Background Report: TypeScript Webapp – Linkbait Article Generator

CPSC 311 Team T3

November 27, 2015

Name: Ashley Lee
Ugrad ID: k7y8
Student ID: 34959122
Email: alee238@hotmail.com

Name: Min Seok Ray Roh
Ugrad ID: l9x9a
Student ID: 33737123
Email: msroh94@gmail.com

Name: Michelle Wilson
Ugrad ID: a2d0b
Student ID: 60855087
Email: m888wilson@gmail.com

Name: Cecile Leung
Ugrad ID: o8r6
Student ID: 90600974
Email: cecilephl@gmail.com

Name: Norman Sue
Ugrad ID: h0e9
Student ID: 20396131
Email normansue3@gmail.com

1 Project Overview

For the 100%-level milestone, we would complete the following:

1. Write and publish an `npm` module in TypeScript which generates a JSON representation of linkbait articles by taking advantage of type checking.
2. Write missing TypeScript type definition files for third-party `npm` modules that our `npm` module depends on.
3. Verify that all of our written TypeScript type definition files are bugfree with tscheck. [1]
4. Use our `npm` module to create a simple Node.js+Express API server that runs the web scraper module and sends the resulting JSON articles to the browser client.
5. Write a TypeScript single-page application browser client that renders the calculated backend JSON API data with jQuery and Backbone.js.

An explanation of how what each of these components are and how they showcase specific TypeScript language features is given in the following subsections.

1.1 Write and publish an `npm` module in TypeScript which generates a JSON representation of linkbait articles by taking advantage of type checking.

We've chosen to publish our project as an `npm` module because, according to modulecounts.com [2], it's the industry-standard JavaScript (and thus TypeScript) package management format for running code on Node.js, the JavaScript runtime environment that allows our compiled TypeScript code to be executed server-side. By using a widely-accepted, industry-standard library format, we can ensure the core functionality of our project can be reused by other programmers.

Specific TypeScript language features to be used by the module include:

1.1.1 Functions

As TypeScript is a superscript of JavaScript, there are similarities between them, however, when writing the former, there are differences that will compile into the latter. These differences may give flexibility in writing functions. Undoubtedly, when coding our linkbait article generator, we're going to end up using functions to represent the behavior of the generator, while probably encapsulating those in classes. TypeScript has added functionality to functions that JavaScript does not, such as binding the context of 'this', contextual typing, and the ability for optional and default parameters. [3] These capabilities make it easier to code and maintain a typed program, which will help us maintain our code better than if we coded in JavaScript.

1.1.2 Generics

In TypeScript, if a generic function is created, the compiler will enforce that all the actions taken in the function are used in such a way that they *could* work with all types. So if you attempt an operation that is only allowed for type `String` but not `Object`, it will be forbidden (causes an error). [4] But this is recognized only at compile time. There is no run-time representation for type parameters. [5]

Despite this, static-checking of the appropriate usage of generic types will be useful for when we iterate over possible articles in a `for... each` loop, to perform bulk formatting such as inserting abbreviations for our headlines.

1.1.3 Mixins

Mixins are a way of reusing code to make new classes by combining desired parts of existing classes without taking on all features of those classes. The specificity of which methods are retained avoids the problems from ambiguity that comes up in multiple inheritance otherwise. They are useful in situations where a particular behaviour is repeated in many classes, providing optional behaviour in a class, and making variations on similar features in the augmented class. [6]

1.1.4 Intersection Types

By using the `pjscrape` web scraping library, we will collect an `Array` of parsed data from other popular articles, which will ensure that our generated articles' headlines will have a high chance of being clicked on.

However, the types of the objects within the collected `Array` data will be unknown ahead of time, aside from the fact that they could be a mathematical union of a discrete number of types. TypeScript's intersection types feature [7] allows us to statically check that our subsequent functions use the parsed `Array` elements in a way that's consistent with JavaScript's built-in methods on native library types. For example, we can only call `String.prototype.concat` on `String` types, but the elements may be both `String` and `Number` types, since some of the articles we parse may contain numeric data.

If we were to write our subsequent data cleaning code without the usage of TypeScript's static type checking features, it may result in JavaScript `TypeErrors`.

1.2 Write missing TypeScript type definition files for third-party `npm` modules that our `npm` module depends on.

We plan on scraping the text content of existing websites in order to generate our own linkbait articles. This will require using the existing pjscrape [8] JavaScript web scraping library. `pjscrape` currently does not have TypeScript type definitions within the DefinitelyTyped [9] repository, so we'll need to inspect their `exported` functions and write `module` and `interface` definitions for each of them.

1.3 Verify that all of our written TypeScript type definition files are bugfree with tscheck.

`tscheck` is an existing JavaScript library based on research by Feldthaus and Møller that can be used to find bugs in handwritten TypeScript type definition files. Running this check ensures that the `module` and `interface` definitions that we need to write for the required `npm` modules (e.g. `pjscrape`) are bugfree, such that when our code calls their functions, TypeScript's `tsc` compiler will correctly perform static type checking. [1], [10]

1.4 Use our `npm` module to create a simple Node.js + Express API server that runs the web scraper module and sends the resulting JSON articles to the browser client.

