PERFECT MAKANJU

How to build a realtime map using Laravel

MAY 10, 2017

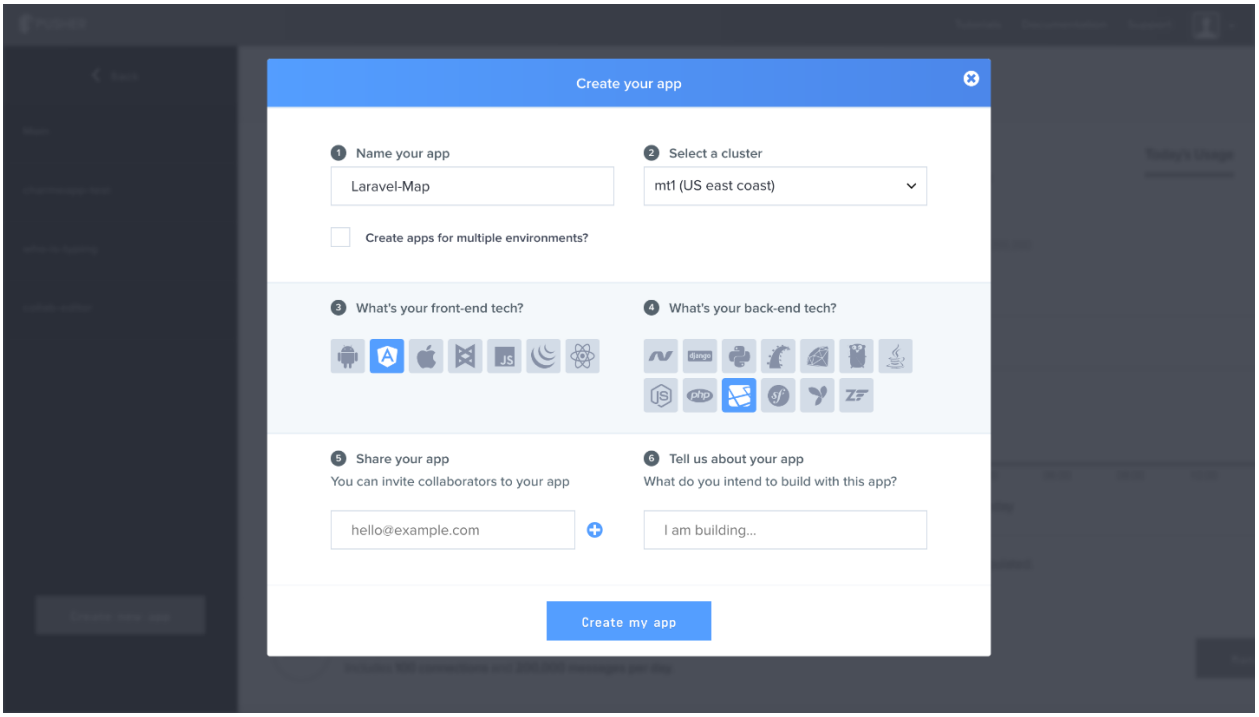
This blog post was written under the Pusher Guest Writer program.
Realtime maps have become very popular with more and more couriers, delivery and transportation services apps using them to show realtime tracking of your order or available vehicles around you. Today we will learn how to build a realtime map using Laravel and Pusher.

What you'll be using

- Pusher
- Google Maps
- Laravel
- Angular

Set up Pusher

To get started you need to sign up to a free Pusher account here. After you have created your account and are logged into the Pusher dashboard, click the `Create new App` button. Name your new app "Laravel-Map" and select a cluster. A cluster represents the physical location of the servers that handle requests from your app, therefore selecting a cluster closer to your location will mean faster response. Next select the front-end and back-end technologies – in this case, Angular and Laravel. (If you want, you can also give a short description of what app you're building.)



When you've done, click the "Create my app" button.

Next, navigate to the "App Keys" tab on the dashboard to see your application keys. Note your application keys because we will be using them later in this tutorial.

