CS Capstone Technology Review

November 2, 2018

Creating Immersive Experiences on the Web using VR and AR

Prepared for

Intel

Alexis Menard

Prepared by

Group 47

Brooks Mikkelsen

Abstract

This document is a Technology Review paper that focuses on the project setup for Capstone group 47, Creating Immersive Experiences on the Web using VR and AR. This document reviews potential languages (and subsets of languages) used for development, including plain JavaScript, TypeScript, and Flow. It also reviews potential Asset Bundlers, including Webpack, Parcel, and Rollup. In addition to this, it reviews three options for continuous integration - Travis CI, Circle CI, and Jenkins.

CONTENTS

1 INTRODUTION

The goal of our project is to provide an example for future developers interested in interfacing with the WebXR API. This API gives developers access to end users' Virtual Reality and Augmented Reality systems, so that they can make interactive VR and AR experiences for the web.

One of my focuses in the project will be project setup. This includes choosing the best language suited to our style of development, the build tools (in our case an asset bundler and development environment), and the continuous integration tool we will use to build, test, and deploy our application.

2 LANGUAGE

2.1 Plain ES2018 JavaScript

One of the requirements for our project is to have the Virtual Reality experience hosted within modern web browsers. Since all modern web browsers use JavaScript, and the WebXR API is implemented in JavaScript, we will need to use JavaScript or some derivation of it to create our project.

Plain JavaScript does not have any compile-time type checking. The only type checking it does is if there is an error. As with other dynamically typed language, plain JavaScript allows developers to rapidly create projects and get products out the door quickly. However, although it is easy to quickly create projects in plain JavaScript, it is much more challenging to write code that is maintainable over the long run. This is because without static types, much more documentation must be written for more than one person to write code in the same code base, since it all must interact with itself.

Every year, the ECMA standards organization makes updates to the JavaScript standard that is implemented in each browser. However, it takes time for browsers to implement these standards, and for users to actually update their browser. Therefore, if developers want to use the latest standard, they will need to use a preprocessor such as Babel to create polyfills for them so that user on older browsers can still use all the functionalities of the website [1].

2.2 TypeScript

TypeScript was created by Microsoft as a way to provide static typing for JavaScript. It does this by forcing users to add type definitions, either explicitly or implicitly (if possible in the situation) to all of their variables, functions, classes, and objects. It also adds interfaces, which are used to tell the compiler (and the developer) about the shape of JavaScript objects and classes, including whether properties are optional or mandatory, and what the type of each of those properties can be [2].

TypeScript does add some overhead to the developer. First, the code must be compiled into regular JavaScript to be run in the browser. Also, each library that is referenced within a TypeScript file must have a type definitions file. This means that for any smaller libraries that don't have type definitions, developers won't be able to get the benefits of typescript when interacting with that library [2].

2.3 Flow

Flow is similar to TypeScript in that both tools are used to provide static types to JavaScript. While TypeScript was made and is supported by Microsoft, Flow is made by Facebook. The type annotation system that Flow uses is very similar to TypeScript, but there are a few differences between the tools as well. Flow will build a deeper understanding of the code and does interprocedural analysis on the code. TypeScript, on the other hand, helps developers by providing intellisense (auto complete), code navigation, and refactoring [2].

3 Asset Bundlers

3.1 Webpack

Webpack has been the most used bundler for the past few years, which means that there is widespread support for it, as well as plenty of tutorials on configuring it. In the past, it has had a bad reputation for being complicated to configure, but in more recent versions, it is significantly easier to have work right out of the box. However, there is configuration required for features such as development server and code splitting. One thing Webpack excels at is allowing the developer full control over their build process, including specifying specific tools for specific filetypes (JavaScript, CSS, Images, etc.) [3].

3.2 Parcel

Parcel is a newer asset bundler that requires almost no configuration. It works by having the user directly include the entry file (that still needs to be compiled) in their HTML file, and then it resolves the dependency tree from there and outputs a modified HTML file, along with other dependencies. This could beneficial for the WebXR demo project, since the team wouldn't have to worry about spending the time to configure Webpack. However, it doesn't have as widespread of support since it is a newer package. It does have out of the box support for TypeScript and Rust, along with many other languages [3].

