AZILE MAGABA

FRONT END AND THE LANGUAGES

WHAT IS FRONT END

Refer to the development client side development where focus on the users visually see on the first in there application. Frond end are responsible for good look and feel of the site.

LANGUAGES BASIC AND THEORY

- HTML
- CSS
- JAVASCRIPT
- JQUERY

BASICS OF HTML

HTML stand for *Hypertext Markup Language*, is the common *markup language* for creating or related to a web pages. It is not a programming language, it is a markup language.

A **markup language** is a language for communicating to a web browser how the make of a page will be displayed. In HTML this extra information is communicated to the browser in the form of codes or "tags".
HTML is written in the form of "tags" that are surrounded by angle brackets like **start tag** <html> and **end tag** </html>.

An **HTML** file have **html** file extension as like **index.html** , **about_us.**which can identify that the page is a **web**

HTML Tags

Tags is the important part of Hypertext Markup language. **HTML Tags** give instruction a browser how to display the page content. In other words, **HTML Tag** is one kind of command, how and what type content will display in the browser window.

HTML tags are English words written by angle brackets like <html>. It has two parts. One is Opening tag or Start tag (written by angel bracket (**<** then **tag-name** and then again angel bracket (**>**), e.g **<h1>**) and another is Closing tag or end tag(written by angel bracket (**<**)then *backslash* or *Slash* (**/**) then **tag-name** then again angel bracket (**>**), e.g **</h1>**).
HTML tags normally come in pairs like opening tag <h1> and closing tag </h1> expect empty

ELEMENT.

How to start

HTML Tags

Tags is the important part of Hypertext Markup language. HTML Tags give instruction a browser how to display the page content. In other words, HTML Tag is one type of command, how and what type content will display in the browser window

HTML Syntax

A **HTML** document have two distinct parts, the head and the body. The head have information about the document that is not shown on the screen. The body is like this

<!DOCTYPE html>

<html>

<head>

<title></title>

</head>

<body>

</body>

</html>

Explained Above HTML Syntax:

- <DOCTYPE – Document types tells which HTML version you use .
- The <html> and </html> tags shows that this document is a web page or HTML document.
- The <head> and </head> tags are specifies page information which is not seen in a web page.
- The <title> and </title> tags are assigns to page name. title content generally you put at top of browser.

- The <body> and </body> tags have page contents which is visible in Browser window.

HTML Elements

HTML documents or web pages are defined by HTML elements. Say other hands a collection of HTML elements make a web page or HTML Documents.
HTML Elements has normally three parts.

- An HTML element starts with a start tag / opening tag. (**<h1>**)
- An HTML element ends with an end tag / closing tag. (</h1>
- The element content is everything between the start and the end tag.

BASICS OF CSS

Cascading Style Sheets have been around since the end of 1996. Despite the relative longevity of the technology, its use in real-world web design has been limited to managing fonts and color, at least until recently. This limitation was imposed by the lack of consistent browser support. Because not all browsers managed CSS equally (if at all), it has been very difficult for designers to tap into the true power of style sheets. Instead, there's been a reliance on HTML for presentation.

Now we have far better support for CSS, so to tap into its many valuable features, web designers are moving away from HTML as a means of adding style and laying out pages, and into pure CSS design. Why is this so important? The reasons are many:

- Keeping presentation separate from the document means you can style that document for numerous media, including the screen, print, projection, and even handheld devices.
- Separating presentation from the document means a lighter document, which, in turn, means the page loads and renders faster, making for happier visitors.
- CSS offers ways to control one document or millions of documents. Any time you'd like to make a change, you change that style in one location and automatically update to all the documents with which that CSS is connected. In HTML, this couldn't be done.
- CSS documents are cached. This means they are loaded into your browser's memory one time. As you move within a site, the browser never has to reinterpret the styles. The results are more fluid movement from page to page and faster-loading pages, which, of course, is always desirable.
- By separating presentation from structure and content, accessibility is easily achieved. Documents that don't have heavy tables and lots of presentational HTML are inherently

CSS Rules

CSS rules are made up of a selector and at least one declaration. A selector is the code that selects the HTML to which you want to apply the style rule. We'll be focusing on common selectors in this tutorial, but you can use more than a dozen selector types as you become more proficient at CSS. A declaration is made up of at least one CSS property and related property value. CSS properties define the style:

```
h1 {color: red;}
```

The H1 is the selector for your h1 headers, and the declaration is made up of the color property with a value of red. Simply said, this rule turns all H1 elements red. You'll note that the syntax of the style rule looks different from what you've seen in HTML. The curly braces contain the declarations, each property is followed by a colon, and a semicolon is used after each property. You can have as many properties as you want in a rule.