Creating the Laravel Project

Navigate into the directory where you would like your apps to be and install Laravel via composer by running the following command in your terminal.

```
composer create-project --prefer-dist laravel/laravel laravel-map
```

Add pusher-php-server to the required dependencies of your composer.json file so that it looks similar to this.

```
"require": {
            "php": ">=5.6.4",
            "laravel/framework": "5.4.*",
            "laravel/tinker": "~1.0",
            "pusher/pusher-php-server": "^2.2"
    },
```

The pusher-php-server is a PHP library which will help us broadcast events to Pusher which will then be listened to by our client side. To install it, run the following command

```
composer install
```

After installation completes, rename the `.env.example` file to `.env` and add the keys you got from Pusher to their respective places in the file. Also set the `BROADCAST_DRIVER` to `pusher` and add a `PUSHER_APP_CLUSTER` key with its value set to your Pusher apps cluster. It should look similar to the following:

```
BROADCAST_DRIVER=pusher
PUSHER_APP_ID=XXXXX
PUSHER_APP_KEY=XXXXXXXXXXXXXXXXX
PUSHER_APP_SECRET=XXXXXXXXXXXXXXXXX
PUSHER_APP_CLUSTER=XXXXXXXXXX
```

Next, open the `config/broadcasting.php` file and set the Pusher cluster option like this:

```
<?php

return [

    ...

    'connections' => [

        'pusher' => [
            'driver' => 'pusher',
            'key' => env('PUSHER_APP_KEY'),
            'secret' => env('PUSHER_APP_SECRET'),
            'app_id' => env('PUSHER_APP_ID'),
            'options' => [
                //add this line
                 'cluster' => env('PUSHER_APP_CLUSTER'),
            ],
        ],

    ...
```

Next, run the following command to generate a Laravel secret key.

```
php artisan key:generate
```

We will be using a public channel to make this tutorial as simple as possible, so we won't be changing anything else.

Next, let us create a SendLocation event. To do this, run the following Laravel command:

`php artisan make:event SendLocation`
You should see a PHP file in your `app/Events` directory named `SendLocation.php`. The only thing we will be changing in the file is the channel-name. We will also implement the ShouldBroadcast interface and add a public variable to it. The complete code should look similar to the following:

```
<?php

namespace App\Events;

use Illuminate\Broadcasting\Channel;
use Illuminate\Queue\SerializesModels;
use Illuminate\Foundation\Events\Dispatchable;
use Illuminate\Broadcasting\InteractsWithSockets;
use Illuminate\Contracts\Broadcasting\ShouldBroadcast;

class SendLocation implements ShouldBroadcast
{
    use Dispatchable, InteractsWithSockets, SerializesModels;

    /**
  * Create a new event instance.
  *
  * @return void
  */
    public $location;

    public function __construct($location)
    {
        $this->location = $location;
    }

    /**
  * Get the channels the event should broadcast on.
  *
  * @return Channel|array
  */
    public function broadcastOn()
    {
        return new Channel('location');
    }
}
```

Finally, create an endpoint where coordinates will be sent. When requests are made to the endpoint, the `SendLocation` event will be triggered and the coordinates will be sent to Pusher. We'll do that in our web.php file located in the `routes` folder. Add the code below the file:

```
Route::post('/map', function (Request $request) {
    $lat = $request->input('lat');
    $long = $request->input('long');
    $location = ["lat"=>$lat, "long"=>$long];
    event(new SendLocation($location));
    return response()->json(['status'=>'success', 'data'=>$location]);
});
```

Set up a Google Map project

We will be using Google Maps to implement our realtime map. This guide will run you through registering a project in the Google API Console and activating the Google Maps JavaScript API. Remember to grab the API key that will be generated for you after registering.

Working with Angular

We'll be using Angular for our front-end. If you don't have angular installed, run the following command:

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

Now create an Angular app with the following command:

```
ng new angular-map
```

Next we will install dependencies to listen to events sent to Pusher by our server. Pusher has a JavaScript library for front-end technologies which we'll be using to listen for events from Pusher.