3.3 Rollup

Rollup is a recent addition to the Asset Bundler scene. It requires some configuration, and has works very similar to Webpack in that users specify an entry point, a destination, and an array of plugins that will be applied to the assets. However, Rollup's configuration is a little bit easier to use, but has less documentation than Webpack, especially when it comes to third party tutorial articles. Rollup also seemed to be much quicker than Webpack and Parcel, from tests done by X-Team in their article titled Rollup Webpack Parcel Comparison [3].

4 Continuous Integration

4.1 Circle CI

Circle CI (Continuous Integration) is a tool that automatically runs unit tests on developers' code whenever a commit is pushed to a branch on a source control server. In our case, we will be using GitHub. It is compatible with Node.js

(which will help for running unit tests of client-side code), along with many other languages. It works by listening for notifications from GitHub (or other source control providers), then pulling the branch into their cloud cluster and preforming a set of user-specified tasks. Typically, these tasks include building, testing, and publishing. These tasks are configured within a YAML config file [4]. Circle CI is free for open source and closed source projects, but one can pay to upgrade the resources the build is given.

Using CI such as Circle CI allows the developer to ensure code quality and that the tests are all passed. Developers can also create releases from within continuous integration, which means that the developer doesn't have to do it themselves manually [4].

4.2 Travis CI

Travis CI is very similar to Circle CI in that users can specify a YAML config with the tasks they want to execute on their code (typically building, testing, and publishing), and then have Travis run them after every source control push or merge. Although Travis CI has a different config schema, they both allow users to do similar tasks the same way. Both also run in the cloud, rather than on one's local computer or build server so there is little setup required. There are a few things that Travis supports that Circle CI does not. Travis CI has build matrixes, which means that one can specify different environments (such as language versions and operating systems) to build and test their code on [4]. Travis is free for open source projects.

4.3 Jenkins

Jenkins is one of the most well-known continuous integration systems in existance, partly because it has been around for a long time. Unlike Travis CI and Circle CI, it is a program that developers download to a build machine. This means that to use Jenkins, we would have to have our own build machine to run it on, then configure it to hook into GitHub. This could be a benefit for closed source projects that do not want to risk their source getting out into the world, but for open source projects, this is just another cost. Jenkins is also known to be challenging and time consuming to configure, so there is a time cost associated with using it as well. Other than those costs, it is very configurable and has a plugin system built into the tool. The software itself is also free [4].

5 Conclusion

One of the requirements of our project is for it to be able to run in the browser. Hence, the language we will need to use must compile to JavaScript. Three options for this are plain JavaScript (using the latetest specification so we get the most features), TypeScript, or Flow. My personal recommendation is to use TypeScript since there is widespread support for it, and it makes the code more maintainable in the long run. That will be beneficial to us since our project will be spanning the entire school year.

In terms of Asset Bundlers, there are three leading options in the community currently - Webpack, Parcel, and Rollup. I believe we should try Parcel first to see if it works for us, since there is no configuration and everything we need should

work right out of the box. If Parcel does not give us enough granularity of control, My second recommendation would be Webpack because it is widely used and has great documentation.

For Continuous Integration, there are several options. I narrowed it down to three popular choices: Circle CI, Travis CI, and Jenkins. Circle CI and Travis are both reasonable options for us because they come at no cost for open source projects and they do not require the developer to supply the build machine. They are also highly similar, and we do not need to worry about the code running on multiple versions of node, since we will be putting our code through Babel to compile it to ES5. Jenkins, on the other hand, is free for the software but requires a build machine, which will increase our costs. Therefore, it is not the best option for an open source project.

REFERENCES

[1] Babel, "What is babel?." https://babeljs.io/docs/en/, 2018.

[2] M. Schulz, "Typescript vs. flow." https://blog.mariusschulz.com/2017/01/13/typescript-vs-flow, 2017.

[3] A. Gerard, "Rollup v. webpack v. parcel." https://x-team.com/blog/rollup-webpack-parcel-comparison/, 2018.

[4] O. Shaporda, "Continuous integration. circleci vs travis ci vs jenkins." https://hackernoon.com/continuous-integration-circleci-vs-travis-ci-vs-jenkins-41a1c2bd95f5, 2017.