

FREE Book: JavaScript Best Practice

Check it out!

(<https://www.sitepoint.com/>).



(https://www.sitepoint.com/premium/products/Z2lkOi8vbGVhcm5hYmxlL1Byb2R1Y3QvMzAy?utm_source=smartbar&utm_medium=article&utm_campaign=free-book)

JavaScript (<https://www.sitepoint.com/javascript/>) > May 29, 2018 > By James Hibbard (<https://www.sitepoint.com/author/jhibbard/>).

Build a Simple Beginner App with Node, Bootstrap & MongoDB

If you're just getting started with Node.js and want to try your hand at building a web app, things can often get a little overwhelming. Once you get beyond the "Hello, World!" tutorials, much of the material out there has you copy-pasting code, with little or no explanation as to what you're doing or why.

This means that, by the time you've finished, you've built something nice and shiny, but you also have relatively few takeaways that you can apply to your next project.

In this tutorial, I'm going to take a slightly different approach. Starting from the ground up, I'll demonstrate how to build a no-frills web app using Node.js, but instead of focusing on the end result, I'll focus on a range of things you're likely to encounter when building a real-world app. These include routing, templating, dealing with forms, interacting with a database and even basic authentication.

This won't be a JavaScript 101. If that's the kind of thing you're after, [look here \(https://www.sitepoint.com/beginners-guide-javascript-variables-and-datatypes/\)](https://www.sitepoint.com/beginners-guide-javascript-variables-and-datatypes/). It will, however, be suitable for those people who feel reasonably confident with the JavaScript language, and who are looking to take their first steps in Node.js.

What We'll Be Building

We'll be using [Node.js \(https://nodejs.org/en/\)](https://nodejs.org/en/) and the [Express framework \(http://expressjs.com/\)](http://expressjs.com/) to build a simple registration form with basic validation, which persists its data to a [MongoDB database \(https://www.mongodb.com/\)](https://www.mongodb.com/). We'll add a view to list successful registration, which we'll protect with basic HTTP authentication, and we'll use [Bootstrap \(https://getbootstrap.com/\)](https://getbootstrap.com/) to add some styling. The tutorial is structured so that you can follow along step by step. However, if you'd like to jump ahead and see the end result, [the code for this tutorial is also available on GitHub \(https://github.com/jameshibbard/demo-node-app\)](https://github.com/jameshibbard/demo-node-app).

Basic Setup



Before we can start coding, we'll need to get Node, npm and MongoDB installed on our machines. I won't go into depth on the various installation instructions, but if you have any trouble getting up, please have a look at the link below, or [visit our forums](https://www.sitepoint.com/community/) (<https://www.sitepoint.com/community/>), and ask for help there.

Check it out!

(<https://www.sitepoint.com/>).



(https://www.sitepoint.com/premium/products/Z2lkOi8vbGVhcm5hYmxlL1Byb2R1Y3QvMzAy?utm_source=smartbar&utm_medium=article&utm_campaign=free-book)

Many websites will recommend that you head to [the official Node download page](https://nodejs.org/en/download/) (<https://nodejs.org/en/download/>), and grab the Node binaries for your system. While that works, I would suggest that you use a version manager instead. This is a program which allows you to install multiple versions of Node and switch between them at will. There are various advantages to using a version manager, for example it negates potential permission issues which would otherwise see you installing packages with admin rights.

If you fancy going the version manager route, please consult our quick tip: [Install Multiple Versions of Node.js Using nvm](https://www.sitepoint.com/quick-tip-multiple-versions-node-nvm/) (<https://www.sitepoint.com/quick-tip-multiple-versions-node-nvm/>). Otherwise, grab the correct binaries for your system from the link above and install those.

npm

npm is a JavaScript package manager which comes bundled with Node, so no extra installation is necessary here. We'll be making quite extensive use of npm throughout this tutorial, so if you're in need of a refresher, please consult: [A Beginner's Guide to npm — the Node Package Manager](https://www.sitepoint.com/beginners-guide-node-package-manager/) (<https://www.sitepoint.com/beginners-guide-node-package-manager/>).

MongoDB

MongoDB is a document database which stores data in flexible, JSON-like documents.

The quickest way to get up and running with Mongo is to use a service such as mLabs. They have a free sandbox plan which provides a single database with 496 MB of storage running on a shared virtual machine. This is more than adequate for a simple app with a handful of users. If this sounds like the best option for you, please consult their [quick start guide](http://docs.mlab.com/) (<http://docs.mlab.com/>).

You can also install Mongo locally. To do this, please visit the [official download page](https://www.mongodb.com/download-center#community) (<https://www.mongodb.com/download-center#community>), and download the correct version of the community server for your operating system. There's a link to detailed, OS-specific installation instructions beneath every download link, which you can consult if you run into trouble.

A MongoDB GUI

Although not strictly necessary for following along with this tutorial, you might also like to install [Compass, the official GUI for MongoDB](https://www.mongodb.com/download-center#compass) (<https://www.mongodb.com/download-center#compass>). This tool helps you visualize and manipulate your data, allowing you to interact with documents with full CRUD functionality.

At the time of writing, you'll need to fill out your details to download Compass, but you won't need to create an account.

Check that Everything Is Installed Correctly

To check that Node and npm are installed correctly, open your terminal and type:



```
node -v
```

FREE Book: JavaScript Best Practice

Check it out!

(<https://www.sitepoint.com/>)



Followed by:
(https://www.sitepoint.com/premium/products/Z2lkOi8vbGVhcm5hYmxlL1Byb2R1Y3QvMzAy?utm_source=smartbar&utm_medium=article&utm_campaign=free-book)

```
npm -v
```

This will output the version number of each program (**8.9.4** and **5.6.0** respectively at the time of writing).

If you installed Mongo locally, you can check the version number using:

```
mongo --version
```

This should output a bunch of information, including the version number (**3.6.2** at the time of writing).

Check the Database Connection Using Compass

If you have installed Mongo locally, you start the server by typing the following command into a terminal:

```
mongod
```

Next, open Compass. You should be able to accept the defaults (server: **localhost**, port: 27017), press the *CONNECT* button, and establish a connection to the database server.



FREE Book: JavaScript Best Practice

localhost:27017

MongoDB 3.6.2 Community

Check it out!

(https://www.sitepoint.com/)

sitepoint (https://www.sitepoint.com/premium/products/Z2lkOi8vbGVhcm5hYmxlL1Byb2R1Y3QvMzAy?utm_source=smartbar&utm_medium=article&utm_campaign=free-book)

> admin

> local

Database Name

Storage Size

Collections

Indexes

admin

16.0KB

0

2

local

32.0KB

1

1

MongoDB Compass connected to localhost

Note that the databases `admin` and `local` are created automatically.

Using a Cloud-hosted Solution



If you're using mLab, create a database subscription (as described in their [quick-start guide \(http://docs.mlab.com/\)](http://docs.mlab.com/)), then copy the connection details to the clipboard. This should be in the form:

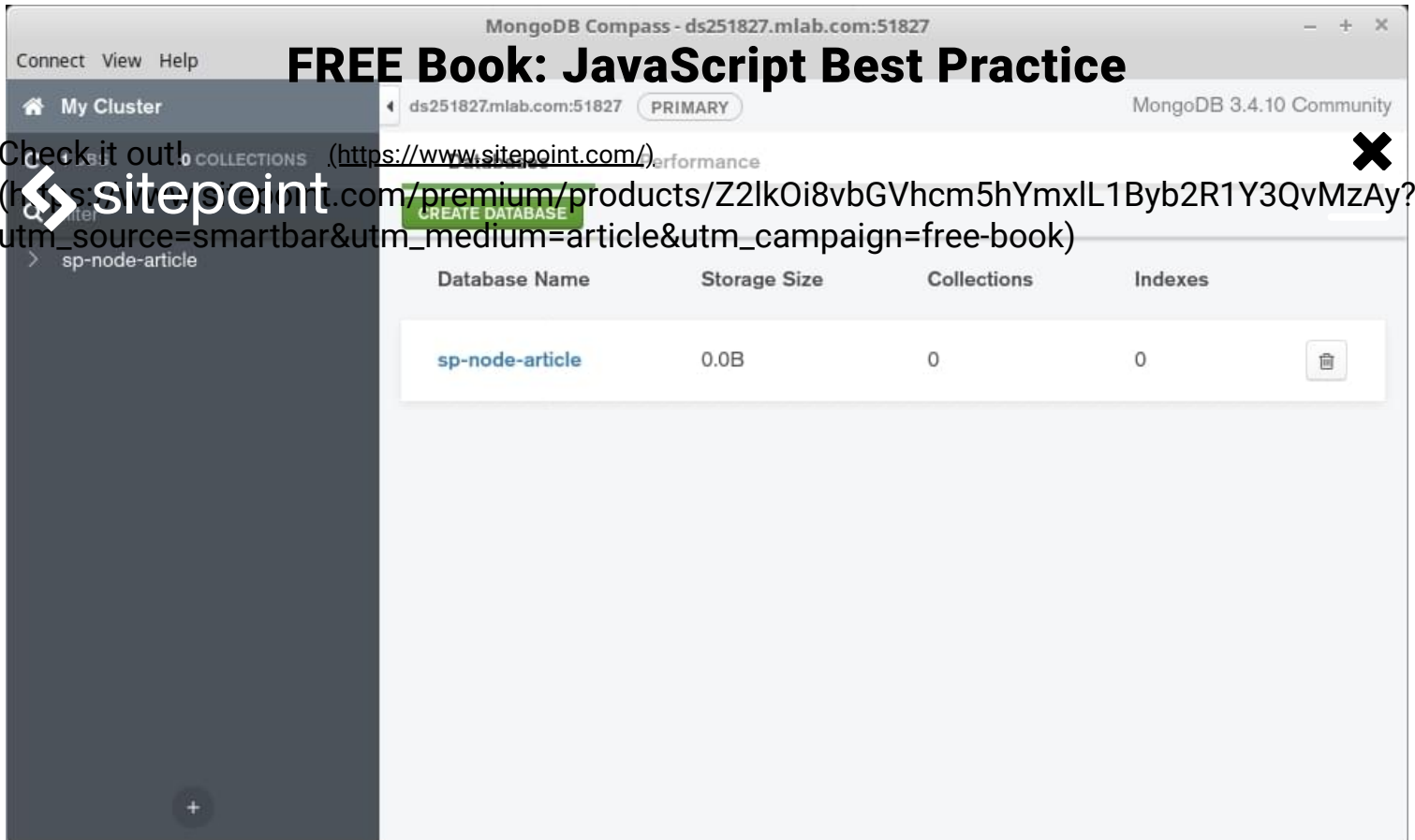
```
mongodb://<dbuser>:<dbpassword>@ds251827.mlab.com:51827/<dbname>
```

When you open Compass, it will inform you that it has detected a MongoDB connection string and asks if you would like to use it to fill out the form. Click Yes, noting that you might need to adjust the username and password by hand. After that, click **CONNECT** and you should be off to the races.



FREE Book: JavaScript Best Practice

Check it out! (https://www.sitepoint.com/)  
(https://www.sitepoint.com/premium/products/Z2lkOi8vbGVhcm5hYmxlL1Byb2R1Y3QvMzAy?utm_source=smartbar&utm_medium=article&utm_campaign=free-book)



MongoDB Compass connected to mLab

Note that I called my database `sp-node-article`. You can call yours what you like.

Initialize the Application

With everything set up correctly, the first thing we need to do is initialize our new project. To do this, create a folder named `demo-node-app`, enter that directory and type the following in a terminal:

```
npm init -y
```

This will create and auto-populate a `package.json` file in the project root. We can use this file to specify our dependencies and to create various npm scripts (<https://www.sitepoint.com/guide-to-npm-as-a-build-tool/>), which will aid our development workflow.

Install Express

Express is a lightweight web application framework for Node.js, which provides us with a robust set of features for writing web apps. These features include such things as route handling, template engine integration and a middleware framework, which allows us to perform additional tasks on request and response objects. There is nothing you can do in Express that you couldn't do in plain Node.js, but using Express means we don't have to re-invent the wheel and reduces boilerplate.



So let's install [Express](http://expressjs.com/) (<http://expressjs.com/>). To do this, run the following in your terminal:

FREE Book: JavaScript Best Practice

Check it out! `npm install --save express` (<https://www.sitepoint.com/>).



(https://www.sitepoint.com/premium/products/Z2lkOi8vbGVhcm5hYmxlL1Byb2R1Y3QvMzAy?utm_source=smartbar&utm_medium=article&utm_campaign=free-book)

By passing the `--save` option to the `npm install` command, Express will be added to the **dependencies** section of the `package.json` file. This signals to anyone else running our code that Express is a package our app needs to function properly.

