Building a Better Thunderbird UI with StencilJS

Table of Contents

Introduction

In modern web development, front-end frameworks enable developers to rapidly build dynamic web applications within a defined structure of HTML, CSS and JavaScript. Popular frameworks such as Angular, React and Vue have dominated the world of front-end development in recent years. These tools have become easier to adopt over time as new features have been released and their user bases have grown.

However, the pace of modern front-end development moves fast. Very fast. While these new technologies might be popular today, what will the next shiny framework be? Judging by the release of new major frameworks, seemingly every year, we are sure to see the next Angular or React in the not-so-distant-future.

In a corporate intranet, such as Sandia's where there is no standardization around one framework for app development, how do we build consistency? Since every framework has a differing opinion about how code should be written, it can be a problem when applications are each using what they think is best. While standardizing on one framework is a great topic, it's not something I want to delve into here. Rather, I want to advocate for a framework-agnostic approach to our design system, Thunderbird UI (TUI). My proposal outlined here would build greater code consistency across applications and break down the front-end framework barrier to adoption of TUI. In turn, this would build a more consistent and beneficial user experience across web applications at Sandia.

Thunderbird UI: Sandia's Design System

A design system is a collection of reusable components, visual styles, user experience requirements and best practices for building user interfaces within an organization. Design systems build a consistent user experience within an organization by enabling products to share a unified "look and feel" and user

experience. Design systems also improve designer and develop efficiency by providing components and design styles that can be re-used and not re-invented with every project.

At Sandia, we designed, developed and currently maintain TUI as our design system. It includes reusable components contain code for developers to implement in their existing websites or applications. It also contains user experience requirements, accessibility standards and resources for designers and developers across the Labs.

Challenge: Developing a Design System for Sandia's Diverse Ecosystem

When we began developing TUI in the latter part of 2017, Bootstrap was a popular front-end library being used at Sandia. With this in mind, we decided to base TUI's code base off this library to accommodate most projects at Sandia.

By the release of TUI in 2018, other front-end frameworks such as Angular and React were growing in prominence at the labs. Developers began implementing these frameworks in more projects and usage of Bootstrap decreased.

The current landscape of technologies being used at Sandia is vast. There is no standardization on a front-end framework for developers to use and therefore a variety of frameworks exist across projects. The framework chosen typically depends on the preference of the project's development team. Due to this heterogeneity in frameworks there is no common language that all front-end developers speak at Sandia.

This situation makes it incredibly difficult to design, develop and maintain a design system where developers are pulling code from. If developers are using various front-end frameworks, it's a daunting task to support a code base that caters to every possible front-end framework. Future-proofing TUI in this current environment is impossible unless we change the way we are building and distributing TUI components.

Solution: Web Components

The challenge of building TUI for a growing number of front-end frameworks led our team to explore various options to genericize TUI. Developers from our group and others brainstormed solutions to accomplish this.

To provide context, the primary issue that developers have with implementing the current version of TUI is its reliance on Bootstrap which implements the jQuery JavaScript library. The dependence on both Bootstrap and jQuery have dissuaded developers who are not using these libraries, in exchange for more modern frameworks such as Angular and React. There can be technical conflicts to using Bootstrap and jQuery on top of another modern front-end framework as well as code bloat and technical debt. Modern web development, simply, does not rely on these technologies as it once did.

Taking these concerns into consideration, the TUI team brainstormed options to make TUI more available for use in various front-end frameworks and to future proof TUI. In mid-2019, development teams from Design for User Experience (DUX) and Developer Experience (DevX) met to exchange ideas and vocalize concerns about the current state of TUI. There were various solutions proposed during this meeting, some of which had been discussed prior. These solutions included providing a CSS-only version of TUI,

leveraging existing front-end libraries such as Angular Material, using COTS products such as Telerik's Kendo UI, and converting TUI components to Web Components.

Providing a CSS-only version of TUI would strip out all the component JavaScript, thus reducing conflicts with other front-end frameworks. While appealing in its simplicity, this solution would force many essential TUI components to be removed due to their reliance on JavaScript. The second option, leveraging existing front-end libraries such as Material UI, would enable us to deliver TUI to targeted frameworks using compatible UI libraries for each framework. With this option we would be in a similar predicament that are in today. Supporting multiple UI libraries would be a maintenance nightmare and would not be sustainable long-term. We researched Telerik's KendoUI, which would provide us with a ready-made library for supporting major frameworks. However, we didn't want to rely on a third-party vendor for TUI. If new frameworks appeared that were not supported by Kendo, we also wouldn't be able to support them in the distribution of TUI. KendoUI also provides many components that we have deemed unnecessary for our design system.

The last solution discussed was to use Web Components for TUI. Web Components would enable us to build a framework-agnostic library of components using standards-based JavaScript. We wouldn't need to sacrifice essential TUI components due to potential JavaScript conflicts and Web Components would provide support for any number of frameworks. Plus, this would give us control over our own in-house component library when updates need to be made. Out of this meeting it was decided to explore using Web Components as the basis for TUI components.

What are Web Components?