Applying CSS

Six types of style sheets exist:

- Browser style This is the default style sheet within a browser. If you declare no style rules, these defaults are applied.
- User style A user can write a style sheet and make it override any styles you create by changing a setting in the browser. These are used infrequently but can be helpful for individuals with special needs, such as low vision. In such a case, the user will create high-contrast, large-font styles that override your own.
- Inline style This is style that is used right in the individual element and applied via the style attribute. It can be very useful for one-time styles by element, but it isn't considered ideal.
- Embedded style This is style that controls one document and is placed inside the style element within the HTML document.
- Linked style This is a style sheet that is linked to an HTML document using the link element in the head of the document. Any document linked to this style sheet gets the styles, and here's where the management power of CSS is found.
- Imported style this is similar to linked styles, but it enables you to import styles into a linked style sheet as well as directly into a document. This is useful in workarounds and when managing many documents.

You'll see examples of these style sheets as we progress throughout the tutorial.

Basics of using the preprocessor

Anyone who has ever read a piece of C source code has seen them—the preprocessor directives. For example, you can find include directives (#include) at the beginning of most source files. The preprocessor is a system that rewrites the source before the actual compiler sees it. Clearly, this is a very powerful tool—the downside is that you could accidentally shoot yourself in the foot.

In this article we will introduce the preprocessor and covered the basics, including object- and function-like macros, the include directive, conditional compilation, and end with the two special directives #error and #pragma.

The #include directive

The most straight-forward preprocessor directive is #include. When the preprocessor finds this directive it simply opens the file specified and inserts the content of it, as though the content of the file would have been written at the location of the directive.

It can take two forms, for example:

#include <systemfile.h>

#include "myfile.h"

What is Sass?

- **Sass** stands for **S**yntactically **A**wesome **S**tyle sheet
- Sass is an extension to CSS
- Sass is a CSS pre-processor
- Sass is completely compatible with all versions of CSS
- Sass reduces repetition of CSS and therefore saves time
- Sass was designed by Hampton Catlin and developed by Natalie Weizenbaum in 2006
- Sass is free to download and use
- Why Use Sass?
- Stylesheets\ are getting larger, more complex, and harder to maintain. This is where a CSS pre-processor can help.
- Sass lets you use features that do not exist in CSS, like variables, nested rules, mixins, imports, inheritance, built-in functions, and other stuff.
-
- A Simple Example why Sass is Useful
- Let's say we have a website with three main colors:

- #a2b9bc
- #b2ad7f
- #878f99
- So, how many times do you need to type those HEX values? A LOT of times. And what about variations of the same colors?
- Instead of typing the above values a lot of times, you can use Sass and write this:
- **Sass Example**

```
/* define variables for the primary colors */
$primary_1: #a2b9bc;
$primary_2: #b2ad7f;
$primary_3: #878f99;

/* use the variables */
.main-header {
  background-color: $primary_1;
}

.menu-left {
  background-color: $primary_2;
}

.menu-right {
  background-color: $primary_3;
}
```

How Does Sass Work?

A browser does not understand Sass code. Therefore, you will need a Sass pre-processor to convert Sass code into standard CSS.

This process is called transpiling. So, you need to give a transpiler (some kind of program) some Sass code and then get some CSS code back.

Tip: Transpiling is a term for taking a source code written in one language and transform/translate it into another language.

WHAT IS JAVASCRIPT

JavaScript is a scripting or programming language is the light height and the mostly commonly

Used as the part of webpage.whose implementation allow client side script to interact with user and makes dynamic pages.

What is JavaScript doing on your page?

