✎ Title

Sander Ronde

s.j.p.m.ronde@student.vu.nl

Student number: 2639938

Supervisor: Ivano Malavolta

April 9, 2021

Abstract—

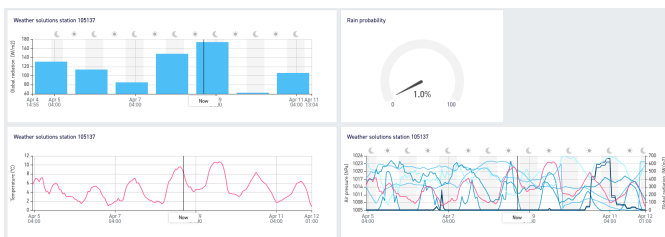***Index Terms—***✎ **keywords**

I. BACKGROUND

This case study was performed at 30MHz, specifically within the context of their software platform. This section describes the company (30MHz), their software platform (the dashboard), and the problem that was solved by the case study.

A. The Company

30MHz is a technology company in the agriculture industry. They offer sensors that collect various types of data, all within the context of agriculture. For example, temperature, humidity, air pressure, etc. They also provide their customers with a dashboard that allows them to view the collected data. 30MHz has several so-called partners. These partners are consultancies in the field of agriculture (among others). They supply 30MHz with additional customers by suggesting their clients become customers of 30MHz as well.

30MHz supplies their customers with the so-called dashboard. An example of it can be seen in Figure 1. It is the central location where any customer can view their data. This dashboard is a web app that, as of this case study, is using Angular 10. Data is fetched from a backend, and the various types of data are displayed in different ways using so-called widgets. An example of a widget is a chart that displays the value of the data over time.

Fig. 1: Widgets on the 30MHz dashboard

1) Apps: The large amounts of data collected by 30MHz can be utilized in many ways. Companies with domain knowledge and expertise in certain areas (such as the partners) can use it to provide customers with new insights and information that simple graphs can not. Because 30MHz itself does not

have this domain knowledge and does not have the resources to create every single possible implementation of this knowledge (in the form of a widget or page in the dashboard), they decided to allow 3rd party developers to develop them instead. In practice, these will be built by the partners and will be provided to customers of 30MHz for a fee through a marketplace. There are currently two implementations:

- *Widgets:* Widgets in the dashboard that take data from one or more sensors and display it. These are made to provide information at a glance and are fairly small, as can be seen in Figure 1. An example would be a new way to display sun data by showing a sun if the plants are able to grow (it's daytime) and a moon if they're not (it's nighttime).
- *Apps:* Full pages in the dashboard web app. These fill the entire screen (bar some 30MHz branding) and can provide more rich and interactive experiences. An example would be a page where clients of a partner can tune some parameters (such as the number of crops, amount of watering) and see a prediction of their revenue. This prediction can be based (in part) on sensor data.

Since these apps will essentially be pages in the 30MHz dashboard and will feel like part of the platform, it is important that they follow the same design style and principles as the rest of the dashboard. This ensures that users are familiar with the apps and that visual consistency across the platform is not broken. This concept has been applied many times. For example on Google's Android through Material Design [1], Apple's design on iOS [2], Zendesk Garden [3] and many more. Importantly, these companies all provide their app developers with a set of components in order to help them maintain their design language. Such a set of components is generally referred to as a UI library. Similarly, 30MHz wants to provide their 3rd party app developers with a UI library as well. There already is an internal set of components that cover the basic set of UI components (buttons, an input, a date picker, etc.), but since these are largely interwoven with other internal code, their source code can not just be provided to 3rd party developers. They have also been written in Angular [4], meaning

---

[1] https://material.io/

[2] https://developer.apple.com/design/

[3] https://garden.zendesk.com/

[4] https://angular.io/

that any developers who wish to develop their app in a different JavaScript (JS) framework are unable to do so. Looking at the most popular web frameworks in the latest Stack Overflow Developer Survey [1] (2020 as of the writing of this paper), we can conclude that the chance that a developer wishes to use a different JS framework is quite large. In order to still provide developers with a UI library, there are two options.

- Write components from scratch in a framework-agnostic format and provide them to developers. Then keep them up to date with the internal set of components by changing one as the other changes.
- Set up automatic conversion from the set of internal components to a framework-agnostic format.