We'll also be installing Laravel Echo. To do this, navigate into the "angular-map" project and run the following command:

```
npm install --save laravel-echo pusher-js
```

Now that we have installed the dependencies, let's get to the code. In your index.html file, add Pusher and Google Maps scripts. Your index.html file should look similar to the code snippet below:

```
<!doctype html>
<html>
<head>
  <title>Simple Map</title>
```

```
  <meta name="viewport" content="initial-scale=1.0">
  <meta charset="utf-8">

</head>
<body style="margin: 0">

<script src="https://js.pusher.com/3.0/pusher.min.js"></script>
<script
src="https://maps.googleapis.com/maps/api/js?key=YOUR_GOOGLE_APP_KEY"></script>
  <app-root>Loading...</app-root>

</body>
</html>
```

We need somewhere to render the map, so add the following line to your app.component.html file:

```
<div id="map"></div>
```

Finally, we move over to our app.component.ts file to add the code which will render Google Maps on our HTML page.

```
import { Component, OnInit } from '@angular/core'
import * as Echo from 'laravel-echo';

declare var google:any;

const PUSHER_API_KEY = 'YOUR_PUSHER_KEY';
const PUSHER_CLUSTER = 'YOUR_PUSHER_CLUSTER';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})

export class AppComponent implements OnInit{

  lat : number = 9.0820;
  long : number = 8.6753;

  ngOnInit() {

    this.launchMap(this.lat, this.long);

  }
```

```
  launchMap(lat, lng){
    var nigeria= {lat: lat, lng: lng};
    this.map = new google.maps.Map(document.getElementById('map'), {
      zoom: 8,
      center: nigeria
    });
    this.marker = new google.maps.Marker({
      map: this.map,
      animation:"bounce",
    });
    this.lineCoordinates.push(new google.maps.LatLng(this.lat, this.long));
  }
}
```

Next we add the code which listens to the event and updates the map when the coordinates change.

```
subscribe(){
    var echo = new Echo({
      broadcaster: 'pusher',
      key: PUSHER_API_KEY,
      cluster: PUSHER_CLUSTER
    });
    echo.channel('location')
      .listen('SendLocation', (e)=>{
          this.data = e.location;
           this.updateMap(this.data);
      });
  }

updateMap(data){
    this.lat = parseFloat(data.lat);
    this.long = parseFloat(data.long);

    this.map.setCenter({lat:this.lat, lng:this.long, alt:0});
    this.marker.setPosition({lat:this.lat, lng:this.long, alt:0});

    this.lineCoordinates.push(new google.maps.LatLng(this.lat, this.long));

    var lineCoordinatesPath = new google.maps.Polyline({
      path: this.lineCoordinates,
      geodesic: true,
      map: this.map,
      strokeColor: '#FF0000',
      strokeOpacity: 1.0,
```

```
      strokeWeight: 2
    });
```