- JavaScript is alight interpreted programming languages
- Designing for creating network centric applications
- Open and cross plat forms
- Complementary to and integrated java
- Complementary to and integrated with html

Here is example of JavaScript code

```
<p id="demo"></p>


<script>

document.getElementById("demo").innerHTML = "Hello JavaScript!";

</script>


</body>

</html>
```

REACT JAVASCRIPT

React is a free and open-source front-end JavaScript library for building user interfaces or UI components. It is maintained by Facebook and a community of individual developers and companies. React can be used as a base in the development of single-page or mobile applications.

WHY LEARN REACTJS?

ReactJS offers graceful solutions to some of front-end programming's most persistent issues, allowing you to build dynamic and interactive web apps with ease. It's fast, scalable, flexible,

powerful, and has a robust developer community that's rapidly growing. There's never been a better time to learn React. You'll develop a strong understanding of React's most essential concepts: JSX, class and function components, props, state, lifecycle methods, and hooks. You'll be able to combine these ideas in Reacts modular programming style.

Why React?

React.js is a JavaScript library. It was developed by engineers at Facebook.

Here are just a few of the reasons why people choose to program with React:

- React is *fast*. Apps made in React can handle complex updates and still feel quick and responsive.
- React is *modular*. Instead of writing large, dense files of code, you can write many smaller, reusable files. React's modularity can be a beautiful solution to JavaScript's maintainability problems.
- React is *scalable*. Large programs that display a lot of changing data are where React performs best.
- React is *flexible*. You can use React for interesting projects that have nothing to do with making a web app. People are still figuring out React's potential. There's room to explore.
- React is *popular*. While this reason has admittedly little to do with React's quality, the truth is that understanding React will make you more employable.

If you are new to React, then this course is for you! No prior React knowledge is expected. We will start at the very beginning and move slowly.

If you are new to JavaScript, then consider taking our JavaScript course and then returning to React.

The Codecademy React courses are not a high-level overview. They are a deep dive. Take your time! By the end.

Tip

This tutorial is designed for people who prefer to **learn by doing**. If you prefer learning concepts from the ground up, check out our step-by-step guide. You might find this tutorial and the guide complementary to each other.

WHAT IS REACT?

React is a declarative, efficient, and flexible JavaScript library for building user interfaces. It lets you compose complex UIs from small and isolated pieces of code called "components".

React has a few different kinds of components, but we'll start with React.Component subclasses:

```
class ShoppingList extends React.Component {
  render() {
    return (
      <div className="shopping-list">
        <h1>Shopping List for {this.props.name}</h1>
        <ul>
          <li>Instagram</li>
          <li>WhatsApp</li>
          <li>Oculus</li>
        </ul>
      </div>
    );
  }
}

// Example usage: <ShoppingList name="Mark" />
```

What is Vue.js

The **Vue.js** story begins in 2013 when Evan You was working at Google creating lots of prototypes right within a browser. For that purpose, Evan used handy practices from other frameworks he worked with, and released Vue.js officially in 2014.

Vue.js is a progressive framework for JavaScript used to build web interfaces and one-page applications. Not just for web interfaces, Vue.js is also used both for desktop and mobile app development with Electron framework. The HTML extension and the JS base quickly made Vue a favored front-end tool, evidenced by adoption by such giants as Adobe, Behance, Alibaba, Galba, and Xiaomi.

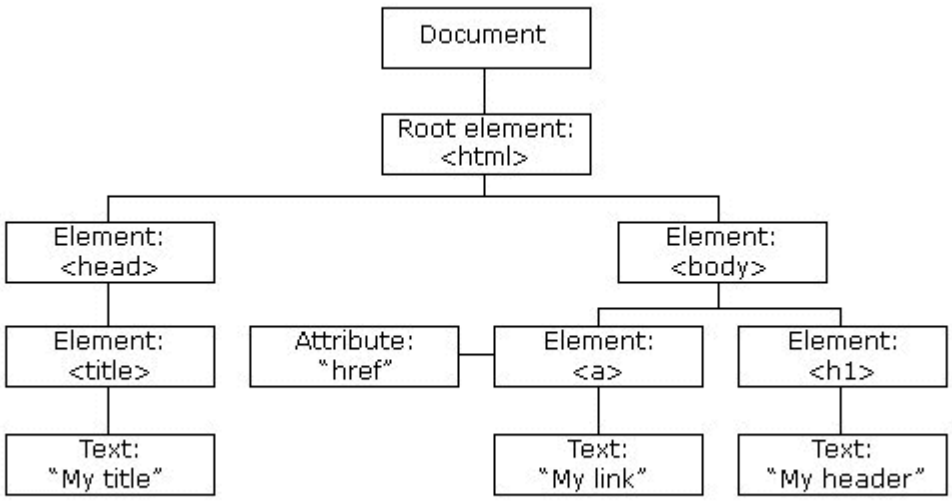
In an interview with Evan, he reveals that initially Vue.js was an attempt to take the best of Angular and build a custom tool, but a lighter weight one: "*For me, Angular offered something cool which is data binding and a data driven way of dealing with a DOM, so you don't have to touch the DOM yourself*."

