QA Consulting

Node Development Environment Course Handout

TRAINING MATERIALS - COURSE HANDOUT

Contacts

Elliot.Womack@qa.com

team.qac.all.trainers@qa.com

www.consulting.qa.com

Description

This document will be detailing what a Node Development Environment contains, recommending particular tools, and what each part of the environment is for.

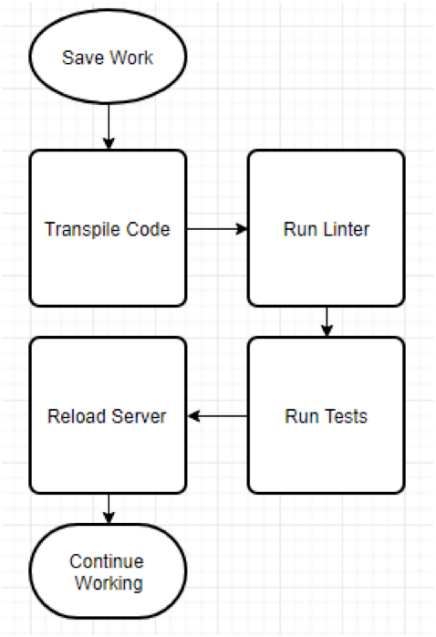
Tools

A Node development Environment is comprised of multiple different technologies that are linked together to promote a simple workflow, saving time and creating a safer development environment for the team.

A typical environment will be comprised of the following.

1. Build Tool
2. Transpiler
3. Linter
4. Testing
    - Mocking
    - Helper Libraries
5. Automation Toolkit

The environment creates its own pseudo-pipeline that would look like this.



BUILD TOOL

Also known as bundlers, build tools will generally move/compile/create files comprised of the files you have created.

For example, in your project you might create 20 separate javascript files, utilising other libraries within those files, you won't want your users to have to download all 20, so you could use a build tool/bundler to package these all together into one file, but then it will change your html references to those script files it created, not altering how your project works. Build tools can do a lot more than this though. Through the use of plugins you can enable other features that it can do to your project, creating a pseudo-pipeline that your code will go through until it hits a stage you can view it or deploy it for production.

Some example features that you'll be exposed to are;

- Minifying
  - The process of reducing the filesize of your code
- Uglifying (Obfuscating)
  - Making your code unreadable, as to protect your originality and improve security
- Hot Reloading
  - Reloading the page every save, speeding up development
- Tree shaking
  - Finding common libraries across packages and deleting duplicates

Example Build Tools

- Browserify
- Webpack
- Rollup
- JSPM

Webpack has been the most popular one for a few years now, and is constantly being updated.

webpack - https://webpack.js.org/

TRANSPILER

A transpiler is a tool that converts one set of code into another set of code, generally this is from one language to another. For example there's a JavaScript superset called Elm which is a functional version of javascript, however the syntax is vastly different so it would not run on a browser, so once you've finished developing you would transpile this Elm code into normal JavaScript code so it would run.

Numerous other languages like to transpile to JavaScript, since they like the ability of deploying where javascript can deploy (Anywhere!) But prefer coding in another language (Such as GoLang)

The transpilers that we are interested are a lot simpler than this, there is a transpiler called **babel** that lets us code to the latest version of JavaScript that is in draft, aka all the latest features of the language, however they're not all available on every environment, so babel will transpile it to a version that is available to run. This lets us use all the features of the latest version with none of the downsides of incompatibility!

babel - https://babeljs.io/

LINTER

Linting simply put is code analysis that will tell you problems will your code, whether that be invalid syntax or formatting errors/preferences that your team has imposed upon you.

Linters generally come with a default set of rules to follow but your team will create a custom set that they want to adhere too.

ESLint is the most popular linter currently and offers a broad range of customizability and configurability.

Linters are also part of the CI Pipeline, where if a linter detects an "error" or a certain number of warnings, it will break the build.

eslint - https://eslint.org/

TESTING

Setting up the unit tests in your environment is as important as every other part of the process, getting confirmation every time you save if all your tests are passing, and if they are not then you know exactly what you need to go and do.

Testing is also a part of the CI Pipeline, if any tests fail then the build will fail. Setting up a test environment requires multiple tools.

1. Testing Framework
2. Assertion Library
3. Helper Libraries
    - Mocking Tools

Mocha and **Jasmine** are the popular unit testing frameworks, Jasmine comes with an assertion library built in, a popular assertion library alongside Mocha is **Chai**. In the end most frameworks/assertion libraries accomplish the same thing, so it's not that important what you choose.

mocha - https://mochajs.org/
chai - https://chaijs.com/

Helper Libraries are important as they determine how effective your tests are, they help you create simpler/quicker tests that may enable you to test things you otherwise could not.

An example helper library is **JSDom**, you will be running your tests in **node** so there isn't inherently a document to open and check elements on the page through the **document** object. JSDom re-creates this **virtual dom** and then you can interact with the page like you would normally.

Another example library is **Cheerio**, which is effectively **jquery** for node, great if you're familiar with jquery and want it to create tests.

JSDom - https://www.npmjs.com/package/node-jsdom

Mocking Tools are libraries that will generate fake data or let you hot-swap environments.

For example, if you have an online database you generally fetch data from and then work with, if you wanted to work offline you wouldn't be able to test your code, so you could **mock** a database with some raw/fake data and then your code should then work with that data.

An example mocking setup would be to use **JSON Schema Faker** to generate the fake data and **JSON Server** to provide that data.

JSON Schema Faker - https://github.com/json-schema-faker/json-schema-faker
JSON Server - https://www.npmjs.com/package/json-server

AUTOMATION TOOLKIT

All of the things listed are the **bare minimum** and it's already a lot of tools, every time we save our code we don't want to have to run 5+ commands to run all our tests, our linting, reloading the server we're working on to see visual feedback and so forth, so we need *another* tool to bring all these together to do it for us, so every time we save this will all happen.

Popular automation toolkits are;

- Grunt (Old)
- Gulp (Replaced Grunt)
- NPM Scripts (Replaced Gulp)

Whilst Gulp has a lot of plugins that might make you want to choose it, choosing another tool instead of using a tool we're already using (npm) is an unnecessary abstraction and makes debugging a pain, also if gulp doesn't support a tool you want to use, you either need to create a plugin yourself or wait for one to be made. Whilst npm talks to the tools directly, not requiring extra downloads to set up to interlink these tools.

NPM Scripts Resource - https://css-tricks.com/why-npm-scripts/