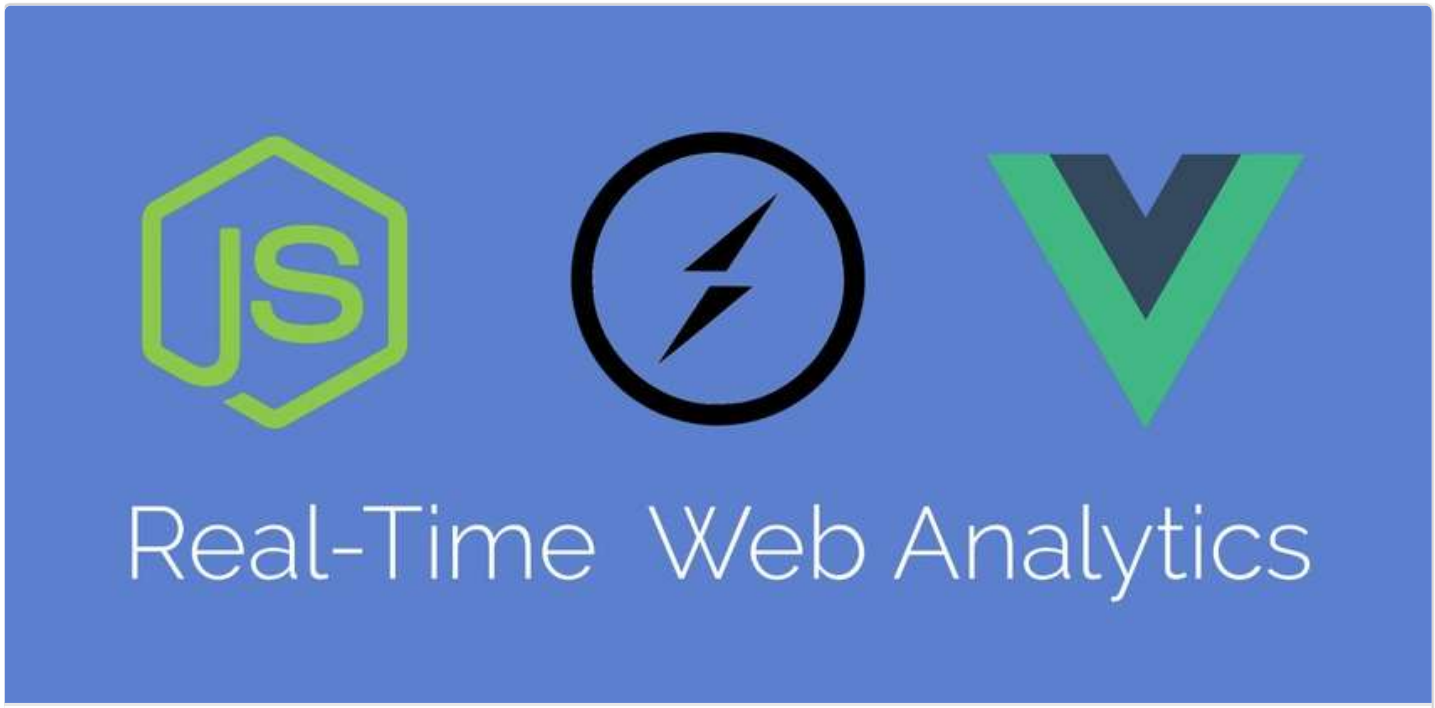
coligo

Real-Time Analytics Dashboard with NodeJs, Socket.io, VueJs

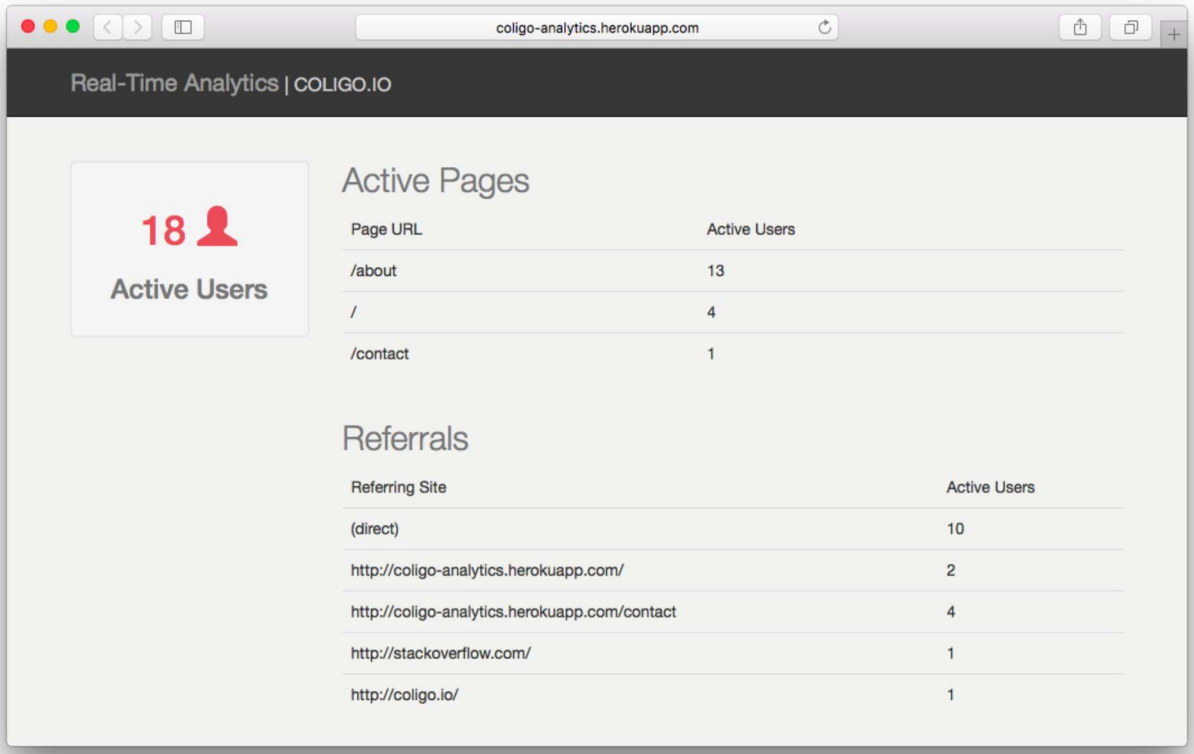
March 14th, 2016 | Node.js, Socket.io, Vue.js



View Demo Get Code

Real-Time Web Analytics with NodeJs, Socket.io, and VueJs

In this tutorial we'll be using NodeJs, Socket.io, and VueJs to build a real-time web analytics dashboard, similar to what you would find on Google Analytics. Have a look at the demo to see what the end product will look like and feel free to clone the GitHub repository which has the completed code for this tutorial.



At a high level, our analytics system will work as follows:

1. A user loads our page
2. A new socket connection is created via the client-side JavaScript
3. The client-side JavaScript sends the NodeJs + Socket.io server information about the user (which page they are on and which website referred them to ours) over that socket connection