The name of the framework – Vue – is the same phonetically in English as *view*, and it

The name of the framework – Vue – is the same phonetically in English as *view*, and it corresponds to the traditional *Model-View-Controller* (MVC) architecture. Simply put, view is a UI of an application/website, and the core library of Vue.js focuses the view layer by default. But, MVC doesn't mean that Vue.js can't be used with a different architectural approach like the *Component Based Architecture* (CBA) used in React.

Virtual DOM rendering and performance

A *Document Object Model* (DOM) is something you'll probably encounter when rendering web pages. A DOM is a representation of HTML pages with its styles, elements, and page content as objects. The objects stored as a tree structure are generated by a browser when loading a page.



Introduction to Angular concepts

Angular is a platform and framework for building single-page client applications using HTML and TypeScript. Angular is written in TypeScript. It implements core and optional functionality as a set of TypeScript libraries that you import into your applications.

The architecture of an Angular application relies on certain fundamental concepts. The basic building blocks of the Angular framework are Angular components that are organized into *NgModules*. NgModules collect related code into functional sets; an Angular application is defined by a set of NgModules. An application always has at least a *root module* that enables bootstrapping, and typically has many more *feature modules*.

- Components define *views*, which are sets of screen elements that Angular can choose among and modify according to your program logic and data.
- Components use *services*, which provide specific functionality not directly related to views. Service providers can be *injected* into components as *dependencies*, making your code modular, reusable, and efficient.

Modules, components and services are classes that use *decorators*. These decorators mark their type and provide metadata that tells Angular how to use them.

- The metadata for a component class associates it with a *template* that defines a view. A template combines ordinary HTML with Angular *directives* and *binding markup* that allow Angular to modify the HTML before rendering it for display.
- The metadata for a service class provides the information Angular needs to make it available to components through *dependency injection (DI)*.

An application's components typically define many views, arranged hierarchically. Angular provides the Router service to help you define navigation paths among views. The router provides sophisticated in-browser navigational capabilities.

See the Angular Glossary for basic definitions of important Angular terms and usage.

For the sample application that this page describes, see the live example / download example.

Modules

Angular *NgModules* differ from and complement JavaScript (ES2015) modules. An NgModule declares a compilation context for a set of components that is dedicated to an application domain, a workflow, or a closely related set of capabilities. An NgModule can associate its components with related code, such as services, to form functional units.

Every Angular application has a *root module*, conventionally named AppModule, which provides the bootstrap mechanism that launches the application. An application typically contains many functional modules.

Like JavaScript modules, NgModules can import functionality from other NgModules, and allow their own functionality to be exported and used by other NgModules. For example, to use the router service in your app, you import the Router NgModule.

Organizing your code into distinct functional modules helps in managing development of complex applications, and in designing for reusability. In addition, this technique lets you take advantage of *lazy-loading*—that is, loading modules on demand—to minimize the amount of code that needs to be loaded at startup.

For a more detailed discussion, see Introduction to modules.

Components

Every Angular application has at least one component, the *root component* that connects a component hierarchy with the page document object model (DOM). Each component defines a class that contains application data and logic, and is associated with an HTML *template* that defines a view to be displayed in a target environment.

The @Component() decorator identifies the class immediately below it as a component, and provides the template and related component-specific metadata.