Install nodemon

[nodemon](https://github.com/remy/nodemon) (<https://github.com/remy/nodemon>) is a convenience tool. It will watch the files in the directory it was started in, and if it detects any changes, it will automatically restart your Node application (meaning you don't have to). In contrast to Express, nodemon is not something the app requires to function properly (it just aids us with development), so install it using:

```
npm install --save-dev nodemon
```

This will add nodemon to the **dev-dependencies** section of the `package.json` file.

Create Some Initial Files

We're almost through with the setup. All we need to do now is create a couple of initial files before kicking off the app.

In the **demo-node-app** folder create an **app.js** file and a **start.js** file. Also create a **routes** folder, with an **index.js** file inside. After you're done, things should look like this:

```
.
├─ app.js
├─ node_modules
│   └─ ...
├─ package.json
├─ routes
│   └─ index.js
└─ start.js
```

Now, let's add some code to those files.

In `app.js`:



```
const express = require('express');
const routes = require('./routes/index');
```

FREE Book: JavaScript Best Practice

Check it out!

(<https://www.sitepoint.com/>)

(https://www.sitepoint.com/premium/products/Z2lkOi8vbGVhcm5hYmxlL1Byb2R1Y3QvMzAy?utm_source=smartbar&utm_medium=article&utm_campaign=free-book)

```
module.exports = app;
```

Here, we're importing both the **express** module and (the export value of) our routes file into the application. The **require** function we're using to do this is a built-in Node function which imports an object from another file or module. If you'd like a refresher on importing and exporting modules, read [Understanding module.exports and exports in Node.js](https://www.sitepoint.com/understanding-module-exports-exports-node-js/) (<https://www.sitepoint.com/understanding-module-exports-exports-node-js/>).

After that, we're creating a new Express app using the **express** (<http://expressjs.com/en/api.html#express>) function and assigning it to an **app** variable. We then tell the app that, whenever it receives a request from forward slash anything, it should use the routes file.

Finally, we export our app variable so that it can be imported and used in other files.

In **start.js**:

```
const app = require('./app');

const server = app.listen(3000, () => {
  console.log(`Express is running on port ${server.address().port}`);
});
```

Here we're importing the Express app we created in **app.js** (note that we can leave the **.js** off the file name in the **require** statement). We then tell our app to listen on port 3000 for incoming connections and output a message to the terminal to indicate that the server is running.

And in **routes/index.js**:



```
const express = require('express');
```

FREE Book: JavaScript Best Practice

```
const router = express.Router();
```

Check it out!

(<https://www.sitepoint.com/>)



(https://www.sitepoint.com/premium/products/Z2lkOi8vbGVhcm5hYmxlL1Byb2R1Y3QvMzAy?utm_source=smartbar&utm_medium=article&utm_campaign=free-book)

```
});
```

```
module.exports = router;
```

Here, we're importing Express into our routes file and then grabbing the router from it. We then use the router to respond to any requests to the root URL (in this case `http://localhost:3000`) with an "It works!" message.

Kick off the App

Finally, let's add an npm script to make nodemon start watching our app. Change the `scripts` section of the `package.json` file to look like this:

```
"scripts": {  
  "watch": "nodemon ./start.js"  
},
```

The `scripts` property of the `package.json` file is extremely useful, as it lets you specify arbitrary scripts to run in different scenarios. This means that you don't have to repeatedly type out long-winded commands with a difficult-to-remember syntax. If you'd like to find out more about what npm scripts can do, read [Give Grunt the Boot! A Guide to Using npm as a Build Tool](https://www.sitepoint.com/guide-to-npm-as-a-build-tool/) (<https://www.sitepoint.com/guide-to-npm-as-a-build-tool/>).

Now, type `npm run watch` from the terminal and visit <http://localhost:3000> (<http://localhost:3000>).

You should see "It works!"

Basic Templating with Pug

Returning an inline response from within the route handler is all well and good, but it's not very extensible, and this is where templating engines come in. As the [Express docs](http://expressjs.com/en/guide/using-template-engines.html) (<http://expressjs.com/en/guide/using-template-engines.html>), state:


A template engine enables you to use static template files in your application. At runtime, the template engine replaces variables in a template file with actual values, and transforms the template into an HTML file sent to the client.

In practice, this means we can define template files and tell our routes to use them instead of writing everything inline. Let's do that now.



Create a folder named **views** and in that folder a file named **form.pug**. Add the following code to this new file:

FREE Book: JavaScript Best Practice

Check it out!  https://www.sitepoint.com/premium/products/Z2lkOi8vbGVhcm5hYmxlL1Byb2R1Y3QvMzAy?utm_source=smartbar&utm_medium=article&utm_campaign=free-book

```
form(action="." method="POST")
  label(for="name") Name:
  input(
    type="text"
    id="name"
    name="name"
  )

  label(for="email") Email:
  input(
    type="email"
    id="email"
    name="email"
  )

  input(type="submit" value="Submit")
```

As you can deduce from the file ending, we'll be using the [pug templating engine](https://pugjs.org) (<https://pugjs.org>) in our app. Pug (formerly known as Jade) comes with its own indentation-sensitive syntax for writing dynamic and reusable HTML. Hopefully the above example is easy to follow, but if you have any difficulties understanding what it does, just wait until we view this in a browser, then inspect the page source to see the markup it produces.

If you'd like a refresher as to what JavaScript templates and/or templating engines are, and when you should use them, read [An Overview of JavaScript Templating Engines](https://www.sitepoint.com/overview-javascript-templating-engines/) (<https://www.sitepoint.com/overview-javascript-templating-engines/>).

Install Pug and Integrate It into the Express App

Next, we'll need to install pug, saving it as a dependency:

```
npm i --save pug
```

Then configure **app.js** to use Pug as a layout engine and to look for templates inside the **views** folder:



FREE Book: JavaScript Best Practice

Check it out!

(<https://www.sitepoint.com/>)



(https://www.sitepoint.com/premium/products/Z2lkOi8vbGVhcm5hYmxlL1Byb2R1Y3QvMzAy?utm_source=smartbar&utm_medium=article&utm_campaign=free-book)

```
const express = require('express');
const path = require('path');
const routes = require('./routes/index');

app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'pug');

app.use('/', routes);

module.exports = app;
```

You'll notice that we're also requiring Node's native [Path module \(https://nodejs.org/api/path.html\)](https://nodejs.org/api/path.html), which provides utilities for working with file and directory paths. This module allows us to build the path to our **views** folder using its [join method \(https://nodejs.org/docs/latest/api/path.html#path_path_join_paths\)](https://nodejs.org/docs/latest/api/path.html#path_path_join_paths) and [__dirname \(https://nodejs.org/docs/latest/api/modules.html#modules_dirname\)](https://nodejs.org/docs/latest/api/modules.html#modules_dirname) (which returns the directory in which the currently executing script resides).

Alter the Route to Use Our Template

Finally, we need to tell our route to use our new template. In `routes/index.js`:

```
router.get('/', (req, res) => {
  res.render('form');
});
```

This uses the [render method \(http://expressjs.com/en/4x/api.html#res.render\)](http://expressjs.com/en/4x/api.html#res.render) on Express's response object to send the rendered view to the client.

So let's see if it worked. As we're using nodemon to watch our app for changes, you should simply be able to refresh your browser and see our brutalist masterpiece.

Define a Layout File for Pug

If you open your browser and inspect the page source, you'll see that Express only sent the HTML for the form: our page is missing a doc-type declaration, as well as a head and body section. Let's fix that by creating a master layout for all our templates to use.

To do this, create a `layout.pug` file in the **views** folder and add the following code:



```
doctype html
```

```
html
```

```
head
```

```
title= ${title}
```

```
body
```

```
h1 My Amazing App
```

```
block content
```

FREE Book: JavaScript Best Practice

Check it out!

(https://www.sitepoint.com/premium/products/Z2lkOi8vbGVhcm5hYmxlL1Byb2R1Y3QvMzAy?utm_source=smartbar&utm_medium=article&utm_campaign=free-book)



The first thing to notice here is the line starting `title=`. Appending an equals sign to an attribute is one of the methods that Pug uses for interpolation. You can read more about it [here \(https://pugjs.org/language/interpolation.html\)](https://pugjs.org/language/interpolation.html). We'll use this to pass the title dynamically to each template.

The second thing to notice is the line that starts with the `block` keyword. In a template, a block is simply a "block" of Pug that a child template may replace. We'll see how to use it shortly, but if you're keen to find out more, read [this page on the Pug website \(https://pugjs.org/language/inheritance.html\)](https://pugjs.org/language/inheritance.html).

Use the Layout File from the Child Template

All that remains to do is to inform our `form.pug` template that it should use the layout file. To do this, alter `views/form.pug`, like so:

```
extends layout
```

```
block content
```

```
form(action="." method="POST")
```

```
label(for="name") Name:
```

```
input(
```

```
  type="text"
```

```
  id="name"
```

```
  name="name"
```

```
)
```

```
label(for="email") Email:
```

```
input(
```

```
  type="email"
```

```
  id="email"
```

```
  name="email"
```

```
)
```

```
input(type="submit" value="Submit")
```

And in `routes/index.js`, we need to pass in an appropriate title for the template to display:



```
router.get('/', (req, res) => {
  res.render('form', { title: 'Registration form' });
});
```

FREE Book: JavaScript Best Practice

Check it out!

(<https://www.sitepoint.com/>)



(https://www.sitepoint.com/premium/products/Z2lkOi8vbGVhcm5hYmxlL1Byb2R1Y3QvMzAy?utm_source=smartbar&utm_medium=article&utm_campaign=free-book)

Now if you refresh the page and inspect the source, things should look a lot better.

Dealing with Forms in Express

Currently, if you hit our form's *Submit* button, you'll be redirected to a page with a message: "Cannot POST /". This is because when submitted, our form POSTs its contents back to / and we haven't defined a route to handle that yet.

Let's do that now. Add the following to `routes/index.js`:

```
router.post('/', (req, res) => {
  res.render('form', { title: 'Registration form' });
});
```

This is the same as our GET route, except for the fact that we're using `router.post` to respond to a different HTTP verb.

Now when we submit the form, the error message will be gone and the form should just re-render.

Handle Form Input

The next task is to retrieve whatever data the user has submitted via the form. To do this, we'll need to install a package named `body-parser` (<https://www.npmjs.com/package/body-parser>), which will make the form data available on the request body:

```
npm install --save body-parser
```

We'll also need to tell our app to use this package, so add the following to `app.js`:

```
const bodyParser = require('body-parser');
...
app.use(bodyParser.urlencoded({ extended: true }));
app.use('/', routes);

module.exports = app;
```



Note that there are various ways to format the data you POST to the server, and using body-parser's [urlencoded](#)

(<https://www.npmjs.com/package/body-parser#body-parser-options>), the node framework can handle data sent as application/x-www-form-urlencoded.

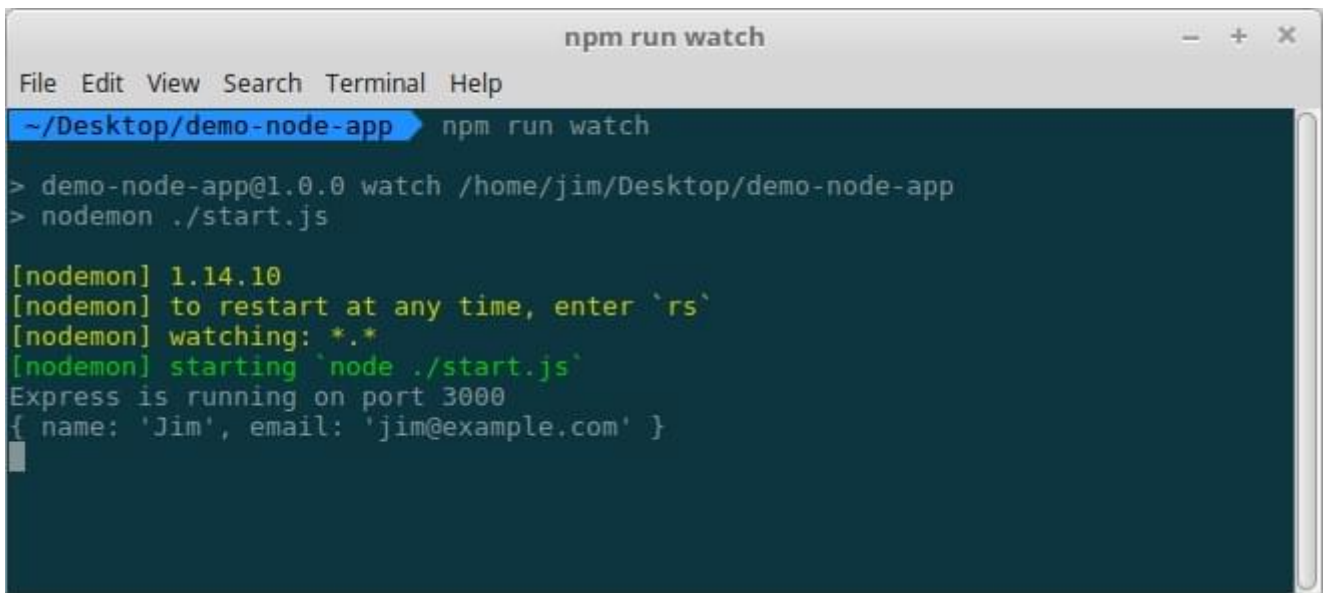
Check it out! (<https://www.sitepoint.com/>).