The obvious problem with the first option is that you're maintaining two separate copies of very similar code. This causes a number of issues. Firstly the time spent maintaining a component is doubled. Additionally, feature differences between the Angular framework and the framework-agnostic format we choose are going to lead to problems. Some things that work in Angular might need workarounds in the other format and the other way around. Another issue with this option is that the components have to be written entirely from scratch. While this would be manageable for simple components such as buttons, this is unfeasible for more complex components. One of which is the chart component. This component is important to have in the 30MHz design library, seeing as it is able to display the sensor data. However, the source code for the chart alone is already 500 lines long, most of which are tightly integrated with the rest of the platform. Through all of its references, the chart component references about half of the source files in the dashboard. Rewriting all of this in another framework is completely unfeasible and not worth the effort, leading us to explore the second option.

While the second option is not an easy one and it will likely be a very complex process to set up, it is one that will scale a lot better. Once it is set up, any new components will be automatically converted, and any changes will be propagated automatically. In the long run, this should save a lot of time. This is the option 30MHz eventually decided on. However, a framework-agnostic format needs to be chosen to facilitate this process.

B. Web Components

When it comes to choosing a framework-agnostic format for a UI library, there are very few options. Looking at the literature, we find Quid, a DSL for generating components in various frameworks [2]. It currently supports Web Components [5], Stencil [6], Angular and Polymer [7]. While this is fairly impressive, the authors do mention it should only be used for rapid prototyping. Since it only supports a fairly small set of supported frameworks and it has the problem of requiring a DSL which the Angular code would have to be converted to, this is not a great fit for us.

[5]https://developer.mozilla.org/en-US/docs/Web/Web_Components
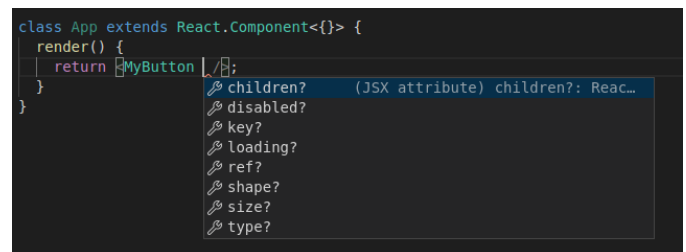[6]https://stenciljs.com/
[7]https://www.polymer-project.org/

This brings us to the other option, namely Web Components. Web Components (also known as Custom Elements) are a technology proposed in 2013 [3] and implemented in major browsers in 2018 [4]. It allows for the creation of custom HTML elements using JavaScript. These elements can then be used like regular HTML elements. Since every JS framework has support for native HTML elements and almost every framework has full support for Web Components [5], we can cover most JS frameworks by using Web Components as our target format.

C. Angular Elements

To perform the conversion of Angular components to Web Components, we will be using Angular Elements [8]. Angular Elements is a JS package published by the developers of Angular that allows the conversion of Angular components to Web Components. Angular apps are normally mounted to the DOM by the user through a call to the $bootstrapModule$ function. After this call, the bootstrap component is mounted to the DOM, after which child components are rendered within its root recursively. Angular Elements works slightly differently. Components registered as Web Components through Angular Elements are instead rendered whenever an HTML element with the registered tag is added to the DOM. When this happens, a new root is created in place of this new HTML element. Instead of there being a single root in which everything is rendered (as is the case in a normal Angular app), components are all rendered in their own local root. We will be using Angular Elements for the conversion to Web Components in this case study.
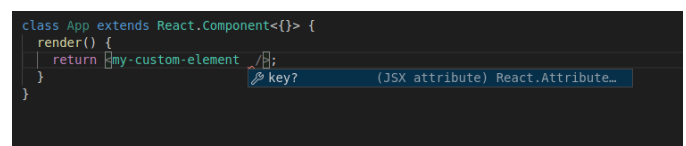
Fig. 2: An example of a component that provides type hints

Fig. 3: An example of a component without type hints

D. Javascript frameworks

While converting components to Web Components makes them usable in most JS frameworks, they do not provide