Decorators are functions that modify JavaScript classes. Angular defines a number of decorators that attach specific kinds of metadata to classes, so that the system knows what those classes mean and how they should work.

Learn more about decorators on the web.

Templates, directives, and data binding

A template combines HTML with Angular markup that can modify HTML elements before they are displayed. Template *directives* provide program logic, and *binding markup* connects your application data and the DOM. There are two types of data binding:

- *Event binding* lets your application respond to user input in the target environment by updating your application data.
- *Property binding* lets you interpolate values that are computed from your application data into the HTML.

Before a view is displayed, Angular evaluates the directives and resolves the binding syntax in the template to modify the HTML elements and the DOM, according to your program data and logic. Angular supports *two-way data binding*, meaning that changes in the DOM, such as user choices, are also reflected in your program data.

Your templates can use *pipes* to improve the user experience by transform

What is Node.js?

Node.js is a server-side platform built on Google Chrome's JavaScript Engine (V8 Engine). Node.js was developed by Ryan Dahl in 2009 and its latest version is v0.10.36. The definition of Node.js as supplied by its official documentation is as follows −

Node.js is a platform built on Chrome's JavaScript runtime for easily building fast and scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.

Node.js is an open source, cross-platform runtime environment for developing server-side and networking applications. Node.js applications are written in JavaScript, and can be run within the Node.js runtime on OS X, Microsoft Windows, and Linux.

Node.js also provides a rich library of various JavaScript modules which simplifies the development of web applications using Node.js to a great extent.

Node.js = Runtime Environment + JavaScript Library

Features of Node.js

What is Typescript?

JavaScript and More

TypeScript is an open-source language which builds on JavaScript, one of the world's most used tools, by adding static type definitions.

Types provide a way to describe the shape of an object, providing better documentation, and allowing TypeScript to validate that your code is working correctly.

Writing types can be optional in TypeScript, because type inference allows you to get a lot of power without writing additional code.

A Result You Can Trust

All valid JavaScript code is also TypeScript code. You might get type-checking errors, but that won't stop you from running the resulting JavaScript. While you can go for stricter behavior, that means you're still in control.

TypeScript code is transformed into JavaScript code via the TypeScript compiler or Babel. This JavaScript is clean, simple code which runs anywhere JavaScript runs: In a browser, on Node.JS or in your apps.

Gradual Adoption

Adopting TypeScript is not a binary choice, you can start by annotating existing JavaScript with JSDoc, then switch a few files to be checked by TypeScript and over time prepare your codebase to convert completely.

TypeScript's type inference means that you don't have to annotate your code until you want more safety

What You Should Already Know

Before you start studying jQuery, you should have a basic knowledge of:

- HTML
- CSS
- JavaScript

If you want to study these subjects first, find the tutorials on our Home page.

WHAT IS JQUERY?

jQuery is a lightweight, "write less, do more", JavaScript library.

The purpose of jQuery is to make it much easier to use JavaScript on your website.

jQuery takes a lot of common tasks that require many lines of JavaScript code to accomplish, and wraps them into methods that you can call with a single line of code.

jQuery also simplifies a lot of the complicated things from JavaScript, like AJAX calls and DOM manipulation.

The jQuery library contains the following features:

- HTML/DOM manipulation
- CSS manipulation
- HTML event methods
- Effects and animations
- AJAX
- Utilities

Tip: In addition, jQuery has plugins for almost any task out there.

Why jQuery?

There are lots of other JavaScript libraries out there, but jQuery is probably the most popular, and also the most extendable.

Many of the biggest companies on the Web use jQuery, such as:

- Google
- Microsoft
- IBM
- Netflix

Will jQuery work in all browsers?

The jQuery team knows all about cross-browser issues, and they have written this knowledge into the jQuery library. jQuery will run exactly the same in all major browsers.

❮ PreviousNext ❯

Downloading jQuery

There are two versions of jQuery available for downloading:

- Production version - this is for your live website because it has been minified and compressed
- Development version - this is for testing and development (uncompressed and readable code)

Both versions can be downloaded from jQuery.com.

The jQuery library is a single JavaScript file, and you reference it with the HTML <script> tag (notice that the <script> tag should be inside the <head> section):

```
<head>
<script src="jquery-3.5.1.min.js"></script>
</head>
```

What is Redux?