(https://www.sitepoint.com/premium/products/Z2lkOi8vbGVhcm5hYmxlL1Byb2R1Y3QvMzAy?utm_source=smartbar&utm_medium=article&utm_campaign=free-book)

```
router.post('/', (req, res) => {
  console.log(req.body);
  res.render('form', { title: 'Registration form' });
});
```

Now when you submit the form, you should see something along the lines of:

```
{name: 'Jim', email: 'jim@example.com'}
```



```
npm run watch
File Edit View Search Terminal Help
~/Desktop/demo-node-app npm run watch
> demo-node-app@1.0.0 watch /home/jim/Desktop/demo-node-app
> nodemon ./start.js

[nodemon] 1.14.10
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node ./start.js`
Express is running on port 3000
{ name: 'Jim', email: 'jim@example.com' }
```

Form output logged to terminal

A Note about Request and Response Objects

By now you've hopefully noticed the pattern we're using to handle routes in Express.

```
router.METHOD(route, (req, res) => {
  // callback function
});
```



The callback function is executed whenever somebody visits a URL that matches the route it specifies. The callback receives a `req` and `res` parameter, where `req` is an object full of information about the request (such as the URL or query parameters) and `res` is an object full of methods for sending data back to the user. There's also an optional `next` parameter which is used if you don't actually want to send any data back, or if you want to pass the request off for something else to handle.

Check it out!

(<https://www.sitepoint.com/>)

(https://www.sitepoint.com/premium/products/Z2lkOi8vbGVhcm5hYmxlL1Byb2R1Y3QvMzAy?utm_source=smartbar&utm_medium=article&utm_campaign=free-book)

Without getting too deep into the weeds, this is a concept known as middleware (specifically, router-level middleware) which is very important in Express. If you're interested in finding out more about how Express uses middleware, I recommend you read the [Express docs](http://expressjs.com/en/guide/using-middleware.html) (<http://expressjs.com/en/guide/using-middleware.html>).

Validating Form Input

Now let's check that the user has filled out both our fields. We can do this using [express-validator module](https://www.npmjs.com/package/express-validator) (<https://www.npmjs.com/package/express-validator>), a middleware that provides a number of useful methods for the sanitization and validation of user input.

You can install it like so:

```
npm install express-validator --save
```

And require the functions we'll need in `routes/index.js`:

```
const { body, validationResult } = require('express-validator/check');
```

We can include it in our route handler like so:

```
router.post('/',
[
  body('name')
    .isLength({ min: 1 })
    .withMessage('Please enter a name'),
  body('email')
    .isLength({ min: 1 })
    .withMessage('Please enter an email'),
],
(req, res) => {
  ...
})
```



As you can see, we're using the `body` method (<https://github.com/ctavan/express-validator#bodyfields-message>) to validate two properties on `req.body` - `email`, `password` and `confirm`. Our basic test is sufficient to just check if these properties exist (i.e. that they have a length greater than one), but `express-validator` offers a whole host of other methods that you can read

Check it out!

(<https://www.sitepoint.com/>)
about it on the project's home page (<https://github.com/ctavan/express-validator>).

(https://www.sitepoint.com/premium/products/Z2lkOi8vbGVhcm5hYmxlL1Byb2R1Y3QvMzAy?utm_source=smartbar&utm_medium=article&utm_campaign=free-book)

In a second step, we can call the `validationResult` (<https://github.com/ctavan/express-validator#validationresultreq>) method to see if validation passed or failed. If no errors are present, we can go ahead and render out a "Thanks for registering" message. Otherwise, we'll need to pass these errors back to our template, so as to inform the user that something's wrong.

And if validation fails, we'll also need to pass `req.body` back to the template, so that any valid form inputs aren't reset:

```
router.post(
  '/',
  [
    ...
  ],
  (req, res) => {
    const errors = validationResult(req);

    if (errors.isEmpty()) {
      res.send('Thank you for your registration!');
    } else {
      res.render('form', {
        title: 'Registration form',
        errors: errors.array(),
        data: req.body,
      });
    }
  }
);
```

Now we have to make a couple of changes to our `form.pug` template. We firstly need to check for an `errors` property, and if it's present, loop over any errors and display them in a list:

```
extends layout

block content
  if errors
    ul
      for error in errors
        li= error.msg
    ...
```



If the `li=` looks weird, remember that pug does interpolation by following the tag name with an equals sign.

FREE Book: JavaScript Best Practice

Finally, we need to check if a `data` attribute exists, and if so, use it to set the values of the respective fields. If it doesn't exist,

Check it out!

(<https://www.sitepoint.com/>)

we'll initialize it to an empty object, so that the form will still render correctly when you load it for the first time. We can do this with `JavaScript`, denoted in Pug by a minus sign.

(https://www.sitepoint.com/premium/products/Z2lkOi8vbGVhcm5hYmxlL1Byb2R1Y3QvMzAy?utm_source=smartbar&utm_medium=article&utm_campaign=free-book)

```
-data = data || {}
```

We then reference that attribute to set the field's value:

```
input(  
  type="text"  
  id="name"  
  name="name"  
  value=data.name  
)
```

That gives us the following:



extends layout

FREE Book: JavaScript Best Practice

block content

Check it out!

(<https://www.sitepoint.com/>)

(https://www.sitepoint.com/premium/products/Z2lkOi8vbGVhcm5hYmxlL1Byb2R1Y3QvMzAy?utm_source=smartbar&utm_medium=article&utm_campaign=free-book)



ul

for error in errors

li= error.msg

form(action="." method="POST")

label(for="name") Name:

input(

type="text"

id="name"

name="name"

value=data.name

)

label(for="email") Email:

input(

type="email"

id="email"

name="email"

value=data.email

)

input(type="submit" value="Submit")

Now, when you submit a successful registration, you should see a thank you message, and when you submit the form without filling out both field, the template should be re-rendered with an error message.

Interact with a Database

We now want to hook our form up to our database, so that we can save whatever data the user enters. If you're running Mongo locally, don't forget to start the server with the command `mongod`.

Specify Connection Details

We'll need somewhere to specify our database connection details. For this, we'll use a configuration file (which *should not* be checked into version control) and the [dotenv package](https://www.npmjs.com/package/dotenv) (<https://www.npmjs.com/package/dotenv>). Dotenv will load our connection details from the configuration file into Node's [process.env](https://nodejs.org/docs/latest/api/process.html#process_process_env).

(https://nodejs.org/docs/latest/api/process.html#process_process_env).

Install it like so:



```
npm install dotenv --save
```

FREE Book: JavaScript Best Practice

Check it out!

(<https://www.sitepoint.com/>).

Any require at the top of start.js is:
(https://www.sitepoint.com/premium/products/Z2lkOi8vbGVhcm5hYmxlL1Byb2R1Y3QvMzAy?utm_source=smartbar&utm_medium=article&utm_campaign=free-book)



```
require('dotenv').config();
```

Next, create a file named `.env` in the project root (note that starting a filename with a dot may cause it to be hidden on certain operating systems) and enter your Mongo connection details on the first line.

If you're running Mongo locally:

```
DATABASE=mongodb://localhost:27017/node-demo-application
```

If you're using mLab:

```
mongodb://<dbuser>:<dbpassword>@ds251827.mlab.com:51827/<dbname>
```

Note that local installations of MongoDB don't have a default user or password. This is definitely something you'll want to change in production, as it's otherwise a security risk.

Connect to the Database

To establish the connection to the database and to perform operations on it, we'll be using [Mongoose](https://www.npmjs.com/package/mongoose) (<https://www.npmjs.com/package/mongoose>). Mongoose is an [ORM](https://en.wikipedia.org/wiki/Object-relational_mapping) (https://en.wikipedia.org/wiki/Object-relational_mapping), for MongoDB, and as you can read on the [project's home page](http://mongoosejs.com) (<http://mongoosejs.com>):

Mongoose provides a straight-forward, schema-based solution to model your application data. It includes built-in type casting, validation, query building, business logic hooks and more, out of the box.

What this means in real terms is that it creates various abstractions over Mongo, which make interacting with our database easier and reduce the amount of boilerplate we have to write. If you'd like to find out more about how Mongo works under the hood, be sure to read our [Introduction to MongoDB](https://www.sitepoint.com/introduction-to-mongodb/) (<https://www.sitepoint.com/introduction-to-mongodb/>).

To install Mongoose:

```
npm install --save mongoose
```



Then, require it in `start.js`:

FREE Book: JavaScript Best Practice

Check it out!

`const mongoose = require('mongoose');`
(<https://www.sitepoint.com/>)



(https://www.sitepoint.com/premium/products/Z2lkOi8vbGVhcm5hYmxlL1Byb2R1Y3QvMzAy?utm_source=smartbar&utm_medium=article&utm_campaign=free-book)

The connection is made like so:

```
mongoose.connect(process.env.DATABASE, { useMongoClient: true });
mongoose.Promise = global.Promise;
mongoose.connection
  .on('connected', () => {
    console.log(`Mongoose connection open on ${process.env.DATABASE}`);
  })
  .on('error', (err) => {
    console.log(`Connection error: ${err.message}`);
  });
```

Notice how we use the `DATABASE` variable we declared in the `.env` file to specify the database URL. We're also telling Mongo to use ES6 Promises (these are necessary, as database interactions are asynchronous), as its own default promise library is deprecated.

This is what `start.js` should now look like:

```
require('dotenv').config();
const mongoose = require('mongoose');

mongoose.connect(process.env.DATABASE, { useMongoClient: true });
mongoose.Promise = global.Promise;
mongoose.connection
  .on('connected', () => {
    console.log(`Mongoose connection open on ${process.env.DATABASE}`);
  })
  .on('error', (err) => {
    console.log(`Connection error: ${err.message}`);
  });

const app = require('./app');
const server = app.listen(3000, () => {
  console.log(`Express is running on port ${server.address().port}`);
});
```

When you save the file, nodemon will restart the app and, if all's gone well, you should see something along the lines of:



FREE Book: JavaScript Best Practice

Check it out!

(<https://www.sitepoint.com/>).

Define a Mongoose Schema

(https://www.sitepoint.com/premium/products/Z2lkOi8vbGVhcm5hYmxlL1Byb2R1Y3QvMzAy?utm_source=smartbar&utm_medium=article&utm_campaign=free-book)



Mongoose can be used as a loose database, meaning it's not necessary to describe what data will look like ahead of time.

However, out of the box it runs in strict mode, which means it'll only allow you to save data it knows about beforehand. As we'll

be using strict mode, we'll need to define the shape our data using a **schema** (<http://mongoosejs.com/docs/guide.html>).

Schemas allow you to define the fields stored in each document along with their type, validation requirements and default values.

To this end, create a **models** folder in the project root, and within that folder, a new file named **Registration.js**.

Add the following code to **Registration.js**:

```
const mongoose = require('mongoose');

const registrationSchema = new mongoose.Schema({
  name: {
    type: String,
    trim: true,
  },
  email: {
    type: String,
    trim: true,
  },
});


module.exports = mongoose.model('Registration', registrationSchema);
```

Here, we're just defining a type (as we already have validation in place) and are making use of the **trim helper method** (http://mongoosejs.com/docs/api.html#schema_string_SchemaString-trim), to remove any superfluous white space from user input. We then **compile a model** (<http://mongoosejs.com/docs/models.html>), from the Schema definition, and export it for use elsewhere in our app.