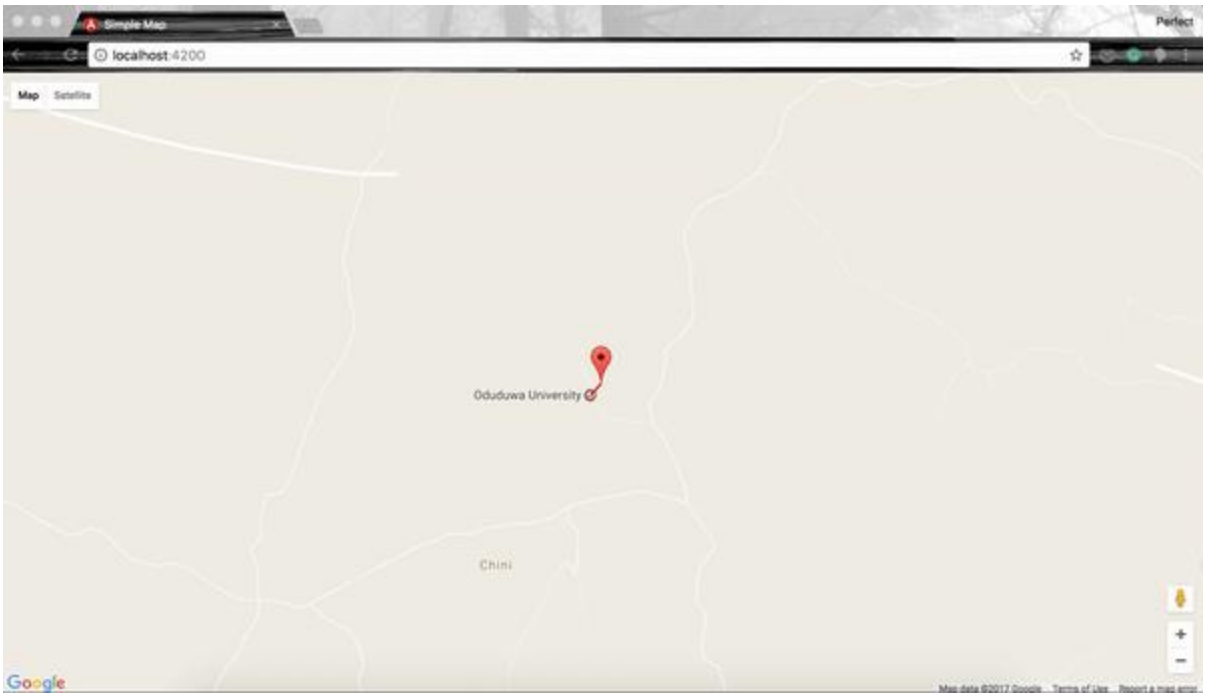
The complete code snippet for the app.component.ts file:

```
import { Component, OnInit } from '@angular/core';
import * as Echo from 'laravel-echo';
import * as Pusher from 'pusher-js';

declare let google:any;

const PUSHER_API_KEY = 'YOUR_PUSHER_KEY';
const PUSHER_CLUSTER = 'YOUR_PUSHER_CLUSTER';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})

export class AppComponent implements OnInit{
  data : any;
  map : any;
  lat : number = 9.0820;
  long : number = 8.6753;
  marker : any;
  lineCoordinates = [];

  ngOnInit() {

    this.subscribe();
    this.launchMap(this.lat, this.long);

  }


  subscribe(){
    let echo = new Echo({
      broadcaster: 'pusher',
      key: PUSHER_API_KEY,
      cluster: PUSHER_CLUSTER
    });
    echo.channel('location')
      .listen('SendLocation', (e)=>{
          this.data = e.location;
           this.updateMap(this.data);
```

```
        });
    }

    launchMap(lat, lng){
        let nigeria= {lat: lat, lng: lng};
        this.map = new google.maps.Map(document.getElementById('map'), {
            zoom: 14,
            center: nigeria
        });
        this.marker = new google.maps.Marker({
            map: this.map,
            animation:"bounce",
        });
        this.lineCoordinates.push(new google.maps.LatLng(this.lat, this.long));
    }

    updateMap(data){
        this.lat = parseFloat(data.lat);
        this.long = parseFloat(data.long);

        this.map.setCenter({lat:this.lat, lng:this.long, alt:0});
        this.marker.setPosition({lat:this.lat, lng:this.long, alt:0});

        this.lineCoordinates.push(new google.maps.LatLng(this.lat, this.long));

        let lineCoordinatesPath = new google.maps.Polyline({
            path: this.lineCoordinates,
            geodesic: true,
            map: this.map,
            strokeColor: '#FF0000',
            strokeOpacity: 1.0,
            strokeWeight: 2
        });
    }
}
```