Redux is a predictable state container designed to help you write JavaScript apps that behave consistently across client, server, and native environments and are easy to test.

While it's mostly used as a state management tool with React, you can use it with any other JavaScript framework or library. It's lightweight at 2KB (including dependencies), so you don't have to worry about it making your application's asset size bigger.

With Redux, the state of your application is kept in a store, and each component can access any state that it needs from **Actions in Redux**

Simply put, actions are events. They are the only way you can send data from your application to your Redux store. The data can be from user interactions, API calls, or even form submissions.

Actions are sent using the store.dispatch() method. Actions are plain JavaScript objects, and they must have a type property to indicate the type of action to be carried out. They must also have a payload that contains the information that should be worked on by the action. Actions are created via an action creator.

Here's an example of an action that can be carried out during login in an app:

```
{
  type: "LOGIN",
  payload: {
    username: "foo",
    password: "bar"
  }
}
```

Here is an example of its action creator:

```
const setLoginStatus = (name, password) => {
  return {
    type: "LOGIN",
```

```
    payload: {
      username: "foo",
      password: "bar"
    }
  }
}
```

FOR MORE INFORMATION

Render webpages using the Flutter WebView plugin

References

- webview_flutter: v2.0.10
- WebView class

TASK RUNNERS

Task runners are the heroes (or villains, depending on your point of view) that quietly toil behind most web and mobile applications. Task runners provide value through the automation of numerous development tasks such as concatenating files, spinning up development servers and compiling code.

That's our goal in this article, to quickly cover the basics of the most popular task runners and to provide solid examples to kickstart your imagination regarding how these tools can fit in your workflow.

Further Reading On SmashingMag: #

- Become A Command-Line Power User With Oh-My-ZSH And Z

- An Introduction To PostCSS
- Get Up And Running With Grunt
- Building With Gulp

Grunt #

Grunt was the first popular JavaScript-based task runner. I've been using Grunt in some form since 2012. The basic idea behind Grunt is that you use a special JavaScript file, Gruntfile.js, to configure various plugins to accomplish tasks. It has a vast ecosystem of plugins and is a very mature and stable tool. Grunt has a fantastic web directory that indexes the majority of plugins (about 5,500 currently). The simple genius of Grunt is its combination of JavaScript and the idea of a common configuration file (like a makefile), which has allowed many more developers to contribute to and use Grunt in their projects. It also means that Grunt can be placed under the same version control system as the rest of the project.

Creating plugins

1. Install grunt-init with npm install -g grunt-init
2. Install the gruntplugin template with git clone git://github.com/gruntjs/grunt-init-gruntplugin.git ~/.grunt-init/gruntplugin (%USERPROFILE%\.grunt-init\gruntplugin on Windows).
3. Run grunt-init gruntplugin in an empty directory.
4. Run npm install to prepare the development environment.
5. Author your plugin.
6. Run npm publish to publish the Grunt plugin to npm!

Notes

Naming your task

The "grunt-contrib" namespace is reserved for tasks maintained by the Grunt team, please name your task something appropriate that avoids that naming scheme.

The three different files represent the various capabilities of browsers and mobile devices. Modern devices will receive the high-resolution SVGs as a single request (i.e. a single CSS file).

Browsers that don’t handle SVGs but handle base-64-encoded assets will get the base-64 PNG style sheet. Finally, any browsers that can’t handle those two scenarios will get the “traditional” style sheet that references PNGs. All this from a single directory of SVGs!

```
module.exports = function(grunt) {

 grunt.config("grunticon", {
  icons: {
   files: [
    {
     expand: true,
     cwd: 'grunticon/source',
     src: ["*.svg", ".png"],
     dest: 'dist/grunticon'
    }
   ],
   options: [
    {
     colors: {
      "blue": "blue"
     }
    }
   ]
  }
 });
```

```
  grunt.loadNpmTasks('grunt-grunticon');
};
```

Gulp: LEGO Blocks For Your Build System #

Gulp emerged sometime after Grunt and aspired to be a build tool that wasn't all configuration but actual code. The idea behind code over configuration is that code is much more expressive and flexible than the modification of endless config files. The hurdle with Gulp is that it requires more technical knowledge than Grunt. You will need to be familiar with the Node.js streaming API and be comfortable writing basic JavaScript.