4. The server adds the connection to a list of active ones and computes the total counts for the pages and referring sites
5. The server then sends the computed statistics to the dashboard over a socket connection to display the information

Setting Up the Project

Let's start of by setting up the project structure and installing the Node modules we'll be needing for this application.

Go ahead and create an empty directory and change into it:

```
mkdir real-time-analytics && cd real-time-analytics
```

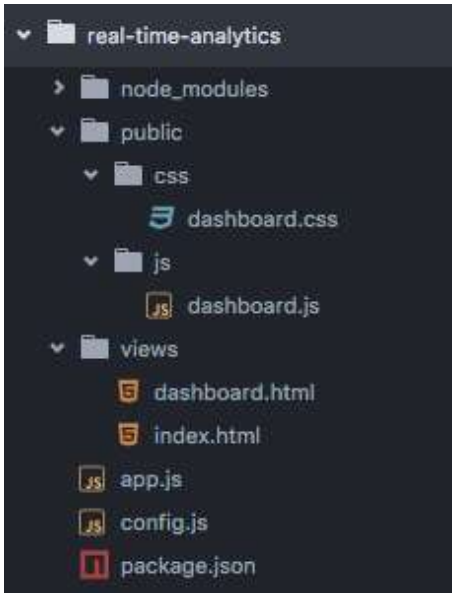
Initialize a new `package.json` file and answer the prompts as you desire:

```
npm init
```

Once you've initialized the `package.json`, install ExpressJs and Socket.io for our server to use:

```
npm install --save express socket.io
```

We should be good to go now! Here's what the final project structure should look like for a visual reference, so that you have an idea of what goes where. We will construct this structure as we go along:



The files and directories worth noting are:

- **public/** directory contains our static assets
- **js/dashboard.js** houses the VueJs and Socket.io code for receiving and rendering the stats to the dashboard
- **css/dashboard.css** has some basic styling for the dashboard
- **views/dashboard.html** contains HTML and Vue directives for data binding and rendering
- **views/index.html** is the page a visitor is served and contains the tracking code
- **app.js** contains our server-side logic
- **config.js** is where we will store any configuration subject to change

Configuring the Node and Socket.io Server

In this section we're going to set up the basic structure for our Node + Socket.io server and configure it to listen to incoming socket connections. Create the **app.js** file in the root of your project directory:

```
// require Express and Socket.io
var express = require('express');
var app = express();
```

```
var http = require('http').Server(app);
var io = require('socket.io')(http);
var path = require('path');
var config = require('./config.js');

// the object that will hold information about the active users currently
// on the site
var visitorsData = {};

app.set('port', (process.env.PORT || 5000));

// serve the static assets (js/dashboard.js and css/dashboard.css)
// from the public/ directory
app.use(express.static(path.join(__dirname, 'public/')));

// serve the index.html page when someone visits any of the following endpoints:
//    1. /
//    2. /about
//    3. /contact
app.get(/\/(about|contact)?$/, function(req, res) {
  res.sendFile(path.join(__dirname, 'views/index.html'));
});

// serve up the dashboard when someone visits /dashboard
app.get('/dashboard', function(req, res) {
  res.sendFile(path.join(__dirname, 'views/dashboard.html'));
});

io.on('connection', function(socket) {
  // a user has visited our page - add them to the visitorsData object
  socket.on('disconnect', function() {
    // a user has left our page - remove them from the visitorsData object
  });
});

http.listen(app.get('port'), function() {
  console.log('listening on *:' + app.get('port'));
});
```

The `visitorsData` object will hold the information collected about each user and associate it to a unique socket ID. This will serve as a way of uniquely identifying a user's socket connection to the data collected about them. A sample of what the `visitorsData` object may look like:

```
{
    "/#HRhTKIFZ1vbQS8vVAAAA": {
        "referringSite": "http://stackoverflow.com",
        "page": "/"
    },
    "/#m2TQwJokQGnNbyNcAAAC": {
```

```
        "referringSite": "http://localhost:5000/",
        "page": "/about"
    }
}
```

The keys of the `visitorsData` object are simply the unique IDs that Socket.io generates for each socket connection.

Also, it's worth noting that you wouldn't typically serve up the same page to three different routes (/, /about, /contact). However, for the purpose of this tutorial we don't really care about what the individual pages look like, but rather the information collected about their URLs.

Adding the Tracking Code to Our Web Pages

Now that we have the server configured to listen to incoming socket connections and serving the pages to their respective routes, we can create the **index.html** file in our **views/** folder. This is the file we will be serving to our our visitors that will automatically execute a small JavaScript snippet to:

1. Get the page the user is on and which site referred them to this page (if any)
2. Create a new socket connection and send that information to the server

Let's focus on the JavaScript tracking snippet first:

```
<script src="/socket.io/socket.io.js"></script>
<script>
    var socket = io();
    var visitorData = {
      referringSite: document.referrer,
      page: location.pathname
    }
    socket.emit('visitor-data', visitorData);
</script>
```

All we're doing here is including Socket.io on the client-side, gathering the data, and emitting a *visitor-data* event with the data we collected about our visitor.

I'd encourage you to explore some of the other cool properties you can collect about the visitors through JavaScript such as:

- user-agent (`navigator.userAgent`)
- browser type (`navigator.vendor`)
- page title (`document.title`)
- and much more...

Let's add the JavaScript snippet above to our HTML file. The complete **index.html** file should look like this:

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Real-Time Analyics</title>

    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/boots
    <style>
        body {
            background-color: #F2F2F0;
        }

        .navbar {
            background-color: #373737;
            padding: 5px;
        }

        .navbar a {
            color: #8C8C8C;
            font-size: 1.5em;
        }

        .coligo {
            color: #D0D0D0;
            font-size: 0.8em;
        }
    </style>
</head>

<body>
    <nav class="navbar navbar-inverse" style="border-radius: 0px;">
        <div class="container">
            <div class="navbar-header">
                <a class="navbar-brand" href="/">Real-Time Analytics<span class="coligo">
            </div>
        </div>
    </nav>