Web Components enable developers to build reusable and encapsulated HTML elements that contain custom templates, behaviors and tags (e.g. <tui-button>). Web Components are lightweight and based on modern web standards. They work within any modern browser and can be implemented within any modern JavaScript framework. With Web Components, the TUI team would not be rewriting the same components for every major framework but rather developing components that can be written once and used across multiple frameworks. This is the primary reason that Web Components are being used in many corporate design systems today.

With these key advantages in mind, Web Components are the ideal technology to build TUI components in order to accommodate Sandia's diverse landscape of front-end frameworks. Using Web Components would not only help solve the challenge today in genericizing TUI to make it more widely used, it would also future-proof TUI as new front-end frameworks become implemented at Sandia.

Creating Web Components with StencilJS

Several libraries exist to simplify the process of building Web Components. Major libraries include StencilJS, Angular Elements, Svelte and Lit-Element. I have identified Stencil as the best option for implementing Web Components in TUI for various reasons, including simplicity, performance and support.

Stencil is a mature compiler that is used at the enterprise-level by major companies such as Apple, Amazon and Microsoft. It has the backing of the Ionic Framework team (https://ionicframework.com/), who develop open-sourced products for web app development. Stencil is open sourced which is in

keeping with the spirit of TUI and opposed to Angular Elements, it doesn't rely on existing knowledge of a specific front-end framework. The build size of Stencil is smaller than Lit Element and Angular Elements, resulting in better performance. Stencil and Svelte are the closest competitors. Both have a very small build size as they are the best performing tools out of the compilers researched. However, Stencil includes support for TypeScript and the backing of a major company, two factors that set it apart from Svelte during my research.

Due to its simplicity, Stencil is easy to setup and get started with. Stencil is also very lightweight and performant. Its lightweight nature enables Web Components to be built faster and delivered with better performance. When Stencil code is compiled it is compiled without the Stencil library itself, making it even more lightweight and ideal for frameworks such as Angular, React and Vue. Beyond size and simplicity, Stencil can generate component code for any modern front-end framework. It's easily reusable and can be used within any framework or no framework at all.

Stencil is well documented and is currently maintained and supported by the Ionic team. Stencil also has thriving user community as its popularity has grown in recent years. The open-sourced nature of Stencil enables contributions to be made to the Stencil library now and into the future.

As with any modern front-end technology there is always the risk that Stencil could lose popularity and lack support in the future. However, I am convinced that due to the current support behind Stencil, its active community base and the fact that it leverages and a standards-based approach to web development, it is an ideal for producing Web Components for TUI.

Anatomy of a Stencil Component

With Stencil, developers can efficiently build Web Components by using modern technologies including JSX, TypeScript and the Virtual DOM. Stencil leverages common patterns found in modular-based web development for building a component-based architecture. Each component is completely independent; however, they can share unifying styling attributes. Their independence allows for components to be easily inserted into a view. Independent components can also be pieced together to create larger components. This atomic design philosophy[1], falls in line with the concept of a design system as a collection of reusable building blocks.

```
import { Component, h, Prop } from '@stencil/core';

@Component({
  tag: 'tui-button',
  styleUrl: 'button.scss',
  shadow: false,
})
export class Button {

  @Prop() context: string
  @Prop() primary: boolean;
  @Prop() message: string;

  render() {
    return (
      <button class={"btn btn-" + (this.primary ? "" : "-outline") + this.context}>{this.message}</button>
    );
  }

}
```

Let's look at an example as found in the TUI button component. In the code snippet, you can see that it contains a few key elements of any component:

- A custom tag: This defines the name of the tag as it is inserted into the view (e.g. <tui-button>).
- A style URL: The path for the button's unique styling.
- Prop(): These are properties that a button can have. With the button we have properties that define the context (e.g. danger, success, etc.), primary (whether the button is primary or not) and message (button text).

By setting the Prop() values, we can distribute this button as a TUI component and let the developer apply the correct values. The button is a simple example and we only need to configure properties. However, many components will have more interactions, therefore requiring more code logic. By leveraging methods, states and events found in Stencil's API, we can build these more advanced components for TUI.

Eventually, once all components are completed and ready to be shipped, we can run a command in Stencil to optimize the components for distribution.

Next Steps

Implementing Web Components with Stencil would modernize TUI and bring about several enhancements to the design system. By building a framework-agnostic set of components, we would be making TUI more appealing to developers at Sandia who are using different frameworks. A greater distribution of TUI across applications at Sandia would improve the overall user experience of applications and generate a more consistent "look and feel." Web Components would also enable us to future proof TUI for when the next major framework becomes implemented at the Labs.

From a maintenance perspective, Web Components via Stencil would also greatly increase efficiency. Instead of supporting a code base for every major framework, we would only focus on one codebase.

I am excited by the potential that Web Components and Stencil hold for building TUI components. I believe that this approach would the primary challenge plaguing TUI and position our design system well for the future. If this proposal to convert TUI components to Web Components is approved, I would like to move forward in earnest. In my opinion, TUI has a great deal of untapped potential that I Web Components and Stencil could unlock.

References

[1] Atomic Design (https://bradfrost.com/blog/post/atomic-web-design/), by Brad Frost