Gulp's use of Node.js streams is the main reason it's faster than Grunt. Using streams means that, instead of using the file system as the "database" for file transformations, Gulp uses in-memory transformations. For more information on streams, check out the Node.js streams API documentation, along with the stream handbook.

AN EXAMPLE #

Gulp has the same massive ecosystem of plugins as Grunt. So, to make this task easy, we're going to lean on the gulp-concat plugin. Let's say our project's structure looks like this:

```
|-- dist
|   `-- app.js
|-- gulpfile.js
|-- package.json
`-- src
    |-- bar.js
    `-- foo.js
```

Two JavaScript files are in our src directory, and we want to combine them into one file, app.js, in our dist/ directory. We can use the following Gulp task to accomplish this.

```
var gulp = require('gulp');
var concat = require('gulp-concat');
```

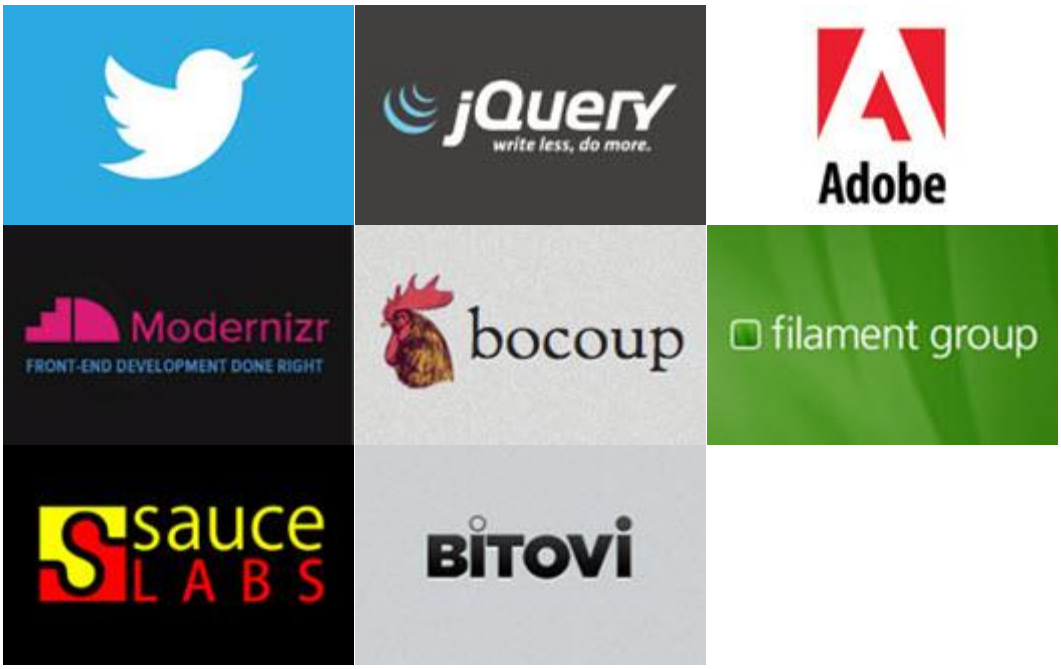
```
gulp.task('default', function() {
  return gulp.src('./src/*.js')
    .pipe(concat('app.js'))
    .pipe(gulp.dest('./dist/'));
});
```

Available Grunt plugins

Many of the tasks you need are already available as Grunt Plugins, and new plugins are published every day. While the plugin listing is more complete, here's a few you may have heard of.

Who uses Grunt?

These are just a few companies and projects that are using Grunt.See more here.



How do I get started?

Read the Getting Started guide for instructions on using Grunt in your projects. Once you've done that, check out a Sample Gruntfile and learn more about Configuring Tasks.

Grunt is an OpenJS Foundation project.

Latest Version

- Stable: v1.4.0 (npm)
- Development: HEAD (GitHub)

*Ads by **Bocoup**.*

Latest News

- Grunt 1.4.0 releasedApril 22, 2021
- Grunt 1.2.0 releasedJuly 03, 2020
- Grunt 1.1.0 releasedMarch 16, 2020

Getting Started

Read the Getting Started guide for instructions on using Grunt in your projects. Once you've done that, check out a Sample Gruntfile and learn more about Configuring Tasks.

