On using the Elm Programming Language

The Tufts Search Experience

By Harrison Kaiser, ECE '19

Introduction
For our Senior Design project we had to design a web page which would search publicly available Tufts information. The goal of the project was to gather the most relevant information and provide an intuitive, unified interface. We worked closely with Tufts Technology Services (TTS) to create a prototype of this service which they could later refine.

Project
From an engineering perspective, the project broadly broke down into two parts: the web page itself and the server which it queries. This tech-note will cover the technology used for the web page, why we chose it, and what it enabled us to do.

Choosing Elm
To create the web page, we chose a relatively new language called Elm. Unlike other alternatives, React (which Facebook backs[1]) and Angular, Elm provides a guarantee of not crashing[2]. This is a rare feature among web frameworks. Whereas a typical JavaScript application would crash several times in production, an Elm application offers a seamless user experience.

Elm separates the data itself (e.g. the search results) from how it looks on the web page. That means that we could easily build many different ways of viewing the same data. So, our human factors engineer could design without having to worry about how the data was structured, while our computer scientists could build different looks for the same data. This separation would allow our system to be flexible and quickly adaptable.

Because Elm doesn't carry the same kind of legacy that a widely used language like JavaScript does, it is more predictable. Whereas JavaScript cannot remove any error-prone features because that could break the web, Elm focuses on being error-free, quick to run[3], and small to download[4].

Elm also allows us to accommodate a host of user interactions. When a user interacts with the web page, Elm processes this as an *event*. Then, we can receive the event and automatically update the view, knowing that nothing the user clicks can crash the page. This forms an *event loop*, pictured in Figure 1. Elm only requires that we specify: 1) Possible events; 2) How to update information based on events; and 3) How to display that information. Because of Elm's guarantees, we can confidently change one of these without breaking the others.

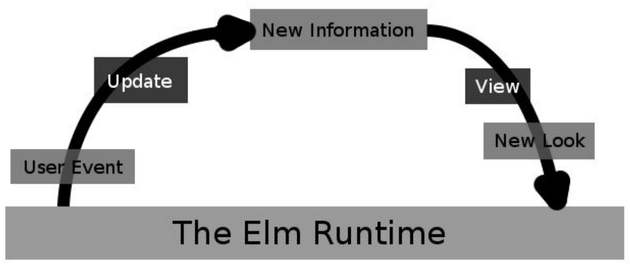

Figure 1. The Elm Runtime Model

How it worked in Practice

We quickly built our first prototype and easily adapted to changes in the specification from TTS. These changes did not significantly add to development time because of Elm's error-free guarantee. Because Elm separated considerations about the data from the look and feel of the web page, we could neatly split work between members.

Elm's Runtime Model allowed us to implement an autocomplete drop down with minimal difficulty. This feature loads the top five results that match the user's incomplete query in real time. In JavaScript, this feature would require extensive manual case-handling that could expose the application to unforeseen corner cases and cause unexpected behavior. However, Elm enforces that we handle all possible cases, which means that we could ensure that there is no undefined behavior. Although we had anticipated this feature to be the most complex, eschewing JavaScript in favor of Elm made it the most simple.

Conclusion

Elm enables agile development and confident reconfiguration as new priorities emerge, which industry professionals have valued[5]. It let us clearly distinguish between the information and the display, allowing for efficient teamwork. Its error-free guarantee made it simple to reorganize, and its structure made it easy to implement seemingly complicated features. Given the experience of our team and the parameters of the project, Elm was clearly the right choice.

References

1. "React – A JavaScript Library for Building User Interfaces." – *A JavaScript Library for Building User Interfaces*, Faceook, reactjs.org/.

2. Feldman, Richard, and Wesley Reisz. "Richard Feldman Discusses Elm and How It Compares to React.js for Front-End Programming." *InfoQ*, InfoQ, 28 Apr. 2017, www.infoq.com/podcasts/richard-feldman.

3. Czaplicki, Evan. "Blazing Fast HTML; Round Two." *Blog/Blazing-Fast-Html-Round-Two*, Evan Czaplicki, 30 Aug. 2019, elm-lang.org/blog/blazing-fast-html-round-two.

4. Czaplicki, Evan. "Small Assets without the Headache; Minification Made Easy with Elm 0.19." *Blog/Small-Assets-without-the-Headache*, Evan Czaplicki, 21 Aug. 2018, elm-lang.org/blog/small-assets-without-the-headache.

5. Olah, Rudolf. "Elm vs. React: Development and Performance." *Elm vs. React: Development and Performance - Codementor*, Codementor, 13 Apr. 2017, www.codementor.io/rudolfolah/elm-vs-react-development-performance-compare-603dyh83m.