Express.js is the most commonly used Node.js library for creating minimalistic API servers. We will use it to serve the HTML, CSS and compiled JavaScript browser client.

1.5 Write a TypeScript single-page application browser client that renders the calculated backend JSON API data with jQuery and Backbone.js.

Our browser client will use Backbone.js, one of the popular client-side JavaScript frameworks for providing structure to the registration of jQuery callback functions to the various DOM elements that we'll use to render our API's JSON data containing the generated linkbait articles.

2 Project Value

Of the most popular compile-to-JS languages, TypeScript greatly lags behind in adoption and has significantly fewer exemplary repositories to draw inspiration from. There are currently 8,267 TypeScript repositories on GitHub, compared with the 1.7 million JavaScript and 54,605 CoffeeScript (the leading compile-to-JS language that competes with TypeScript) repositories.

Creating an additional TypeScript project provides future potential TypeScript programmers with an additional example of how they could utilize TypeScript's static typing features.

3 Project Importance

3.1 Background

Clickbait is a term for headlines that catch people's attention and curiosity enough for them to follow a link. The main goal is for the user to click through to the target website. Secondary is the content, which has a reputation for being of very low quality. They exist due to a business model whereby the more visitors there

are to a website, the more advertisers are willing to pay to use that site. Ads on that site gain more visibility with each visitor and can have further referral links. More visitors can also affect a site's rank when it shows up on search results due to search engine optimization (SEO) rules. Therefore, any click-through on a link can have compounding effects on visibility and subsequently revenue from advertising.

3.2 Negatives and possible positive side-effects

Clickbait (also known as linkbait) headlines may read like news, but the articles are often hastily created with little research, no insight, misleading information, or they can be outright advertisements masquerading as impartial articles. It can annoy consumers [11] and it can also affect the way content is created, encouraging poor journalism [12]. So, there is little *inherent* value to creating clickbait. However, we may gain some beneficial knowledge as a byproduct. Creating successful clickbait involves the psychology of curiosity and persuasion. If we can learn more about what works, we can put it to good use (without annoying the readership) by providing high quality content at the target site. For example, actual information.

3.3 Our perspective

We chose this as our application because it would be an amusing way to showcase the features of TypeScript. However, it could be extended to track data on which generated headlines are successful (number of click throughs and duration spent on target sites). This information could be used to learn how to best reach certain audiences with information. It could be anything from public health outreach to how undergrads can apply for co-op. Another possible benefit is that our team could sign up for ad revenue and make money for this project.

4 Project Impact

Standardized support for writing TypeScript `npm` modules that compile to JavaScript by using tsconfig.json files to integrate with existing JavaScript and TypeScript `npm` modules has only been added 4 months ago in TypeScript 1.5. [13]

By publishing a library using this relatively-new build process, we are contributing to the TypeScript community by providing an additional working example of how to use this new build feature of the language, since there are only 678 results [14] when searching for GitHub code that use tsconfig.json files.

Citations

[1] J. Doe, *First book*. Cambridge: Cambridge University Press, 2005.

[2] J. Doe, "Article," *Journal of Generic Studies*, vol. 6, pp. 33–34, 2006.

[3] J. Doe, "Article," *Journal of Generic Studies*, vol. 6, pp. 33–34, 2006.

[4] J. Doe and J. Roe, "Why water is wet," in *Third book*, S. Smith, Ed. Oxford: Oxford University Press, 2007.

[5] J. Doe and J. Roe, "Why water is wet," in *Third book*, S. Smith, Ed. Oxford: Oxford University Press, 2007.

[6] J. Doe and J. Roe, "Why water is wet," in *Third book*, S. Smith, Ed. Oxford: Oxford University Press, 2007.

[7] J. Doe and J. Roe, "Why water is wet," in *Third book*, S. Smith, Ed. Oxford: Oxford University Press, 2007.

[8] J. Doe and J. Roe, "Why water is wet," in *Third book*, S. Smith, Ed. Oxford: Oxford University Press, 2007.

[9] J. Doe and J. Roe, "Why water is wet," in *Third book*, S. Smith, Ed. Oxford: Oxford University Press, 2007.

[10] J. Doe and J. Roe, "Why water is wet," in *Third book*, S. Smith, Ed. Oxford: Oxford University Press, 2007.

[11] J. Doe and J. Roe, "Why water is wet," in *Third book*, S. Smith, Ed. Oxford: Oxford University Press, 2007.

[12] J. Doe and J. Roe, "Why water is wet," in *Third book*, S. Smith, Ed. Oxford: Oxford University Press, 2007.

[13] J. Doe and J. Roe, "Why water is wet," in *Third book*, S. Smith, Ed. Oxford: Oxford University Press, 2007.

[14] J. Doe and J. Roe, "Why water is wet," in *Third book*, S. Smith, Ed. Oxford: Oxford University Press, 2007.