The final piece of boilerplate is to require the model in **start.js**:



FREE Book: JavaScript Best Practice

Check it out!  https://www.sitepoint.com/premium/products/Z2lkOi8vbGVhcm5hYmxlL1Byb2R1Y3QvMzAy?utm_source=smarthbar&utm_medium=article&utm_campaign=free-book

```
...
require('./models/Registration');
const app = require('./app');
const server = app.listen(3000);

console.log(`Express is running on port ${server.address().port}`);
});
```

Save Data to the Database

Now we're ready to save user data to our database. Let's begin by requiring Mongoose and importing our model into our `routes/index.js` file:

```
const express = require('express');
const mongoose = require('mongoose');
const { body, validationResult } = require('express-validator/check');

const router = express.Router();
const Registration = mongoose.model('Registration');
...
```

Now, when the user posts data to the server, if validation passes we can go ahead and create a new **Registration** object and attempt to save it. As the database operation is an asynchronous operation which returns a Promise, we can chain a `.then()` onto the end of it to deal with a successful insert and a `.catch()` to deal with any errors:

```
if (errors.isEmpty()) {
  const registration = new Registration(req.body);
  registration.save()
    .then(() => { res.send('Thank you for your registration!'); })
    .catch(() => { res.send('Sorry! Something went wrong.');
```

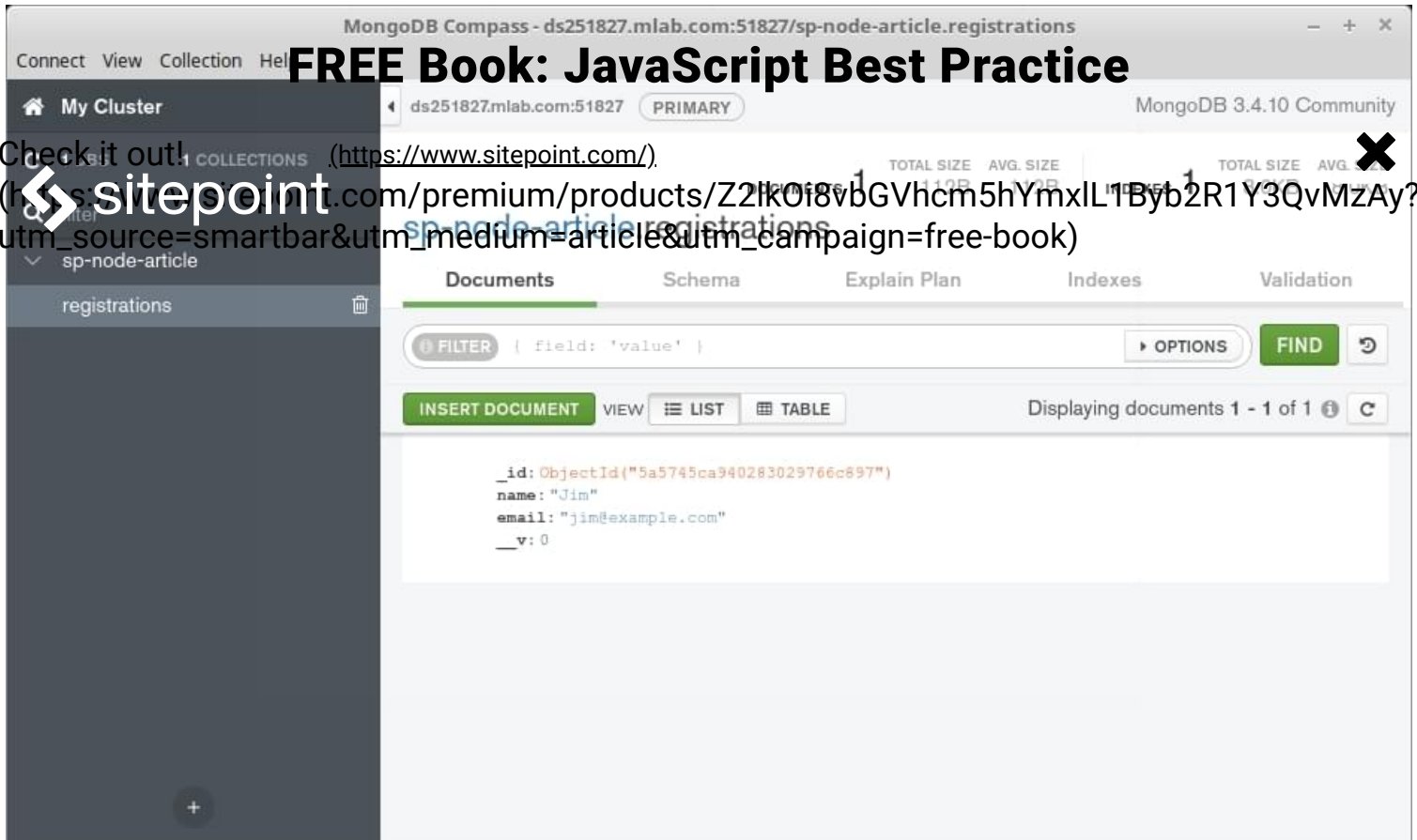
Now, if you enter your details into the registration form, they should be persisted to the database. You can check this using Compass (making sure to hit the refresh button in the top left if it's still running).



FREE Book: JavaScript Best Practice

Check it out! (https://www.sitepoint.com/)

(https://www.sitepoint.com/premium/products/Z2lkOi8vbGVhcm5hYmxlL1Byb2R1Y3QvMzAy?utm_source=smartbar&utm_medium=article&utm_campaign=free-book)



Using Compass to check that our data was saved to MongoDB

Retrieve Data from the Database

To round the app off, let's create a final route, which lists out all of our registrations. Hopefully you should have a reasonable idea of the process by now.

Add a new route to `routes/index.js`, as follows:

```
router.get('/registrations', (req, res) => {
  res.render('index', { title: 'Listing registrations' });
});
```

This means that we'll also need a corresponding view template (`views/index.pug`):

```
extends layout

block content
  p No registrations yet :(
```



Now when you visit <http://localhost:3000/registrations> (<http://localhost:3000/registrations>), you should see a message telling you that there aren't any registrations.

FREE Book: JavaScript Best Practice

Check it out!

(<https://www.sitepoint.com/>)

Let's fix that by retrieving our registrations from the database and passing them to the view. We'll still display the "No registrations yet" message, but only if there really aren't any.

(https://www.sitepoint.com/premium/products/Z2lkOi8vbGVhcm5hYmxlL1Byb2R1Y3QvMzAy?utm_source=smartbar&utm_medium=article&utm_campaign=free-book)

In `routes/index.js`:

```
router.get('/registrations', (req, res) => {
  Registration.find()
    .then((registrations) => {
      res.render('index', { title: 'Listing registrations', registrations });
    })
    .catch(() => { res.send('Sorry! Something went wrong.');
```

Here, we're using Mongo's [Collection#find method](#)

(<https://docs.mongodb.com/manual/reference/method/db.collection.find/>), which, if invoked without parameters, will return all of the records in the collection. Because the database lookup is asynchronous, we're waiting for it to complete before rendering the view. If any records were returned, these will be passed to the view template in the `registrations` property. If no records were returned `registrations` will be an empty array.

In `views/index.pug`, we can then check the length of whatever we're handed and either loop over it and output the records to the screen, or display a "No registrations" message:

```
extends layout


block content

  if registrations.length
    table
      tr
        th Name
        th Email
      each registration in registrations
        tr
          td= registration.name
          td= registration.email
  else
    p No registrations yet :(
```

The final feature we'll add to our app is [HTTP authentication \(https://en.wikipedia.org/wiki/Basic_access_authentication\)](https://en.wikipedia.org/wiki/Basic_access_authentication), locking down the list of successful registrations from spilling out.

FREE Book: JavaScript Best Practice

Check it out!

To do this, we'll use the [http-auth module \(https://www.npmjs.com/package/http-auth\)](https://www.npmjs.com/package/http-auth), which we can install using: 
(https://www.sitepoint.com/premium/products/Z2lkOi8vbGVhcm5hYmxlL1Byb2R1Y3QvMzAy?utm_source=smartbar&utm_medium=article&utm_campaign=free-book)

```
npm install --save http-auth
```

Next we need to require it in `routes/index.js`, along with the Path module we met earlier:

```
const path = require('path');  
const auth = require('http-auth');
```

Next, let it know where to find the file in which we'll list the users and passwords (in this case `users.htpasswd` in the project root):

```
const basic = auth.basic({  
  file: path.join(__dirname, '../users.htpasswd'),  
});
```

Create this `users.htpasswd` file next and add a username and password separated by a colon. This can be in plain text, but the http-auth module also supports hashed passwords, so you could also run the password through a service such as [Htpasswd Generator \(http://www.htaccesstools.com/htpasswd-generator/\)](http://www.htaccesstools.com/htpasswd-generator/).

For me, the contents of `users.htpasswd` looks like this:

```
jim:$apr1$FhFmamtz$PgXfrNI95HFCuXIm30Q4V0
```

This translates to user: `jim`, password: `password`.

Finally, add it to the route you wish to protect and you're good to go:

```
router.get('/registrations', auth.connect(basic), (req, res) => {  
  ...  
});
```



Serve Static Assets in Express

FREE Book! JavaScript Best Practice

Let's give the app some polish and add some styling using [Twitter Bootstrap \(https://getbootstrap.com/\)](https://getbootstrap.com/). We can serve static files, such as images, JavaScript files and CSS files in Express using the built-in `express.static` middleware function (https://www.sitepoint.com/premium/products/Z2lkOi8vbGVhcm5hYmxlL1Byb2R1Y3QvMzAy?utm_source=smartbar&utm_medium=article&utm_campaign=free-book) (<https://expressjs.com/en/starter/static-files.html>).

Setting it up is easy. Just add the following line to `app.js`:

```
app.use(express.static('public'));
```

Now we can load files that are in the `public` directory.

Style the App with Bootstrap

Create a `public` directory in the project root, and in the `public` directory create a `css` directory. Download [the minified version of Bootstrap v4 \(https://getbootstrap.com/docs/4.0/getting-started/download/\)](https://getbootstrap.com/docs/4.0/getting-started/download/) into this directory, ensuring it's named `bootstrap.min.css`.

Next, we'll need to add some markup to our pug templates.

In `layout.pug`:

```
doctype html
html
  head
    title= `${title}`
    link(rel='stylesheet', href='/css/bootstrap.min.css')
    link(rel='stylesheet', href='/css/styles.css')

  body
    div.container.listing-reg
      h1 My Amazing App

      block content
```

Here, we're including two files from our previously created `css` folder and adding a wrapper div.

In `form.pug` we add some class names to the error messages and the form elements:



extends layout

FREE Book: JavaScript Best Practice

block content

Check it out!

(<https://www.sitepoint.com/>)

(https://www.sitepoint.com/premium/products/Z2lkOi8vbGVhcm5hYmxlL1Byb2R1Y3QvMzAy?utm_source=smartbar&utm_medium=article&utm_campaign=free-book)



```
ul.my-errors
  for error in errors
    li= error.msg

form(action="." method="POST" class="form-registration")
  label(for="name") Name:
  input(
    type="text"
    id="name"
    name="name"
    class="form-control"
    value=data.name
  )

  label(for="email") Email:
  input(
    type="email"
    id="email"
    name="email"
    class="form-control"
    value=data.email
  )

  input(
    type="submit"
    value="Submit"
    class="btn btn-lg btn-primary btn-block"
  )
```

And in `index.pug`, more of the same:



extends layout

FREE Book: JavaScript Best Practice

block content

Check it out!

(<https://www.sitepoint.com/>)



(https://www.sitepoint.com/premium/products/Z2lkOi8vbGVhcm5hYmxlL1Byb2R1Y3QvMzAy?utm_source=smartbar&utm_medium=article&utm_campaign=free-book)

```
tr
  th Name
  th Email
each registration in registrations
  tr
    td= registration.name
    td= registration.email
else
  p No registrations yet :(
```

Finally, create a file called **styles.css** in the **css** folder and add the following:



FREE Book: JavaScript Best Practice

Check it out!

(<https://www.sitepoint.com/>)



(https://www.sitepoint.com/premium/products/Z2lkOi8vbGVhcm5hYmxlL1Byb2R1Y3QvMzAy?utm_source=smartbar&utm_medium=article&utm_campaign=free-book)

```
body {
  padding: 40px 10px;
  background-color: #eee;
}

/* css for registration form and errors*/
.form-registration {
  max-width: 330px;
  padding: 15px;
  margin: 0 auto;
}
.form-registration {
  display: flex;
  flex-wrap: wrap;
}
.form-registration input {
  width: 100%;
  margin: 0px 0 10px;
}
.form-registration .btn {
  flex: 1 0 100%;
}
.my-errors {
  margin: 0 auto;
  padding: 0;
  list-style: none;
  color: #333;
  font-size: 1.2rem;
  display: table;
}
.my-errors li {
  margin: 0 0 1rem;
}
.my-errors li:before {
  content: "! Error : ";
  color: #f00;
  font-weight: bold;
}

/* Styles for listing table */
.listing-table {
  width: 100%;
}
.listing-table th,
.listing-table td {
  padding: 10px;
```



```
border-bottom: 1px solid #666;
}
```

FREE Book: JavaScript Best Practice

Check it out!

(https://www.sitepoint.com/premium/products/Z2lkOi8vbGVhcm5hYmxlL1Byb2R1Y3QvMzAy?utm_source=smartbar&utm_medium=article&utm_campaign=free-book)

```
background: #000;
color: #fff;
}
.listing-table td:first-child,
.listing-table th:first-child {
border-right: 1px solid #666;
}
```

Now when you refresh the page, you should see all of the Bootstrap glory!

