ready to use components / features Based on more modern technologies than AngularJS. MapStore 2: already extensible through plugins system QGWC 2: good integration with QGIS server (like printing)
Disadvantages	Keep developing in a deprecated technology. Increase technical debt. Will make future migration harder. Less and less people know the AngularJS framework over time	No refactoring, no improvement of actual architecture Tunnel effect Won't ensure the best code quality Keeps ngeo/gmf segregation Difficult to integrate	In case of progressive migration : period of cohabitation with different technologies.	Need to migrate existing configuration in admin interface (layer tree, metadatas) Difficult to customize as we really want Almost zero capitalization of the existing

		Migration will be hard		Dependency on another community We should think again all the application Backward compatibility very difficultQGIS web client and Mapstore2: Required to rewrite a part of the backend QGIS web client: really linked to QGIS Server (need more details) Not so big code base Based on React16, not the latest MapStore 2: complex due to compatibility with multiple client map libraries
Opportunities	Take more time to consider another solution	Port our actual code base to modern framework	Choose the best technical options available in 2021	Could quickly get a functional application Join an existing community
Risks	Private LTS stops suddenly	Some components can't be ported 1:1 due to technology switch. Regressions. Migration of everything else but code could be painful (because we don't decide of the whole architecture). Risk on the integration. We remain closely linked to the framework, version migrations may be painful depending of the chosen framework.	Risks related to technical choices, but relatively predictable since the architecture will be similar as GMF2	Difficult to collaborate with the existing community (owned by only one other company). Ngeo components have advanced functionalities that may be harder or impossible to reimplement on top of an existing solution (snapping, requests grouping optimization, enumerations support in filtering, and other layers's metadatas based features, etc). Really hard to evaluate the cost

				of porting existing features or adding new features
Cost	Very low	Medium Code migration can be very predictable if done externally. Integration of the migrated code needs to be done by GMF experts.	High Predictable from experience and https://xlts.dev/blog/2021-01-15-the-math-of-migrating-from-angularjs	High to very high to get full featured applications like Ngeo essentially because we can't capitalize on the existing. Harder to predict
Horizon	Now	1-2 years	2-4 years (can be flexible, depends on the strategy, progressive or not)	2-3 years

Conclusion

The end of official AngularJS Long-Term Support is really something to consider, but not to be afraid of. First, there is no hurry, the LTS ends in December 2021, and it could be extended until at least 2026 by private companies (which are officially recommended by Google). Nevertheless, the AngularJS framework is a legacy, and it would be beneficial to the project to migrate from it.

GMF is not the first application that needs to transition out of AngularJS, this transition has been experimented by thousands of projects and the literature is globally positive about it. We have tools to estimate such a transition, and an excellent point with GMF is that the code base is really healthy and is already in a perfect position to concretise such a transition: the code is spread in modules and components, we already use webpack, we are already using the last the 1.8 version of AngularJS, what really increase the chances to be successful. All this context really enforces the choice to do a migration over to switching to a completely different solution. By evolving his own solution, the GMF user group keeps its full control over it and maximizes the reuse of the experience and knowledge that has been built and gathered during GMF2 development.

Switching to an entirely new codebase is possible and a good solution if the user group is open to important compromise; otherwise it may be brittle, complex and time-consuming to reimplement the same level of customization as the user group enjoys with GMF2.

We presented 4 realistics scenarios, with all their pros and cons, trying to give as much material as possible to help you to take the most appropriate path.

In order to decide the technical stack and architecture of the solution, we need to do some POCs. There are many possibilities and we will need new inputs from the user group to know which POCs are the most valuable.

The next steps looks to be:

- Think about these canonicals scenarios inside the user group
- Share new inputs
- Test the most appropriate technical stacks and architecture through POCs
- Decide the best solution

Appendix 1 - GeoMapFish 2.6 Features

Feature	Sub-feature	Desktop	Mobile
Navigation	Zoom&pan	X	X
.	Map rotation	2.3	2.3
.	Geolocation	2.5	X
.	Info bar	X	-
Full-text search	Data	X	X
.	Layers/groups/themes	X	-
.	Coordinates	X	X
Data	Background selector	X	X
.	Themes (flush)	X	X
.	Themes (non-flush)	X	2.3
Layers	(Un)select layer/group	X	X
.	Clear all groups	X	X
.	Remove group	X	X
.	Reorder groups	X	-
.	Legend	X	X
.	Go to scale	X	X
.	Info on layer	X	X
.	Opacity	X	2.3
.	Time layers	X	-
.	Map swiper	2.5	-