    <div class="container">
        <div class="row">
            <div class="col-xs-12">
                <h3>Open up the analytics dashboard in a new tab/window</h3>
                <h4 class="header"><a href="/dashboard">Analytics Dashboard</a></h4>
                <h3>Use the links below to test the real-time analytics dashboard</h3>
                <h4 class="header"><a href="/">Homepage</a></h4>
```

```
                <h4 class="header"><a href="/about">About Us</a></h4>
                <h4 class="header"><a href="/contact">Contact Us</a></h4>
            </div>
        </div>
    </div>

    <script src="/socket.io/socket.io.js"></script>
    <script>
    var socket = io();
    var visitorData = {
        referringSite: document.referrer,
        page: location.pathname
    }
    socket.emit('visitor-data', visitorData);
    </script>

</body>

</html>
```

Capturing New Visitor Data and Computing Stats

Let's go back to our server code in **app.js** and capture the data that the visitor is sending over the socket connection.

```
io.on('connection', function(socket) {
  // a user has visited our page - add them to the visitorsData object
  socket.on('visitor-data', function(data) {
    visitorsData[socket.id] = data;
  });

  socket.on('disconnect', function() {
    // a user has left our page - remove them from the visitorsData object
    delete visitorsData[socket.id];
  });
});
```

Now whenever the client-side emits a `visitor-data` event we will add the data we captured on the client-side to the `visitorsData` object. Similarly, when they leave the page, the socket will disconnect and trigger a *disconnect* event, in which we will remove the associated socket ID and it's data from the `visitorData` object.

We now have a system in which the visitor data is being added and removed from the *visitorsData* object depending on whether they visit our page or leave it. The next steps are:

1. Compute total counts for active users on each page and referring URLs
2. Emit the computed data to the dashboard to display it

Let's start off with step 1. We'll create 4 simple utility functions that will go through the *visitorsData* object and sum up the total users on each page, the total users per referring URL and the total active users:

```
// wrapper function to compute the stats and return a object with the updated stats
function computeStats(){
  return {
    pages: computePageCounts(),
    referrers: computeRefererCounts(),
    activeUsers: getActiveUsers()
  };
}

// get the total number of users on each page of our site
function computePageCounts() {
  // sample data in pageCounts object:
  // { "/": 13, "/about": 5 }
  var pageCounts = {};
  for (var key in visitorsData) {
    var page = visitorsData[key].page;
    if (page in pageCounts) {
      pageCounts[page]++;
    } else {
      pageCounts[page] = 1;
    }
  }
  return pageCounts;
}

// get the total number of users per referring site
function computeRefererCounts() {
  // sample data in referrerCounts object:
  // { "http://twitter.com/": 3, "http://stackoverflow.com/": 6 }
  var referrerCounts = {};
  for (var key in visitorsData) {
    var referringSite = visitorsData[key].referringSite || '(direct)';
    if (referringSite in referrerCounts) {
      referrerCounts[referringSite]++;
    } else {
      referrerCounts[referringSite] = 1;
    }
  }
  return referrerCounts;
}

// get the total active users on our site
function getActiveUsers() {
  return Object.keys(visitorsData).length;
}
```

We can now use those utility functions to complete step 2 - emitting the computed data to the dashboard:

```
io.on('connection', function(socket) {
  if (socket.handshake.headers.host === config.host
  && socket.handshake.headers.referer.indexOf(config.host + config.dashboardEndpoint) > -1

    // if someone visits '/dashboard' send them the computed visitor data
    io.emit('updated-stats', computeStats());

  }

  // a user has visited our page - add them to the visitorsData object
  socket.on('visitor-data', function(data) {
    visitorsData[socket.id] = data;

    // compute and send visitor data to the dashboard when a new user visits our page
    io.emit('updated-stats', computeStats());
  });

  socket.on('disconnect', function() {
    // a user has left our page - remove them from the visitorsData object
    delete visitorsData[socket.id];

    // compute and send visitor data to the dashboard when a user leaves our page
    io.emit('updated-stats', computeStats());
  });
});
```

Let's break down the crazy *if-statement* in the snippet above. First, create a **config.js** file which will contain our host and dashboard endpoint:

```
module.exports = {
    host: "localhost:5000",
    dashboardEndpoint: "/dashboard"
}
```

You can change these depending on what host your application is running on.

In our *if-statement* above, we're checking to make sure that `socket.handshake.headers.host` is equal to the host we defined in our config file and that `socket.handshake.headers.referer` contains *localhost:5000/dashboard*. If the condition is met, we will send the data to that socket, which is our dashboard socket in this case.

This simply avoids having to compute and send the data every time any new socket is created, whether it's from our visitors or from our dashboard endpoint.

That wraps it up for our server! Here's what the completed **app.js** looks like:

```
// require Express and Socket.io
var express = require('express');
var app = express();
var http = require('http').Server(app);
var io = require('socket.io')(http);
var path = require('path');
var config = require('./config.js');

// the object that will hold information about the active users currently
// on the site
var visitorsData = {};

app.set('port', (process.env.PORT || 5000));

// serve the static assets (js/dashboard.js and css/dashboard.css)
// from the public/ directory
app.use(express.static(path.join(__dirname, 'public/')));

// serve the index.html page when someone visits any of the following endpoints:
//    1. /
//    2. /about
//    3. /contact
app.get(/\/(about|contact)?$/, function(req, res) {
  res.sendFile(path.join(__dirname, 'views/index.html'));
});

// serve up the dashboard when someone visits /dashboard
app.get('/dashboard', function(req, res) {
  res.sendFile(path.join(__dirname, 'views/dashboard.html'));
});

io.on('connection', function(socket) {
  if (socket.handshake.headers.host === config.host
  && socket.handshake.headers.referer.indexOf(config.host + config.dashboardEndpoint) > -1

    // if someone visits '/dashboard' send them the computed visitor data
    io.emit('updated-stats', computeStats());

  }

  // a user has visited our page - add them to the visitorsData object
  socket.on('visitor-data', function(data) {
    visitorsData[socket.id] = data;

    // compute and send visitor data to the dashboard when a new user visits our page
    io.emit('updated-stats', computeStats());
  });

  socket.on('disconnect', function() {
```

```
    // a user has left our page - remove them from the visitorsData object
    delete visitorsData[socket.id];

    // compute and send visitor data to the dashboard when a user leaves our page
    io.emit('updated-stats', computeStats());
  });
});

// wrapper function to compute the stats and return a object with the updated stats
function computeStats(){
  return {
    pages: computePageCounts(),
    referrers: computeRefererCounts(),
    activeUsers: getActiveUsers()
  };
}

// get the total number of users on each page of our site
function computePageCounts() {
  // sample data in pageCounts object:
  // { "/": 13, "/about": 5 }
  var pageCounts = {};
  for (var key in visitorsData) {
    var page = visitorsData[key].page;
    if (page in pageCounts) {
      pageCounts[page]++;
    } else {
      pageCounts[page] = 1;
    }
  }
  return pageCounts;
}

// get the total number of users per referring site
function computeRefererCounts() {
  // sample data in referrerCounts object:
  // { "http://twitter.com/": 3, "http://stackoverflow.com/": 6 }
  var referrerCounts = {};
  for (var key in visitorsData) {
    var referringSite = visitorsData[key].referringSite || '(direct)';
    if (referringSite in referrerCounts) {
      referrerCounts[referringSite]++;
    } else {
      referrerCounts[referringSite] = 1;
    }
  }
  return referrerCounts;
}

// get the total active users on our site
function getActiveUsers() {
  return Object.keys(visitorsData).length;
```

```
}

http.listen(app.get('port'), function() {
  console.log('listening on *:' + app.get('port'));
});
```

Building the Dashboard

We're at the final section of this tutorial - building the dashboard with VueJs. Let's start off by creating the **dashboard.html** file in the **views/** directory that will define our page structure and all the scripts we need to include:

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="utf-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>Real-Time Web Analytics Dashboard - coligo</title>

  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstr
  <link rel="stylesheet" href="css/dashboard.css">
</head>

<body>
  <nav class="navbar navbar-inverse navbar-fixed-top">
    <div class="container">
      <div class="navbar-header">
        <a class="navbar-brand" href="/">Real-Time Analytics<span class="coligo"> | COLIGO
      </div>
    </div>
  </nav>

  <div class="container">
    <div id="app">
      <div class="row">
        <div class="col-xs-3">
          <div class="well">
            <h1 class="dash-red">{{ activeUsers }} <i class="glyphicon glyphicon-user"></i
            <h3 class="text-muted">Active Users</h3>
          </div>
        </div>

        <div class="col-xs-9">
          <h2 class="sub-header">Active Pages</h2>
          <div class="table-responsive">
            <table class="table">
              <thead>
```

```
              <td>Page URL</td>
              <td>Active Users</td>
            </thead>
            <tbody>
              <tr v-for="(page, count) in pages">
                <td>{{ page }}</td>
                <td>{{ count }}</td>
              </tr>
            </tbody>
          </table>
        </div>

        <h2 class="sub-header">Referrals</h2>
        <div class="table-responsive">
          <table class="table">
            <thead>
              <td>Referring Site</td>
              <td>Active Users</td>
            </thead>
            <tbody>
              <tr v-for="(referringSite, count) in referrers">
                <td>{{ referringSite }}</td>
                <td>{{ count }}</td>
              </tr>
            </tbody>
          </table>
        </div>

      </div>

    </div>
  </div>
</div>

<script src="https://cdnjs.cloudflare.com/ajax/libs/vue/1.0.17/vue.js"></script>
<script src="/socket.io/socket.io.js"></script>
<script src="js/dashboard.js"></script>

</body>

</html>
```