Conclusion

I hope you've enjoyed this tutorial. While we didn't build the next Facebook, I hope that I was nonetheless able to help you get your feet wet in the world of Node-based web apps and offer you some solid takeaways for your next project in the process.

Of course it's hard to cover everything in one tutorial and there are lots of ways you could elaborate on what we've built here. For example, you could check out [our article on deploying Node apps \(https://www.sitepoint.com/how-to-deploy-node-applications-heroku-vs-now-sh/\)](https://www.sitepoint.com/how-to-deploy-node-applications-heroku-vs-now-sh/) and try your hand at launching it to [Heroku \(https://www.heroku.com/\)](https://www.heroku.com/) or [now \(https://zeit.co/now\)](https://zeit.co/now). Alternatively, you might augment the CRUD functionality with the ability to delete registrations, or even write a couple of tests to test the app's functionality.

Wherever you go from here, I'd be glad to hear any questions or comments below.



Meet the author

James Hibbard (<https://www.sitepoint.com/author/jhibbard/>)

'http

(<https://twitter.com/jchibbard>) (<https://github.com/jameshibbard>)

Currently I work for SitePoint as editor of their [JavaScript hubs \(https://www.sitepoint.com/learning-hubs/\)](https://www.sitepoint.com/learning-hubs/) and technical editor for various books (e.g. [JavaScript: Novice to Ninja \(https://www.sitepoint.com/premium/books/javascript-novice-to-ninja-2nd-edition\)](https://www.sitepoint.com/premium/books/javascript-novice-to-ninja-2nd-edition) and [Jump Start Vue.js \(https://www.sitepoint.com/premium/books/jump-start-vue-js/\)](https://www.sitepoint.com/premium/books/jump-start-vue-js/)). I also work for my local university as a network admin / web dev where I spend a fair bit of my time working on Rails apps.



How to Set Up a Vue Development Environment

Check it out! (<https://www.sitepoint.com/>).

(https://www.sitepoint.com/premium/products/Z2lkOi8vbGVhcm5hYmxlL1Byb2R1Y3QvMzAy?utm_source=smartbar&utm_medium=article&utm_campaign=free-book)

If you're going to do any serious amount of work with Vue, it'll pay dividends in the long run to invest some time in setting up your coding environment. A powerful editor and a few well-chosen tools will make you more productive and ultimately a happier developer.

In this post, I'm going to demonstrate how to configure VS Code to work with Vue. I'm going to show how to use ESLint and Prettier to lint and format your code and how to use Vue's browser tools to take a peek at what's going on under the hood in a Vue app. When you've finished reading, you'll have a working development environment set up and will be ready to start coding Vue apps like a boss.

Let's get to it!

Want to learn Vue.js from the ground up? This article is an extract from our Premium library. Get an entire collection of Vue books covering fundamentals, projects, tips and tools & more with SitePoint Premium. [Join now for just \\$9/month](https://www.sitepoint.com/premium/products/Z2lkOi8vbGVhcm5hYmxlL1Byb2R1Y3QvMzA3?utm_source=blog&utm_medium=articles) (https://www.sitepoint.com/premium/products/Z2lkOi8vbGVhcm5hYmxlL1Byb2R1Y3QvMzA3?utm_source=blog&utm_medium=articles).

Installing and Setting Up Your Editor

I said that I was going to be using [VS Code](https://code.visualstudio.com/) (<https://code.visualstudio.com/>), for this tutorial, but I'm afraid I lied. I'm actually going to be using [VSCodium](https://github.com/VSCodium/vscodium) (<https://github.com/VSCodium/vscodium>), which is an open-source fork of VS Code without the Microsoft branding, telemetry and licensing. The project is under active development and I'd encourage you to check it out.

It doesn't matter which editor you use to follow along; both are available for Linux, Mac and Windows. You can [download the latest release of VSCodium here](https://github.com/VSCodium/vscodium/releases) (<https://github.com/VSCodium/vscodium/releases>), or [download the latest release of VSCode here](https://code.visualstudio.com/Download) (<https://code.visualstudio.com/Download>), and install it in the correct way for your operating system.

Throughout the rest of this guide, for the sake of consistency, I'll refer to the editor as VS Code.

Add the Vetur Extension

When you fire up the editor, you'll notice a set of five icons in a toolbar on the left-hand side of the window. If you click the bottom of these icons (the square one), a search bar will open up that enables you to search the [VS Code Marketplace](https://marketplace.visualstudio.com/) (<https://marketplace.visualstudio.com/>). Type "vue" into the search bar and you should see dozens of extensions listed, each claiming to do something slightly different.



Type "Vetur" into the search box and select the package authored by Pine Wu. Then hit *Install*.



If you visit [the project's home page \(https://vuejs.github.io/vetur/\)](https://vuejs.github.io/vetur/), you'll see that the extension gives you a number of features:

- syntax highlighting
- snippets
- Emmet
- linting/error checking
- formatting
- auto completion
- debugging

Let's see some of these in action now.

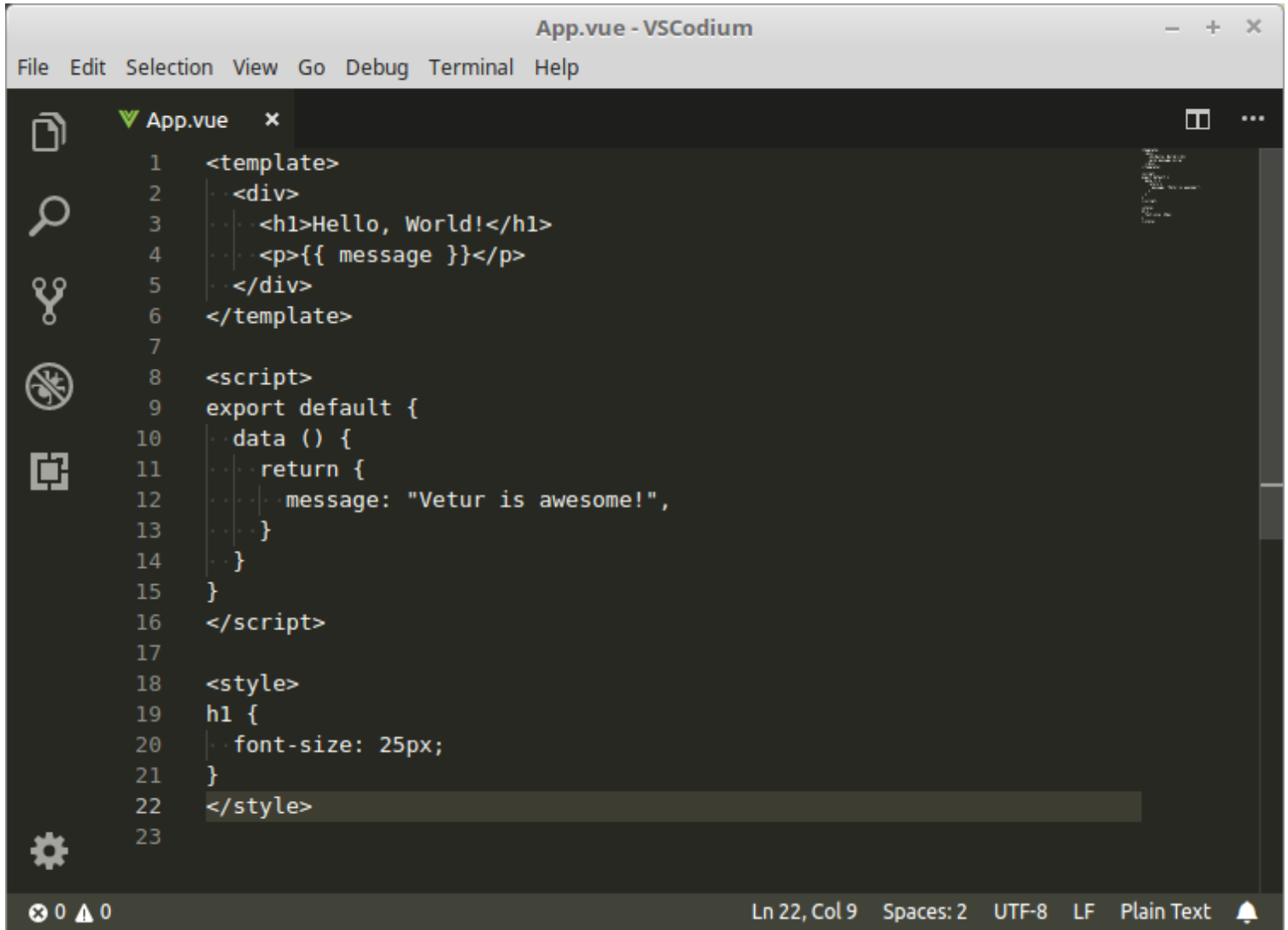


Note: many of these features only work when you have a project set up. This means you need to create a folder to contain your Vue files, open the folder using VS Code and install the Vue CLI and the VS Code extension.

FREE Book: JavaScript Best Practice

Check it out! [\(https://www.sitepoint.com/\)](https://www.sitepoint.com/)

As you get going, you'll undoubtedly want to make use of [single-file components](https://vuejs.org/v2/guide/single-file-components.html) (SFCs) to organize your code. These have a .vue ending and contain a template section, a script section and a style section. Without Vetur, this is what an SFC looks like in VS Code:



The screenshot shows a VS Code window titled 'App.vue - VSCodium'. The editor displays the code for a Vue Single-File Component (SFC) without Vetur syntax highlighting. The code is as follows:

```
1 <template>
2   <div>
3     <h1>Hello, World!</h1>
4     <p>{{ message }}</p>
5   </div>
6 </template>
7
8 <script>
9   export default {
10     data () {
11       return {
12         message: "Vetur is awesome!",
13       }
14     }
15   }
16 </script>
17
18 <style>
19   h1 {
20     font-size: 25px;
21   }
22 </style>
23
```

The status bar at the bottom indicates 'Ln 22, Col 9', 'Spaces: 2', 'UTF-8', 'LF', and 'Plain Text'.

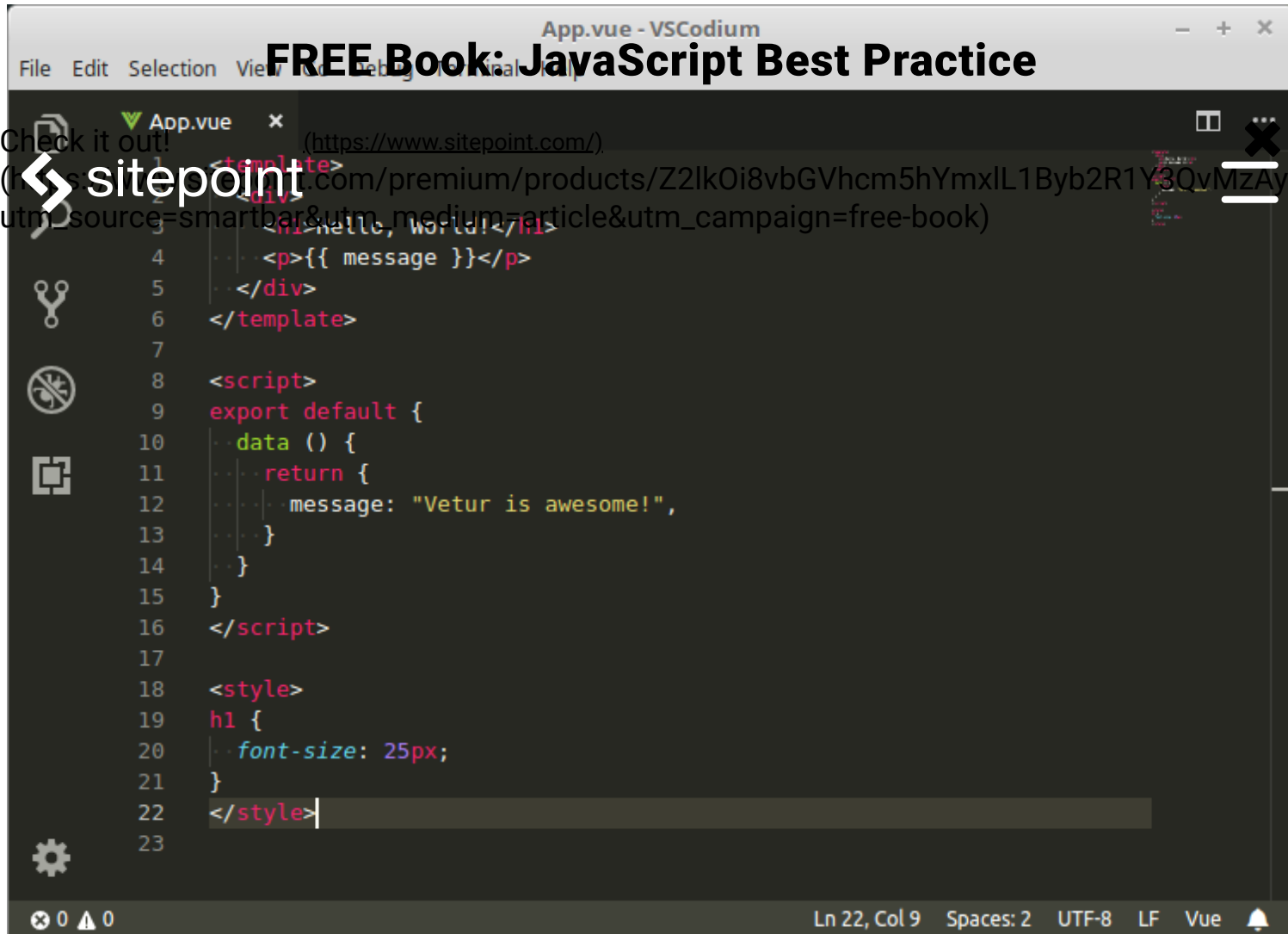
However, installing Vetur will make it look like so:



FREE Book: JavaScript Best Practice

Check it out!

sitepoint (https://www.sitepoint.com/)
(https://www.sitepoint.com/premium/products/Z2lkOi8vbGVhcm5hYmxlL1Byb2R1Y3QvMzAy?utm_source=smartbar&utm_medium=article&utm_campaign=free-book)



```
1 <template>
2   <h1>Hello, World!</h1>
3
4   <p>{{ message }}</p>
5
6 </template>
7
8 <script>
9   export default {
10     data () {
11       return {
12         message: "Vetur is awesome!",
13       }
14     }
15   }
16 </script>
17
18 <style>
19   h1 {
20     font-size: 25px;
21   }
22 </style>
23
```

Snippets

As you can read on the [VS Code website \(https://code.visualstudio.com/docs/editor/userdefinedsnippets\)](https://code.visualstudio.com/docs/editor/userdefinedsnippets), snippets are templates that make it easier to enter repeating code patterns, such as loops or conditional-statements. Vetur makes it possible for you to use these snippets in single-file components.

It also comes with some snippets of its own. For example, try typing "scaffold" (without the quotes) into an area outside a language region and it will generate all the code you need to get going with an SFC:



<template>

FREE Book: JavaScript Best Practice

</template>

Check it out!

(<https://www.sitepoint.com/>).



(https://www.sitepoint.com/premium/products/Z2lkOi8vbGVhcm5hYmxlL1Byb2R1Y3QvMzAy?utm_source=smartbar&utm_medium=article&utm_campaign=free-book)

}

</script>

<style>

</style>

Emmet

Emmet (<https://www.emmet.io/>) takes the idea of snippets to a whole new level. If you've never heard of this and spend any amount of time in a text editor, I'd recommend you head over to the Emmet website and spend some time acquainting yourself with it. It has the potential to boost your productivity greatly.

In a nutshell, Emmet allows you to expand various abbreviations into chunks of HTML or CSS. Vetur ships with this turned on by default.

Try clicking into a `<template>` region and entering the following:

```
div#header>h1.logo>a{site Name}
```

Then press `Tab`. It should be expanded to this:

```
<div id="header">
  <h1 class="logo"><a href="">sitename</a></h1>
</div>
```

Error Checking/Linting

Out of the box, Vetur offers some basic error checking. This can be handy for spotting typos in your code.



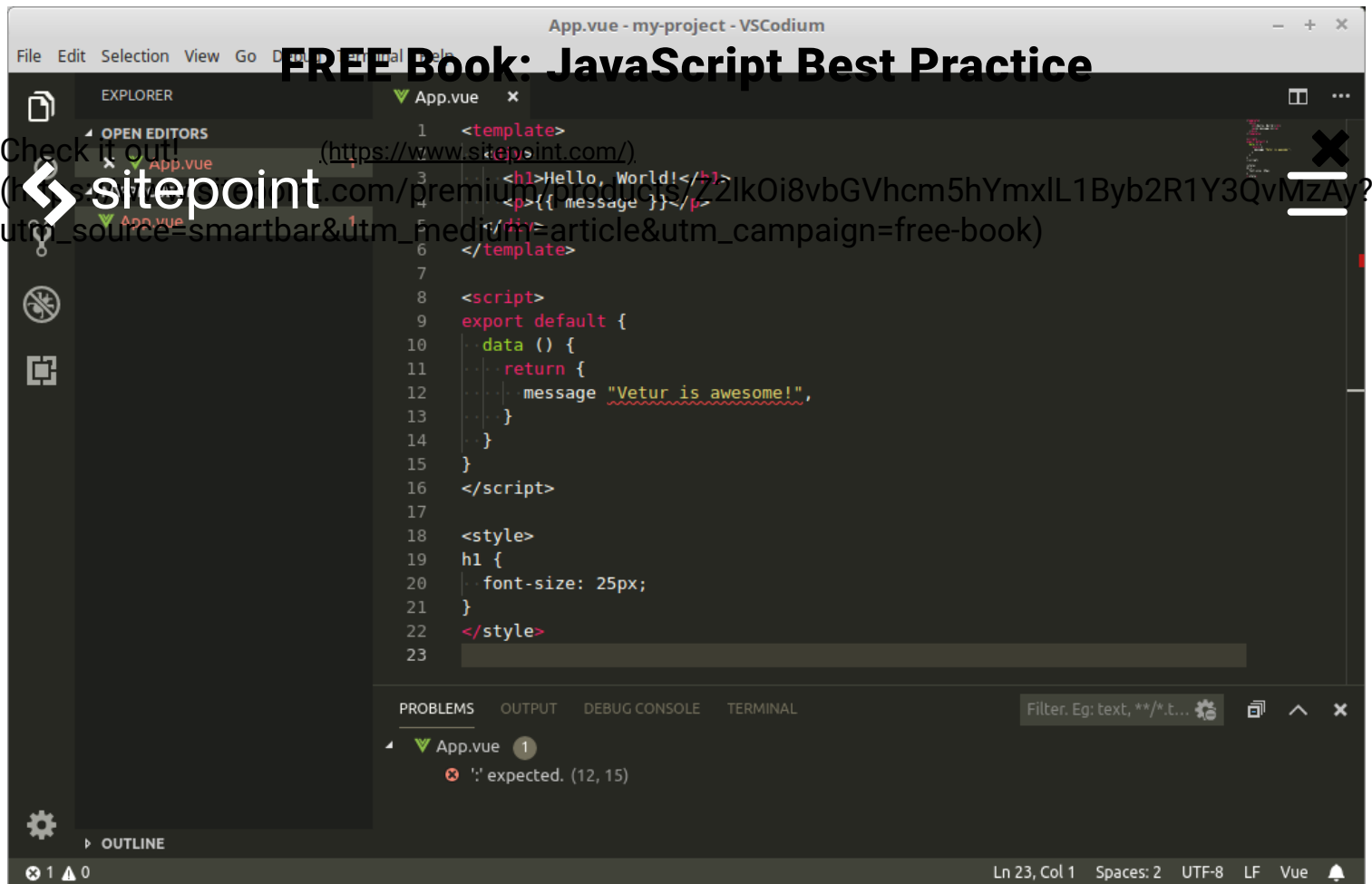
FREE Book: JavaScript Best Practice

Check it out!

sitepoint

(<https://www.sitepoint.com/>)

(https://www.sitepoint.com/premium/products/22lkOi8vbGVhcm5hYmxlL1Byb2R1Y3QvMzAy?utm_source=smartbar&utm_medium=article&utm_campaign=free-book)



In the above example, Vetur has noticed that I've forgotten the colon following the object property name and has consequently underlined it in red. Opening up the error panel (click on the small cross in the bottom left-hand corner) gives you a description of the error.

Vetur also uses [eslint-plugin-vue](https://github.com/vuejs/eslint-plugin-vue) (<https://github.com/vuejs/eslint-plugin-vue>) to lint templates. We'll find out more about why linting your code is a good idea in the next section, but for now, let's just see it in action.

Let's add an `items` key to our `data` property:

```
export default {
  data () {
    return {
      message: "Vetur is awesome!",
      items: [
        { message: 'Foo' },
        { message: 'Bar' }
      ]
    }
  }
}
```



Then add some code to output it in our template:

FREE Book: JavaScript Best Practice

Check it out!

(<https://www.sitepoint.com/>).



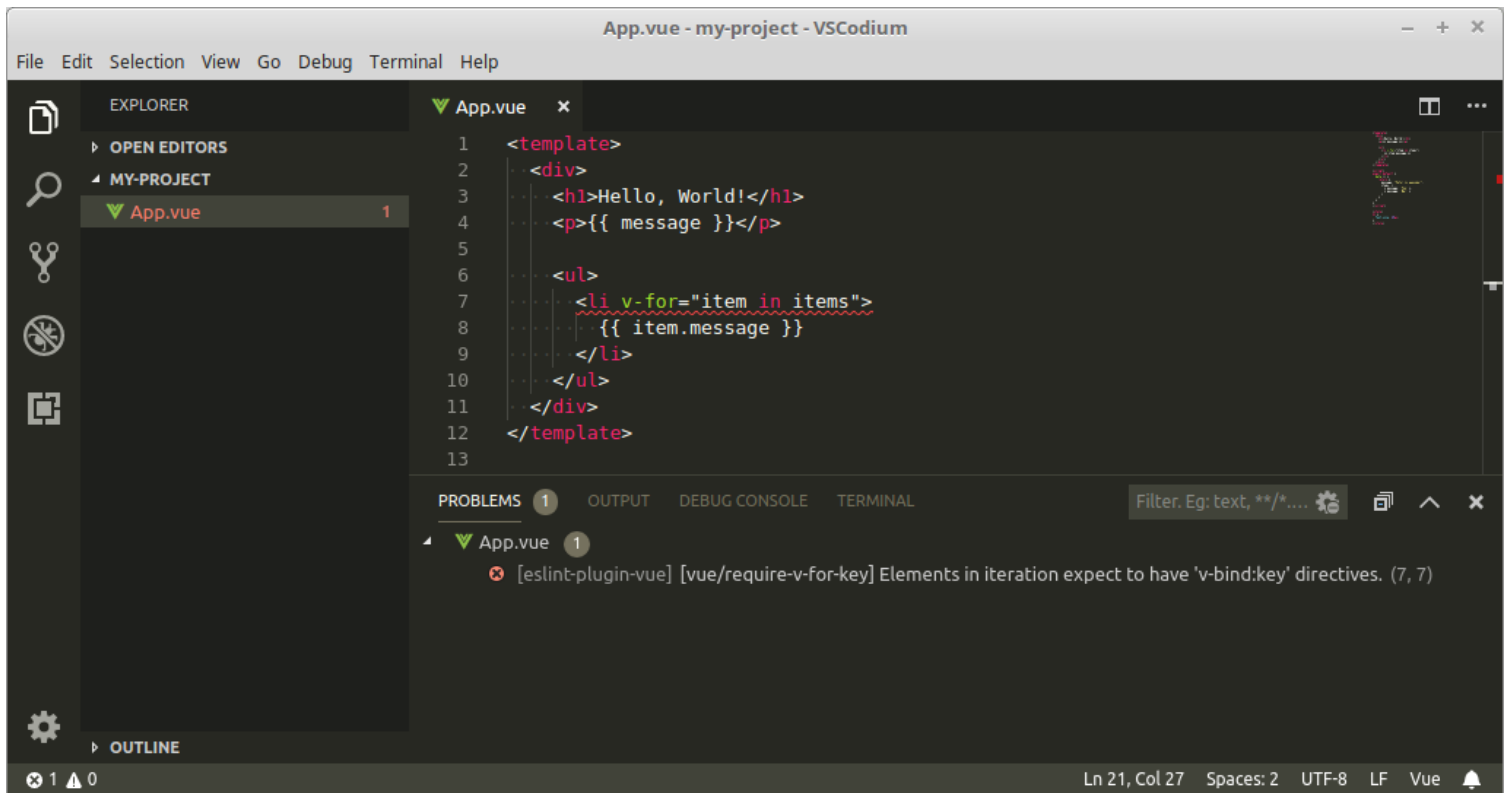
(https://www.sitepoint.com/premium/products/Z2lkOi8vbGVhcm5hYmxlL1Byb2R1Y3QvMzAy?utm_source=smartbar&utm_medium=article&utm_campaign=free-book)

```
<template>
  <div>
    <h1>Hello, World!</h1>
    <p>{{ message }}</p>

    <ul>
      <li v-for="item in items">
        {{ item.message }}
      </li>
    </ul>
  </div>
</template>
```

Straight away we'll see that `<li v-for="item in items">` is underlined in red. What gives?

Well, you can hover over the offending code, or open the error console to see what's bothering Vetur.



The error appears to be that we've forgotten to declare a key (<https://vuejs.org/v2/style-guide/#Keyed-v-for-essential>). Let's remedy that:



```
<li v-for="(item, i) in items" :key="i">
  {{ item.message }}
</li>
```

FREE Book: JavaScript Best Practice

Check it out!