[8]https://angular.io/guide/elements

a perfect experience. The first problem is them not being perfectly usable in every JS framework. As of the writing of this paper, there are still some issues preventing them from working completely in React JS [9]. These issues mostly concern the passing of non-primitive data to the components, such as JavaScript Objects, Arrays and Functions. The second problem is that they are not native to JS frameworks, and as such, do not integrate very well with the tooling provided by the framework. An example of type hinting provided by the framework and editor can be seen in Figure 2. Compared to Figure 3, which shows a component with no type hinting, Figure 2 provides the developer with much more information and shows them what options are available to them. Instead of having to search the web for the available properties, they are provided by the element's source code and displayed by the framework tooling and editor. In order to provide developers with proper tooling and type hinting for components, we will provide what we'll call a wrapper for each framework that either provides this tooling or does not support Web Components fully. This wrapper is written in the language the framework which it is targeting provides, and as such, allows the framework to infer information from its source code. Under the hood, this wrapper still makes use of the Web Components to render the components in the UI library, but this wrapper serves as glue code between the framework and the Web Components. By combining the two steps of converting the original Angular components to Web Components, and the Web Components to wrappers for frameworks, we are able to provide developers with an experience that is native to their framework, regardless of the fact that the original source code was written in Angular.

II. Related Work

There are various fields in which the related work is important to us in this paper. Namely related work in the area of UI Libraries, Angular Elements (and the accompanying process of converting to Web Components), related work on Web Components themselves, and related work on the creation of wrappers around Web Components to target JS frameworks.

A. UI Libraries

We found a few studies that cover the area of UI libraries. In [6] [7] [8], the authors all build UI libraries. They focus mostly on the technologies used, how they work, and how they contribute to the building of the UI library. Looking at blog posts, we find numerous posts on Web Components. In [9] [10] [11] [12] [13], the authors provide guidance in setting up and creating a UI library. These blog posts mainly concern the basics, explaining how to get started with the process.

B. Angular Elements

Research on the area of Angular Elements is very sparse. In our search, we only found a single paper on this subject. In [14], Angular Elements is used to convert form components

to Web Components so that they can be rendered dynamically. On the other hand, blog posts on this area are numerous. In [15] [16] [17] [18] [19] [20] [21] [22] [23] [24] [25] [26], the authors explain how to set up Angular Elements and how to use it to create a new component library. These blog posts mostly focus on creating new components or converting simple components through Angular Elements, not so much on converting larger and more complex components. They all use new and empty projects, contrary to [27] and [28]. The authors use Angular Elements to convert existing AngularJS (an older version of Angular) components to the newer Angular. They do this by converting the source code of existing AngularJS components to Angular source code. By itself, this would break since the application root still runs on AngularJS and is unable to handle Angular code. By using Angular Elements to convert the Angular code into Web Components, the Web Components are able to run inside the AngularJS root. This is thanks to the low-level nature of Web Components, allowing any framework that can render HTML elements to use them. Through this iterative process, they are able to convert components one by one, converting the root component once all of its children have been converted as well.

Unfortunately, we were unable to find any related work on the conversion of complex Angular components to Web Components through Angular Elements. Related work seems to focus mostly on small get-started style projects. In the cases where they do focus on more complex projects, it seems like the only use is the conversion from AngularJS to Angular.

C. Web Components

✎ Web Components section

D. JS Framework Wrappers

We were unable to find any research on JS framework wrappers. This does not seem to be a problem that has been tackled very often, at least in literature. In [5], the authors keep track of the current usability of Web Components in various JS frameworks. Notably, the ReactJS framework does not fully support Web Components at the time of writing for this paper. In ReactJS, non-primitive values (such as Objects, Arrays, and Functions) can not be passed to Web Components, along with some other issues. As such, it is the only framework that needs a wrapper for the UI library to function at all. Looking at how to fix this issue, we find that [29] proposes some effective solutions. In this blog post, the author explores various options to tackle this problem of passing non-primitive data.

References

[1] StackOverflow, "Stack overflow 2020 developer survey," https://insights.stackoverflow.com/survey/2020, 2020.

[2] P. J. Molina, "Quid: prototyping web components on the web," in *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, 2019, pp. 1–5.

[3] W3C, May 2013. [Online]. Available: https://www.w3.org/TR/2013/WD-custom-elements-20130514/#about

[4] 2020. [Online]. Available: https://caniuse.com/?search=webcomponents

[5] 2020. [Online]. Available: https://custom-elements-everywhere.com/

[6] T. Ky Nam, "Ui library project setup for vaimo group with modern web technology," 2019.