Webpack #

The newest addition to the JavaScript task runner club is Webpack. Webpack bills itself as a "module bundler," which means it can dynamically build a bundle of JavaScript code from multiple separate files using module patterns such as the CommonJS pattern. Webpack also has plugins, which it calls loaders.

```
module.exports = {
   entry: "./src/foo.js",
   output: {
      filename: "app.js",
      path: "./dist"
   }
};
//foo.js
var bar = require("./bar");

var foo = function() {
  console.log('foo');
  bar();
```

```
};


module.exports = foo;
```

Npm Scripts #

npm scripts are the latest hipster craze, and for good reason. As we've seen with all of these tools, the number of dependencies they might introduce to a project could eventually spin out of control. The first post I saw advocating for npm scripts as the entry point for a build process was by James Halliday. His post perfectly sums up the ignored power of npm scripts (emphasis mine):

This example is a little more advanced than a typical npm script task. A typical npm script task is a call to a command-line tool, with the appropriate flags or config file. Here's a more typical task that compiles our Sass to CSS:

```
"sass": "node-sass src/scss/ -o dist/css",
```

About npm
Table of contents

npm is the world's largest software registry. Open source developers from every continent use npm to share and borrow packages, and many organizations use npm to manage private development as well.

npm consists of three distinct components:

- the website
- the Command Line Interface (CLI)
- the registry

Use the *website* to discover packages, set up profiles, and manage other aspects of your npm experience. For example, you can set up organizations to manage access to public or private packages.

The *CLI* runs from a terminal, and is how most developers interact with npm.

The *registry* is a large public database of JavaScript software and the meta-information surrounding it.

CLI reference documentation

While relevant CLI commands are covered throughout this user documentation, the CLI includes command line help, its own documentation section, and instant help (man pages).

Bower

Bower offers a generic, unopinionated solution to the problem of **front-end package management**, while exposing the package dependency model via an API that can be consumed by a more opinionated build stack. There are no system wide dependencies, no dependencies are shared between different apps, and the dependency tree is flat.

Bower runs over Git, and is package-agnostic. A packaged component can be made up of any type of asset, and use any type of transport (e.g., AMD, CommonJS, etc.).

View complete docs on bower.io

Install

$ npm install -g bower

Bower depends on Node.js and npm. Also make sure that git is installed as some bower packages require it to be fetched and installed.

Usage

See complete command line reference at bower.io/docs/api/

Installing packages and dependencies

```
# install dependencies listed in bower.json
$ bower install

# install a package and add it to bower.json
$ bower install <package> --save
```

```
# install specific version of a package and add it to bower.json
$ bower install <package>#<version> --save
```

The Command Line

There are a lot of different ways to use Git. There are the original command-line tools, and there are many graphical user interfaces of varying capabilities. For this book, we will be using Git on the command line. For one, the command line is the only place you can run **all** Git commands — most of the GUIs implement only a partial subset of Git functionality for simplicity. If you know how to run the command-line version, you can probably also figure out how to run the GUI version, while the opposite is not necessarily true. Also, while your choice of graphical client is a matter of personal taste, **all** users will have the command-line tools installed and available.

So we will expect you to know how to open Terminal in macOS or Command Prompt or PowerShell in

Windows. If you don't know what we're talking about here, you may need to stop and research that quickly so that you can follow the rest of the examples and descriptions in this book.

Related Articles

Using Redis with Node.js

JAVASCRIPT

By Ivaylo Gerchev, July 13, 2021

Need fast data interactions in your Node app? Learn how Redis speeds caching, message brokering, sessions, analytics, streaming and more.

Advanced Tips

Advanced Git Tutorials

Merging vs. Rebasing

Resetting, Checking Out, and Reverting

Advanced Git log

Git Hooks

AZILE MAGABA

Follow me

AZILE MAGABA

https://www.linkedin.com/in/azile-portia-magaba-b6b312216/

azile.magaba@younglings.africa

AZILE MAGABA