(<https://www.sitepoint.com/>).



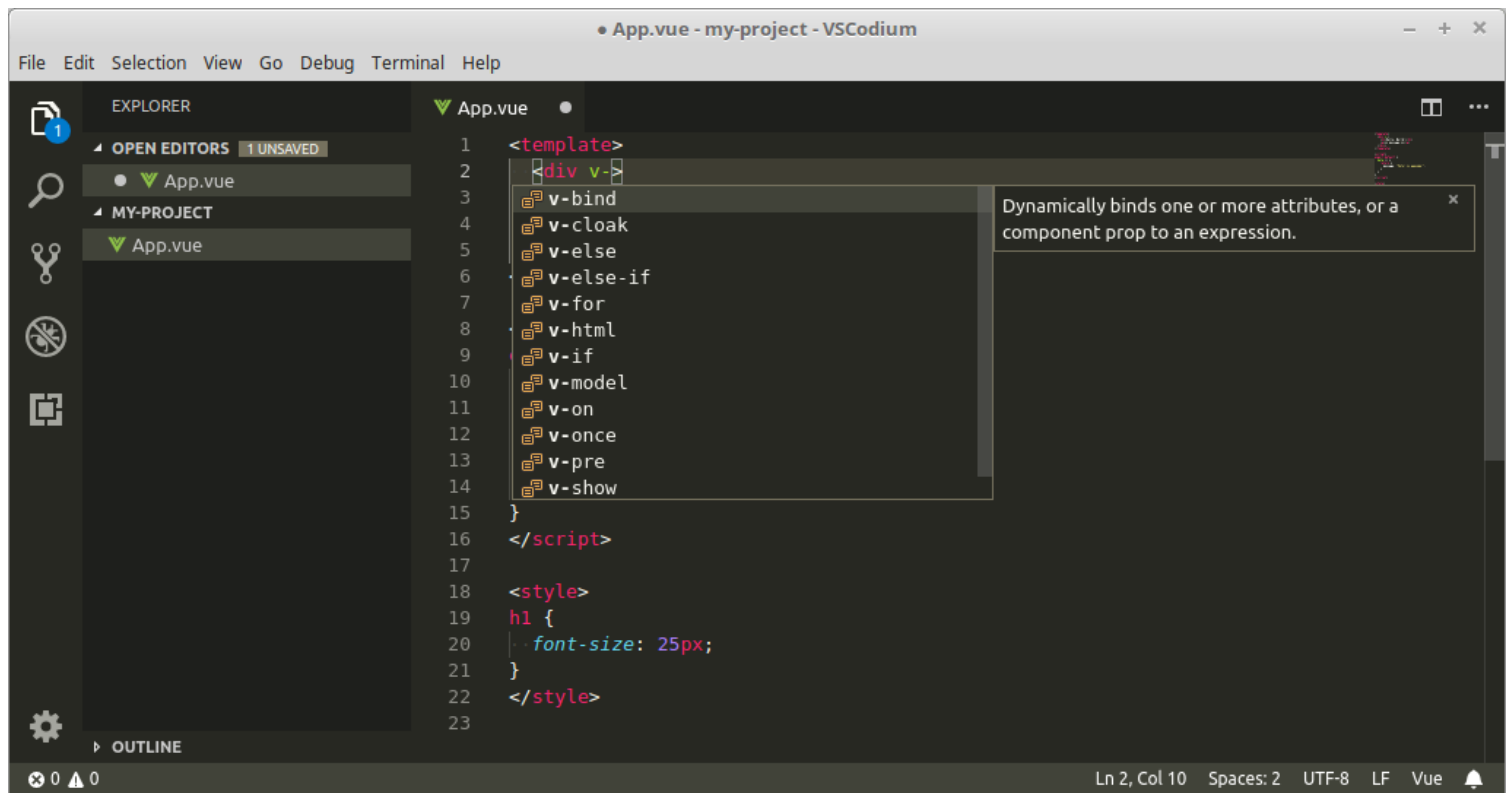
(https://www.sitepoint.com/premium/products/Z2lkOi8vbGVhcm5hYmxlL1Byb2R1Y3QvMzAy?utm_source=smartbar&utm_medium=article&utm_campaign=free-book)

And the error vanishes from our editor.

IntelliSense

IntelliSense (<https://code.visualstudio.com/docs/editor/intellisense>) is one of my favorite features in VS Code, but it's limited to a few formats that the editor can understand. Installing Vetur makes IntelliSense available in your `.vue` file, which is mighty handy.

You can try this out by clicking into the `<template>` region of a Vue component and typing "v-" on any tag (minus the quotes). You should see this:



This allows you to select from any of the listed directives, and also provides you with an explanation of what each does.

That's not all that Vetur can do, but we'll leave the extension there and turn our attention to setting up a project with Vue's CLI.

An Example Project with Vue CLI

When building a new Vue app, the best way to get up and running quickly is using Vue CLI (<https://cli.vuejs.org/>). This is a command-line utility that allows you to choose from a range of build tools which it will then install and configure for you. It will also scaffold out your project, providing you with a pre-configured starting point that you can build on, rather than starting



everything from scratch.

FREE Book: JavaScript Best Practice

Note: if the CLI is new for you, you might like to check out our tutorial "[A Beginner's Guide to Vue CLI](https://www.sitepoint.com/vue-cli-intro/)"

Check it out!

(<https://www.sitepoint.com/>)

(https://www.sitepoint.com/premium/products/Z2lkOi8vbGVhcm5hYmxlL1Byb2R1Y3QvMzAy?utm_source=smartbar&utm_medium=article&utm_campaign=free-book)

To get started, you'll need to have Node installed on your system. You can do this by downloading the binaries for your system from the [official website](https://nodejs.org) (<https://nodejs.org>), or using a [version manager](https://www.sitepoint.com/quick-tip-multiple-versions-node-nvm/) (<https://www.sitepoint.com/quick-tip-multiple-versions-node-nvm/>). I recommend the second of the two methods.

With Node installed, execute the following command:

```
npm install -g @vue/cli
```

And create a new Vue project with the command:

```
vue create my-project
```

This will open a wizard which asks you to choose a preset. Choose to manually select features, then accept the defaults for everything, apart from when you're asked to pick a linter/formatter config. In this step, be sure to select *ESLint + Prettier* and *Lint on save*, and to place config files in `package.json`.

Linting Your Code with ESLint

If you open this newly created project and take a peek inside the `package.json` file, you'll notice that the CLI has set up [ESLint](https://eslint.org/) (<https://eslint.org/>) for you. This is a tool that can automatically check your code for potential problems. This provides many benefits, such as:

- keeping your code style consistent
- spotting potential errors and bad patterns
- enforcing quality when you follow a [style guide](https://www.sitepoint.com/why-use-javascript-style-guide/) (<https://www.sitepoint.com/why-use-javascript-style-guide/>).
- saving time for all of the above reasons

Note: if you'd like to dive deeper into ESLint, check out our article "[Up and Running with ESLint – the Pluggable JavaScript Linter](https://www.sitepoint.com/up-and-running-with-eslint-the-pluggable-javascript-linter/)" (<https://www.sitepoint.com/up-and-running-with-eslint-the-pluggable-javascript-linter/>).

If you look at the `devDependencies` property in `package.json`, you'll see that the CLI is also using [eslint-plugin-vue](https://github.com/vuejs/eslint-plugin-vue) (<https://github.com/vuejs/eslint-plugin-vue>). This is the official ESLint plugin for Vue.js, which is able to detect code problems in your `.vue` files.



Let's put that to the test.

FREE Book: JavaScript Best Practice

Start the Vue dev server with `npm run serve` and navigate to localhost:8080 (<http://localhost:8080>).

Check it out!

(<https://www.sitepoint.com/>).



(https://www.sitepoint.com/premium/products/Z2lkOi8ybGVhcm5hYmxlL1Byb2R1Y3QvMzAy?utm_source=smartbar&utm_medium=article&utm_campaign=free-book)
In VS Code, open up the project you just created with the CLI (File > Open Folder), then navigate to `src/components/HelloWorld.vue` in the VS Code explorer. Let's add a method to our Vue instance:

```
export default {
  name: 'HelloWorld',
  props: {
    msg: String
  },
  methods: {
    sayHi() {
      console.log("hi");
    }
  }
}
```

Now, if you look at the terminal window in which the dev server is running, you'll see Vue complaining.

```
npm run serve

File Edit View Search Terminal Help

WARNING Compiled with 1 warnings 2:07:02 PM

Module Warning (from ./node_modules/eslint-loader/index.js):
error: Unexpected console statement (no-console) at src/components/HelloWorld.vue:41:7:
 39 |     methods: {
 40 |       sayHi() {
> 41 |         console.log("hi");
    |         ^
 42 |       }
 43 |     }
 44 |   }

1 error found.

You may use special comments to disable some warnings.
Use // eslint-disable-next-line to ignore the next line.
Use /* eslint-disable */ to ignore all warnings in a file.

App running at:
- Local: http://localhost:8080/
- Network: http://192.168.178.27:8080/
```

This is because, under the hood, Vue CLI has configured ESLint to use the `eslint:recommended` ruleset. This enables any rules marked with a check mark on the [ESLint rules page](https://eslint.org/docs/rules/) (<https://eslint.org/docs/rules/>), of which `no-console` (<https://eslint.org/docs/rules/no-console>) is one.



While it's nice that the CLI shows us ESLint errors in the terminal, wouldn't it'd be nicer if we could see them in our editor, too?

Well, luckily we can. Hop back into VS Code, click the search icon and type in "ESLint" (with the double quotes). The top result should be for a **package named ESLint by Dirk Baeumer** (<https://marketplace.visualstudio.com/items?itemName=dpbaurer.vscode-eslint>). Install that and restart VS Code.

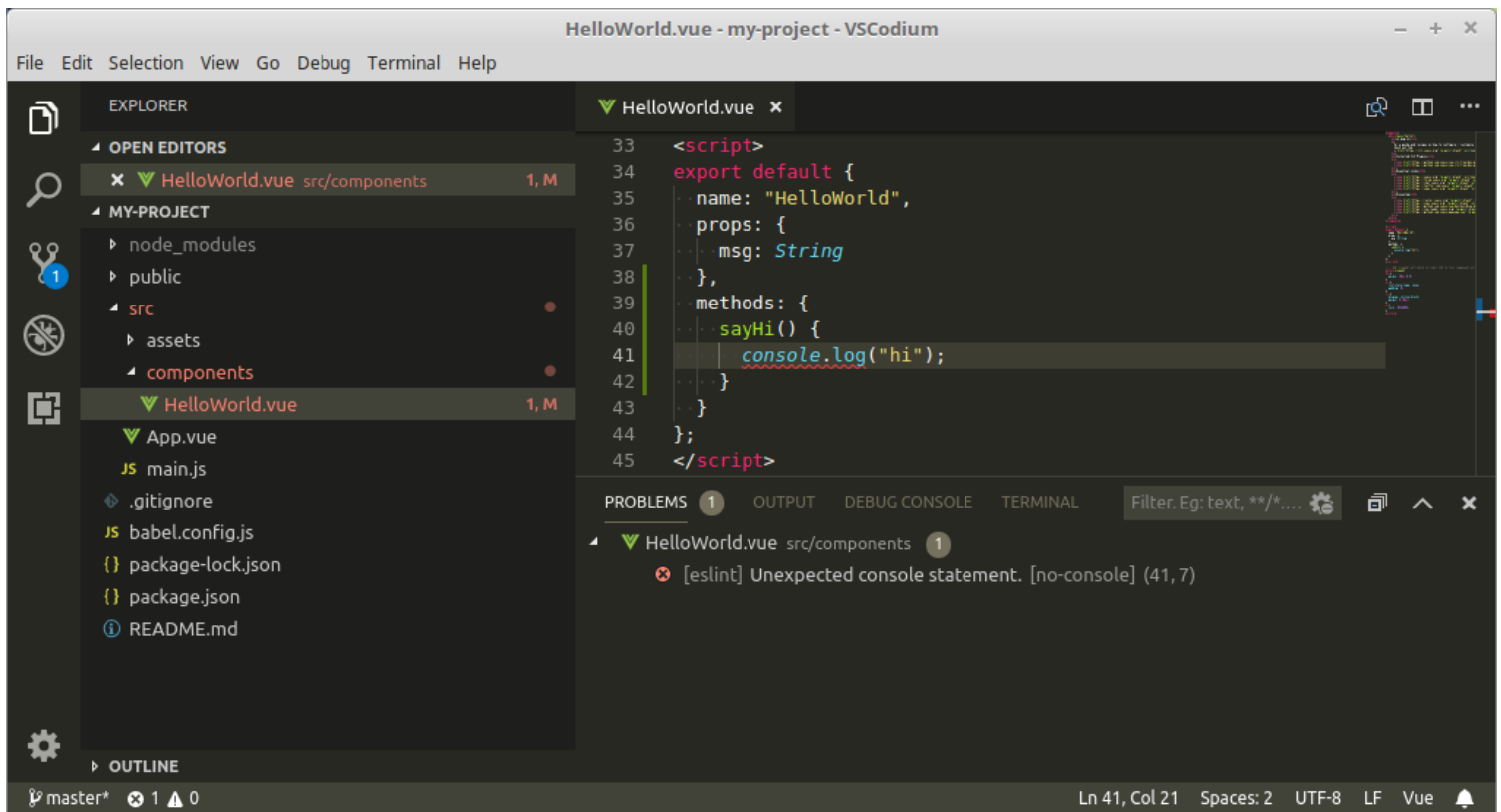
Check it out!