Query	Point query	X	X
.	Rectangular query	X	-
.	Polygon query	2.5	-
.	Add/remove query	2.4	-
.	Results window	X	X
.	Results window filter	X	-
.	Results window export	2.6	-
.	Results grid	X	-
.	Results grid zoom to	X	-
.	Results grid export	X	-
.	Infos on point (includes raster)	X	X
Access rights	Login	X	X
.	Lost password	X	X
.	Rights on layers/themes/...	X	X
Print	PDF	X	-
.	PNG	X	-
Drawing/measure	Drawing	X	-
.	Precise lengths	2.5	-
.	Arrows	2.6	-
.	Measure point	X	X
.	Measure line	X	X
.	Measure polygon	X	2.4
.	Measure circle/azimuth	X	-

.	Measure rectangle	X	-
Filters	Spatial filter	X	-
.	Attributes filter	X	-
.	WMS filter	X	-
.	WFS filter	X	-
.	Directed filters	X	-
.	Advanced filters	X	-
.	Saved filters	X	-
Editing	Edit point	X	-
.	Edit line	X	-
.	Edit polygon	X	-
.	Snapping	X	-
Profile	Draw profile	X	-
.	CSV export	X	-
360° images	StreetView	X	-
.	Mapillary	2.6	-
External layers	Online WMS	2.3	-
.	Online WMTS	2.3	-
.	Local KML	2.3	-
.	Local GPX	2.3	-
.	Query point on WMS	2.3	-
.	KML styles	2.5	-
.	Drag/drop external files	2.6	-

Permalink	Share window	X	-
.	Parameter location	X	X
.	Parameter layers/groups/themes/background	X	X
.	Parameter crosshair	X	X
.	Parameter tooltip	X	X
.	Parameter dimensions	X	X
.	Parameter WFS	X	X
.	Parameter drawing	X	2.4
.	Parameter external layers	2.3	-
.	Short URL	X	X
Object editing	Edit one feature	X	-
.	Add new parts	X	-
.	Remove parts	X	-
.	Add fixed shape	X	-
.	Copy from	X	-
.	Cut from	X	-
.	Query other features	X	-
Story maps	Story maps	2.5	-

Appendix 2 - Refactoring Code Base, Concrete Strategy

Libs and Frameworks (Not Definitive)

- Bootstrap (for the CSS)
- Typescript
- Webpack (except for Angular)
- 1 framework (see above)
- Storybooks (for the examples)
- Jest (for the tests)
- OpenLayers
- D3? (for the profile)

See also

Technology acceptances: https://2020.stateofjs.com/en-US/technologies/#arrows_overview

Web components support: https://custom-elements-everywhere.com/

Framework Comparison

Theoretical advantages and disadvantages for each possible framework.

Vue.js

+ Very modern
+ Very trendy
+ Light, fast and uncomplex structure
+ Easy learning curve
+ Used by Swisstopo
+ Very good Web component integration
+ Fast rendering (virtual DOM)
+ Flexible (TypeScript, JSX, ...)
- Carried by one man
- Not the biggest community
- Not very mature

VueJS is a very popular framework with one of the most growing communities. It attracts many developers because of its simplicity and the easy learning curve, while performant in terms of rendering.

It's also in the trend for people who want to run away from the heavy structure like Angular applications, and from the mainstream over used ReactJs. Scales well, even if it does not provide many tools to help you along your project construction (only the component rendering).

Angular

+ Complete framework all in one (CLI, routing, components, dependency injection).
+ Documentation, support
+ Big team at Google
+ Widely used
+ Code organization (lib, apps, components, services)
+ Fast (incremental DOM, ivy renderer)
+ Smooth transition from AngularJS
+ Big ecosystem
+ Mature
+ Scale well
+ No choice to make about typescript, rxjs ...
+ Export to native web components
- Monolithic, strongly opinionated
- Heavy and not easily customizable
- Verbose
- CLI is a bit of a pain, a lot of boilerplates files, too many dependencies and configuration files
- Upgrades painful
- Testing system based on Jasmine/Karma, can be changed to Jest though
- Not trendy
- Steep learning curve

Angular is a very good framework, complete and robust that fits perfectly with big and well structured projects. It requires a very good knowledge to take the real advantage of it. It comes on the long side with a very big (too complex) architecture that is very hard to maintain and to customize, which makes it stay away from the community.