Realtime map demo

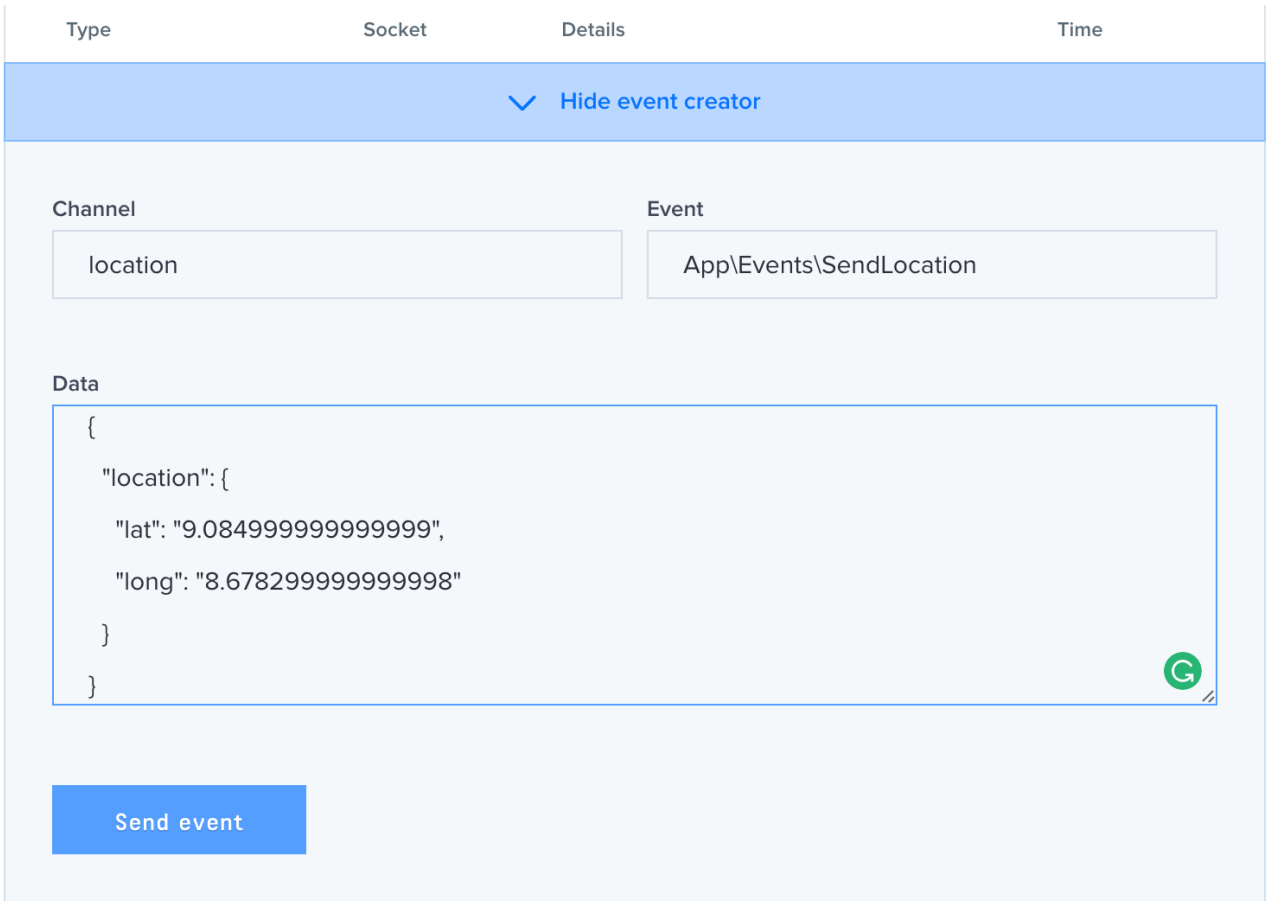
Here is a gif showing the map being updated in realtime:



In order to see the marker move on the map, you will need to send `App\Events\SendLocation` events to the `location` channel. The easiest way to do this is by using the event creator on the Pusher Debug Console. Here is a sample data format that can be used to trigger an update:

```
{
  "location": {
    "lat": "9.084999999999999",
    "long": "8.678299999999998"
  }
}
```

Here is an image of how the event would look on the Pusher event creator:



Alternatively, location updates can also be triggered by sending web requests to the Laravel application. Here is an Angular function that can be used to send location update requests:

```
sendLocation(lat: string, long: string) {
  const serverUrl = 'http://localhost:8000';
  const params = new URLSearchParams();
  params.set("lat", lat);
  params.set("long", long);

  return this.http.post(serverUrl + '/map', params);
}
```

Note that the above function is assuming that the Laravel application is accessible via 'http://localhost:8000', you can update the `serverUrl` to your own configuration.

Conclusion

We have successfully created a real time map which will be updated when new coordinates are sent to the endpoint on our server. You could use this to track the

location of almost anything – but remember, if you are using this in production, you should consider privacy needs and build that functionality in.

You can find the complete project in these repositories angular app and laravel app.