[7] L. Annala, "Documentation of a ui-library used in web development," 2017.

[8] M. Mráz, "Component-based ui web development," 2019.

[9] W. Dobry, "How to build an effective design framework (includes a free sketch ui framework)," Nov 2017. [Online]. Available: https://www.toptal.com/designers/ui/design-framework

[10] E. Ella, "Building a ui component library for your startup," Nov 2019. [Online]. Available: https://dev.to/giteden/building-a-ui-component-library-for-your-startup-4cek

[11] J. Pscheid, "Component library: How to build a library for a ux ui design system," Nov 2020. [Online]. Available: https://www.emergeinteractive.com/insights/detail/how-to-ux-ui-design-system-component-library/

[12] J. Saring, "Building an awesome ui component library in 2020," Nov 2020. [Online]. Available: https://codeburst.io/building-an-awesome-ui-component-library-in-2020-a85cb8bec20

[13] ——, "Build a react component library," Nov 2020. [Online]. Available: https://itnext.io/building-a-scalable-ui-component-library-4607de91955a

[14] J. Armengol Barahona, "Development of an angular library for dynamic loading of web components," B.S. thesis, Universitat Politècnica de Catalunya, 2020.

[15] N. Basal, "Understanding the magic behind angular elements," Jun 2019. [Online]. Available: https://netbasal.com/understanding-the-magic-behind-angular-elements-8e6804f32e9f

[16] C. Kitson, "Angular element (web components) — wrapping an angular app in a custom element," Feb 2019. [Online]. Available: https://medium.com/kitson.mac/wrapping-an-angular-app-in-a-custom-element-web-component-angular-element-in-4-simple-steps

[17] R. Mackey-Paulsen, "Angular elements: use them everywhere including your angular app," Jun 2020. [Online]. Available: https://medium.com/@smarth55/angular-elements-use-them-everywhere-including-your-angular-app-697f8e51e08a

[18] P. Nalepa, "How to convert angular component into reusable web component?" Feb 2020. [Online]. Available: https://blog.piotrnalepa.pl/2020/02/02/how-to-convert-angular-component-into-reusable-web-component/

[19] Notiz.dev, "Angular elements: Create a component library for angular and the web," Apr 2020. [Online]. Available: https://medium.com/swlh/angular-elements-create-a-component-library-for-angular-and-the-web-8f7986a82999

[20] N. Sonara, "Angular elements," Sep 2020. [Online]. Available: https://www.thirdrocktechkno.com/blog/angular-elements/

[21] J. Strumpflohner, "Intro to angular elements," Apr 2019. [Online]. Available: https://juristr.com/blog/2019/04/intro-to-angular-elements/

[22] S. L. C. Nostra, "Build a reusable angular library and web component," Dec 2020. [Online]. Available: https://studiolacosanostra.github.io/2019/07/19/Build-a-reusable-Angular-library-and-web-component/

[23] R. S, "Using angular elements - why and how? - part 1," Apr 2019. [Online]. Available: https://blog.bitsrc.io/using-angular-elements-why-and-how-part-1-35f7fd4f0457

[24] T. Team, "Angular 10/9 elements tutorial by example: Building web components," Aug 2020. [Online]. Available: https://www.techiediaries.com/angular/angular-9-elements-web-components/

[25] A. Vardanyan, "Angular + web components: a complete guide - angular indepth," Feb 2020. [Online]. Available: https://indepth.dev/posts/1116/angular-web-components-a-complete-guide

[26] C. Williams, Apr 2020. [Online]. Available: https://indepth.dev/posts/1228/web-components-with-angular-elements

[27] T. Helgevold, Mar 2019. [Online]. Available: https://blog.nrwl.io/upgrading-angularjs-to-angular-using-elements-f2960a98bc0e

[28] T. in 4 simple steps, Aug 2019. [Online]. Available: https://medium.com/capital-one-tech/capital-one-is-using-angular-elements-to-upgrade-from-angularjs-to-angular-42f38ef7f

[29] A. Remdt, Jun 2019. [Online]. Available: https://itnext.io/handling-data-with-web-components-9e7e4a452e6e