React.JS

+ Not a framework, a library, what makes it light and not intrusive
+ Very efficient to render components (virtual DOM)
+ Simple and intuitive

+ Widely used, particularly in geospatial world
+ Biggest community, very trendy in frontend development
+ Facebook team
+ Very big ecosystem, a lot of react extensions for geospatial (react-geo, deckgl, dataviz, elastic search etc..)
+ Medium learning curve
+ Pure javascript
- Just a rendering engine, you'll have to compose your infrastructure on your own to manage the routing, the services, the state etc.. You'll have to take decisions all along your project life which could sometimes be bad decisions
- No community conventions
- Quite old
- No real support for Web Components

React is the most famous and used javascript framework (library). There is a huge community and a very big ecosystem to help you to build your project with. However, it does not do a lot, so you'll have to depend on third party libraries to manage important things of your application which could make it a bit hard to maintain in the end.

Lit-element (geoblocks)

lit-element is a simple base class for creating fast, lightweight web components that work in any web page with any framework.

In addition, the project is also developing the lit-html library that is an HTML templating library for JavaScript. The library is focused on speed and unlike VDOM libraries, it only ever updates the parts of templates that actually change - it doesn't re-render the entire view.

The goal of the project is to provide a very thin layer on top of the native Custom Element API.

+ Easy to learn because it introduces very few concepts
+ Very fast and efficient
+ Developed and supported by Google
+ Can be used with typescript or JavaScript
+ Existing knowledge (ngm project, Cesium UI components)
+ Chosen by Cesium for https://cesium.com/ion
- Only provide the base class, other libraries have to be used to do the routing, state management, etc ...
- Not suited for the core of medium or large projects
- Smaller community compared to others

Synthesis

Overall, there is not a good choice and a bad choice, the best framework is the one that fits the best with your team in terms of:

- Existing knowledge in the team
- Excitement
- What framework you transition from
- What kind of developer are you (like abstraction or not, like auto generated stuff or not etc..)

In terms of technical aspects, some constraints or characteristics of your project may lead to using one framework rather than one other.

- Project size
- Complexity
- Scope of the application
- Dependencies you think to rely on

We need POCs to validate concrete solutions.

Progressive Steps

a. For each component or loosely coupled piece of code
    - Isolate it in:
        - In GeoBlocks for shareable code:
            - As shareable pure JavaScript code (no UI)
        - In a directory of GMF3 for non-shareable code, clearly separate:
            - The pure JavaScript code (Controller)
            - The UI as a WebComponent
            - If needed the example will be migrated to StoryBook?
    - Integrate the new component in the current AngularJS application
    - Remove the no more used code

    During this step the application still be usable then:

    - We can do a release in the middle of the full step
    - We can do an evolution on an already modified code
    - The code is progressively pushed in production (then after each release the code is production ready)
    - It can take e.-g. 2 years
b. Migrate the rest of the code to the chosen framework

This step should be done in one release, here we replace the core of the application with the new framework.

Notes

About Shadow DOM and Bootstrap Versions

The Shadow DOM is a web component option that isolates the web component CSS and the application CSS, then it implies that the custom CSS can't change the web component, but the CSS variable can.

We are not certain that it is good to use Shadow DOM in the final version.

And the new version of Bootstrap will be released soon.

Then we have 3 choices:

- Start by upgrade bootstrap on the old components (without Shadow DOM)
- Upgrade bootstrap in the last step (without Shadow DOM)
- Use bootstrap 4 in the application, and in the old component, and Bootstrap 5 in the new component (with Shadow DOM), the advantage is that we don't have to update Bootstrap, but the inconvenient is that it's possible that we load more than one time the Bootstrap 5 code, and during the migration we can't change the CSS of the new component (Except variables).

Ideas of Strategies Around Web Components

- monolithic components (GMF2 style, max rely on the framework)

  choose and use the same framework for all components, complex or trivial

- hybrid approach (modern web, rely on ES6 and Web Component standards)

  lightweight Web Component library and/or standard JS for simple components

  the chosen framework for building the apps, the glue

About the Tests

Actually, we have only minimal testing coverage, in the new structure it will be easier to test the view, then it's a good opportunity to have better coverage, but it takes time.

About the UI

It will be good to pass through all the UI to identify and fix the weakness.