The dashboard page is really simple as you can see. I just want to take a second to go over some of the Vue-specific things. If you're not familiar with VueJs, you can read over the VueJs tutorial I wrote to get you started. In short, VueJs is a simple, flexible JavaScript library that shares some similarities with Angular and React. I personally find it's learning curve is much smaller and the API is quite easy to understand!

dashboard.js will contain 3 data properties: the *pages*, *referrers*, and *activeUsers*, all of which the server computes and sends over as we saw in the previous section.

We render the *pages* and *referrers* properties onto the page using the `v-for` directive to loop over the 2 objects and create a row for each of their entires.

Let's move on to **dashboard.js** to create the Vue instance and mount it to our `div` with and id of #app:

```
var socket = io();

var vm = new Vue({
  el: '#app',
  data: {
    pages: {},
    referrers: {},
    activeUsers: 0
  },
  created: function() {
    socket.on('updated-stats', function(data) {
            this.pages = data.pages;
            this.referrers = data.referrers;
            this.activeUsers = data.activeUsers;
    }.bind(this));
  }
});
```

We're creating a new socket connection to the server at the top of the file and using the `new Vue()` constructor to create a new Vue instance mounted to #app.

The *created* property you see in the options object passed to the Vue constructor is called a lifecycle hook. It is invoked only one time and that is when the instance is created. This makes it a great place to assign event handlers for our socket object.

Now whenever an *updated-stats* event is emitted by the server (telling us that a user has been added/removed), the event handler will set the pages, referrers, and activeUsers data properties to the newly computed stats received from the server.

Note: if you're not familiar with `bind(this)`, it is just a way of making the **this** variable point to the Vue instance as we change context. Alternatively, you can use the arrow function (=>) in ES6 to avoid using `bind`.

Let's add the final touches to the dashboard by creating the **dashboard.css** in **public/css**:

```
body {
  background-color: #F2F2F0;
```

```
  padding-top: 80px;
}

.navbar {
  background-color: #373737;
  padding: 5px;
}

.navbar a {
  color: #8C8C8C;
  font-size: 1.5em;
}

.coligo {
  color: #D0D0D0;
  font-size: 0.8em;
}

.well {
  text-align: center;
  margin-top: 20px;
}

.dash-red {
  color: #DD6161;
}

.sub-header{
  color: #777779;
  font-weight: 300;
}
```

And that's it! You can can start up the server by running:

```
node app.js
```

and going to http://localhost:5000/ in your browser.

Wrapping Up

I hope that you've enjoyed reading this tutorial as much as I've enjoyed writing it. We've managed to use some basic JavaScript on the client-side along with a NodeJs + Socket.io server to capture and compute stats about our visitors and transmit it to the analytics dashboard that we built using VueJs.

There is still plenty of room for improvement in terms of adding new features and optimizing the server and dashboard for better performance. Here are some ideas that I'd

encourage you to explore and implement as a great opportunity to learn more about these technologies:

- Collect other data about your visitors:
  - user-agent to tell what browser your visitor is using and whether they are using a smartphone/tablet/desktop
  - preferred user language
  - geolocate your users using the Google Maps API
- Parse the referring URLs to determine if they are from search engines (organic), social (Twitter/FB/Reddit, etc...) or a generic site referral
- Only send the stats that have been updated as opposed to the entire `visitorsData` object
- Determine unique visitors using their IPs
- Add authentication for your dashboard
- Reject cross domain requests so only visitors from a specific domain can create a socket connection to your server
- Add sorting and filtering to the dashboard using VueJs Filters

Post any questions you have in the comments section below and be sure to follow @coligo_io on Twitter to get notified of new tutorials. Feel free to browse some of our other tutorials on coligo or even suggest ones you'd like to see!