(https://www.sitepoint.com/premium/products/Z2lkOi8vbGVhcm5hYmxlL1Byb2R1Y3QvMzAy?utm_source=smartbar&utm_medium=article&utm_campaign=free-book)

Finally, you'll need to edit your VS Code preferences. Go to **File > Preferences > Settings** and edit the *User Settings* file and add the following configuration:

```
"eslint.validate": [  
  "vue"  
]
```

This will tell the ESLint plugin we just installed to perform validation for **.vue** files.



Should you desire, you can turn this (or any) rule off in the **rules: {}** section of **package.json**:

```
"eslintConfig": {  
  ...  
  "rules": {  
    "no-console": 0  
  },  
  ...  
}
```



Formatting Your Code with Prettier

FREE Book: JavaScript Best Practice

Prettier is an opinionated code formatter. It can be added to your project's tooling (https://prettier.io/docs/en/why-prettier.html), it enforces a consistent style by parsing your code and re-printing it with its own rules that take the maximum line length into account, wrapping code when necessary. (https://www.sitepoint.com/)

Check it out! (https://www.sitepoint.com/premium/products/Z2lkOi8vbGVhcm5hYmxlL1Byb2R1Y3QvMzAy?utm_source=smartbar&utm_medium=article&utm_campaign=free-book)

That might sound a little draconian at first, but once you get used to it, you literally never have to think about code formatting again. This is very useful if you're part of a team, as Prettier will halt all the on-going debates over styles in their tracks.

Note: if you're not convinced, you can read more about why you should use Prettier here (https://prettier.io/docs/en/why-prettier.html).

The way Prettier works in conjunction with Vue CLI is similar to ESLint. To see it in action, let's remove the semicolon from the end of the `console.log("hi");` statement from our previous example. This should display a warning in the terminal:

```
warning: Insert `;` (prettier/prettier) at src/components/HelloWorld.vue:41:24:
 39 |   methods: {
 40 |     sayHi() {
> 41 |       console.log("hi")
    |                               ^
 42 |   }
 43 | }
 44 | };

1 error and 1 warning found.
1 warning potentially fixable with the `--fix` option.
```

It will also display a warning in VS Code, thanks to the ESLint plugin we installed previously.

We can also have Prettier fix any formatting errors for us whenever we save a file. To do this, go to *File > Preferences > Settings* and edit the *User Settings file* to add the following configuration:

```
"editor.formatOnSave": true
```

Now when you press save, everything will be formatted according to Prettier's standard rule set (https://prettier.io/docs/en/rationale.html).

Note, you can configure a handful of rules in Prettier via a `"prettier"` key in your `package.json` file:



```
"prettier": {  
  "semi": false  
}
```

FREE Book: JavaScript Best Practice

Check it out!

(<https://www.sitepoint.com/>).



(https://www.sitepoint.com/premium/products/Z2lkOi8vbGVhcm5hYmxlL1Byb2R1Y3QvMzAy?utm_source=smartbar&utm_medium=article&utm_campaign=free-book)

The above, for example, would turn the semicolon rule off.

You can read more about configuration options [here](https://prettier.io/docs/en/options.html) (<https://prettier.io/docs/en/options.html>).

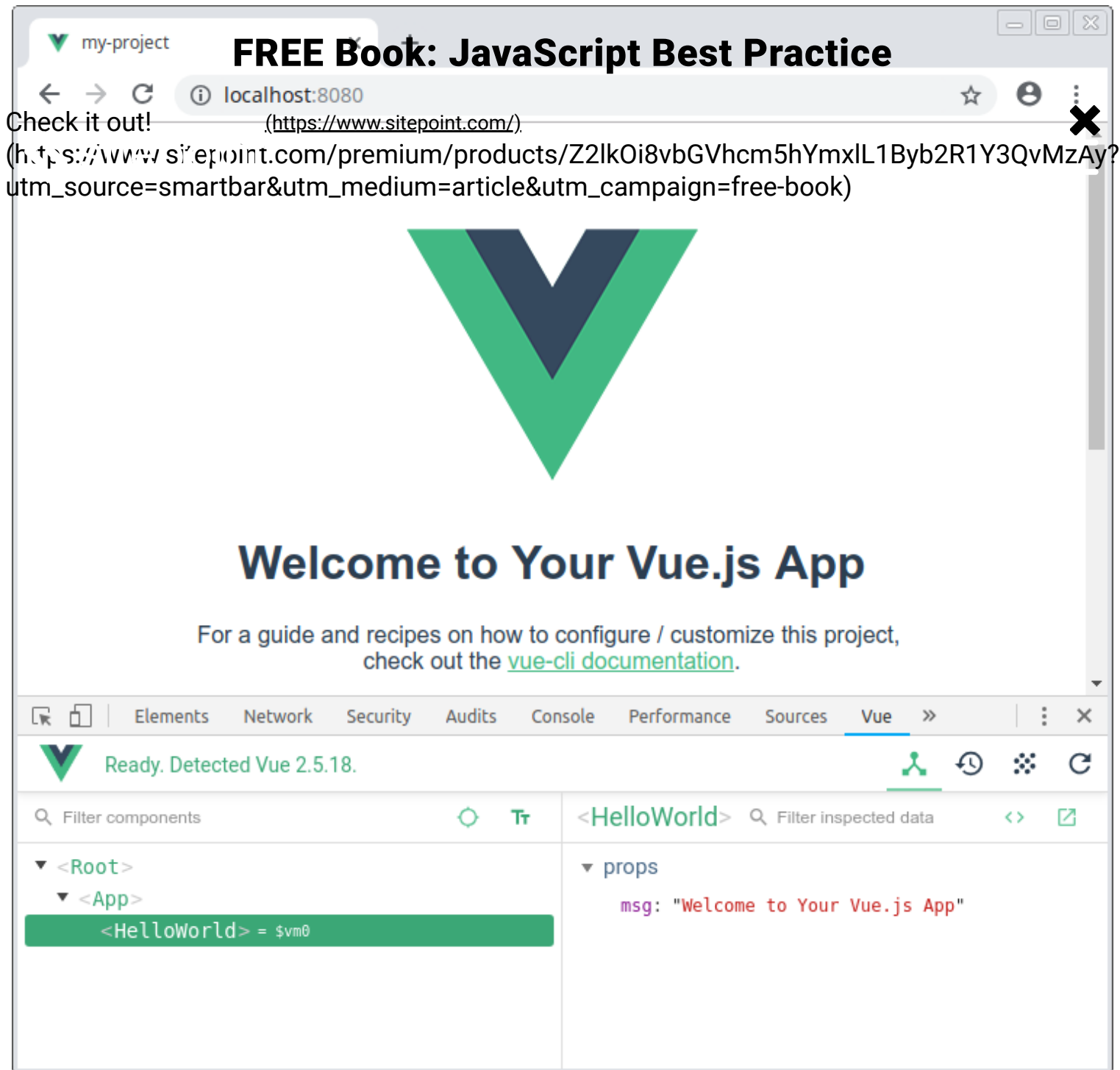
Vue Browser Tools

In this section, I want to take a look at the the Vue.js devtools, which are available as a browser plugin for both [Chrome](https://chrome.google.com/webstore/detail/vuejs-devtools/nhdogjmejiglipccpnnnanhbledajbpd) (<https://chrome.google.com/webstore/detail/vuejs-devtools/nhdogjmejiglipccpnnnanhbledajbpd>) and [Firefox](https://addons.mozilla.org/en-US/firefox/addon/vue-js-devtools/) (<https://addons.mozilla.org/en-US/firefox/addon/vue-js-devtools/>), and as [a cross-platform Electron app](https://github.com/vuejs/vue-devtools/blob/master/shells/electron/README.md) (<https://github.com/vuejs/vue-devtools/blob/master/shells/electron/README.md>), which can also debug Vue apps running on mobile devices.

Once you have them installed, you can access them by visiting a running Vue app, opening your browser's console and hitting the *Vue* button. You'll then see three further sections: *Components*, *Vuex* and *Events*.

The first section gives you a hierarchical view of all the components that make up your application. Selecting a component from the tree allows you to inspect its state (for example, the **props** it received) in the right-hand pane. Some of its values (such as its **data** object) can be dynamically edited while the app runs.





The *Vuex* tab is only active if a *Vuex* store is detected in the application. (For more information on this, check out “Getting Started with *Vuex*: a Beginner’s Guide” in this *Vue* series.) It allows you to examine the state of the store at any point in time, and all the mutations that have been committed. You can even move back and forth through the mutations, effectively time traveling through the state of your application.

The *Events* tab aggregates all the events emitted by your application, from anywhere in the component tree. Selecting an event will display more information about it in the right-hand pane, allowing you to see which component emitted it and any payload that was sent.

There’s a lot more to *Vue*’s browser tools than I’ve demonstrated here, so I encourage you to install them and experiment with them as your application grows.



Conclusion

FREE Book: JavaScript Best Practice

And that's a wrap. In this guide, I've demonstrated how to install the Vetur plugin for VS Code and highlighted several of its features. I've also shown how to use Vue CLI to generate a project and how to use ESLint and Prettier to ensure code quality and consistency. We also took a brief look at Vue's browser tools and saw how to inspect the state of a running Vue application, something which is important for debugging purposes.

Check it out! [\(https://www.sitepoint.com/\)](https://www.sitepoint.com/) https://www.sitepoint.com/premium/products/Z2lkOi8vbGVhcm5hYmxlL1Byb2R1Y3QvMzAy?utm_source=smartbar&utm_medium=article&utm_campaign=free-book

This should see you well on the way to having a sensible environment for writing Vue applications and hopefully make you a productive and happy developer.



Meet the author

James Hibbard (<https://www.sitepoint.com/author/jhibbard/>)

(<https://twitter.com/jchibbard>) (<https://github.com/jameshibbard>)

Currently I work for SitePoint as editor of their [JavaScript hubs](https://www.sitepoint.com/learning-hubs/) (<https://www.sitepoint.com/learning-hubs/>) and technical editor for various books (e.g. [JavaScript: Novice to Ninja](https://www.sitepoint.com/premium/books/javascript-novice-to-ninja-2nd-edition) (<https://www.sitepoint.com/premium/books/javascript-novice-to-ninja-2nd-edition>)) and [Jump Start Vue.js](https://www.sitepoint.com/premium/books/jump-start-vue-js/) (<https://www.sitepoint.com/premium/books/jump-start-vue-js/>)). I also work for my local university as a network admin / web dev where I spend a fair bit of my time working on Rails apps.

Start A Discussion (<https://www.sitepoint.com/community/t/how-to-set-up-a-vue-development-environment/335070>)

Stuff We Do

- [Premium](/premium/) (</premium/>).
- [Forums](/community/) (</community/>).
- [Corporate Memberships](https://sitepoint.typeform.com/to/fNY7XG) (<https://sitepoint.typeform.com/to/fNY7XG>).
- [Jobs](/premium/l/developer-jobs) (</premium/l/developer-jobs>).
- [Deals](/premium/deals) (</premium/deals>).

About

- [Our Story](/about-us/) (</about-us/>).
- [Press Room](/press/) (</press/>).



Contact

FREE Book: JavaScript Best Practices

- [Contact Us \(/contact-us/\)](/contact-us/)

- [Terms of Use \(/legals/\)](/legals/)

Check it out!

(<https://www.sitepoint.com/>)

(https://www.sitepoint.com/premium/products/Z2lkOi8vbGVhcm5hYmxLL1Byb2R1Y3QvMjAy?utm_source=smartbar&utm_medium=article&utm_campaign=free-book)

- [Publish With Us \(https://sitepoint.typeform.com/to/HtAXVN\)](https://sitepoint.typeform.com/to/HtAXVN)

- [Advertise \(/advertise/\)](/advertise/)

Connect



(<https://www.facebook.com/sitepoint>)



(<http://twitter.com/sitepointdotcom>)



(<https://www.sitepoint.com/feed/>)



(<https://plus.google.com/+sitepoint>)

© 2000 – 2019 SitePoint Pty. Ltd.