That's a great question and the answer is yes, it's definitely possible! On the tracking code we place on our pages, you can specify which server to connect to instead of the one serving the webpage to the visitor. For example, our tracking code would become:

```
var socket = io('http://my-analytics-server.com');
```

instead of:

```
var socket = io();
```

where `my-analytics-server.com` is your `Socket.io` server that tracks connections from different websites you own.

Ideally you should be rejecting any cross domain requests from websites that you don't whitelist in your server.

1 • Reply • Share ›

NetOperatorWibby ➚ Fady Makram • 2 years ago

Oh nice, thanks for the tips!

1 • Reply • Share ›

Arun T • 3 months ago

Hi, Its a very nice article. while running the code, its showing the active users as 0. May i know how to get the valid detail

• Reply • Share ›

Arjun S K • 3 months ago

Good one

• Reply • Share ›

Kofi Mokome • 6 months ago

Pls I have problem deploying a vue project created with vue-cli.
Please can you show me how you deployed your app to heroku?

• Reply • Share ›

espenlg • 7 months ago

Awesome post!

• Reply • Share ›

Văn Hồng Xuân • 9 months ago

Many thanks :D

• Reply • Share ›

Nick • 2 years ago

Is it possible to use this code (with amendments) on other websites which are not running Nodejs. In a sense use the client side JS and paste it on a php based website to track visitors? Or does each website have to specifically be running on node in order for it to work?

• Reply • Share ›

Fady Makram Mod → Nick • 2 years ago

Node.js is being used with Socket.io to create those socket connections to track the visitors. If you'd like to do it with PHP you can just implement with regular web sockets or use a library if one exists

▲ ▼ • Reply • Share ›

reverland • 2 years ago

cool~

▲ ▼ • Reply • Share ›

NetOperatorWibby • 2 years ago

How would I store the results of the analytics to view trends over time?

▲ ▼ • Reply • Share ›

Fady Makram Mod → NetOperatorWibby • 2 years ago

Persisting it in some sort of database along with timestamps should allow you to view trends over time. Redis is particularly good at handling real time data like this at a high velocity but it's worth noting that redis is a data structures store with an *option* of persisting data. So you wont get the flexibility of querying as you would with a SQL database like MySQL/PostgreSQL/etc...

Alternatively you could go with the NoSQL route and use something like MongoDB.

Here are some tutorials on coligo that use MongoDB and redis

Even mixing up redis with a SQL/NoSQL database would be a good way to leverage the benefits of both.

▲ ▼ • Reply • Share ›

NetOperatorWibby → Fady Makram • 2 years ago

I've recently discovered NeDB, which is as flat as you can get with a database. Thanks for the suggestions, I look forward to building upon your work.

EDIT: I'm already using a URL shortener, but it requires PHP. I think I'll move to something simpler.

▲ ▼ • Reply • Share ›

Ari S. • 2 years ago

What editor were you using?
I like the file navigator which has the html5 logo for .html files.

▲ ▼ • Reply • Share ›

Fady Makram Mod → Ari S. • 2 years ago

Hey Ari! I'm using Atom with the file-icons package

▲ ▼ • Reply • Share ›

Safoor Safdar • 2 years ago

Very nice article. Worth reading.

▲ ▼ • Reply • Share ›

Fady Makram Mod ➚ Safoor Safdar • 2 years ago

Thanks! I'm glad you enjoyed it :) Feel free to recommend any other articles you might find interesting/useful

▲ ▼ • Reply • Share ›

ALSO ON **COLIGO**

Building a File Uploader with NodeJs | coligo

43 comments • 2 years ago

C.R.A.Z.Y. — i wanted to use data which i already know --> for example : usernameauto create /uploads/username dirsend file to user

VueJs: Components | coligo

21 comments • 2 years ago

William Park — I've let you know this on twitter before, but I'm absolutely loving these tutorials. You couldn't have made this any easier to

VueJs: The Basics | coligo

17 comments • 2 years ago

Building Faster APIs with NodeJs and redis

6 comments • 2 years ago