PUSHER
A **MessageBird** company

Create a realtime results feed with React and Node.js

> **To follow this tutorial you will need basic knowledge of React and Node.js.**

In November 2020 **The Washington Post broke Pusher records** as they delivered realtime results coverage of one of their largest news events to date using a WebSocket implementation with Pusher Channels. Channels offers substantial scope for building dynamic realtime features into your apps.

In this tutorial we'll show you how you can deliver engaging live data updates to your own users by building a dynamic realtime results app using Channels (and add some bonus features using Beams, our comprehensive push notifications API). We'll use some sample data to build an app which reports live votes.

Prerequisites

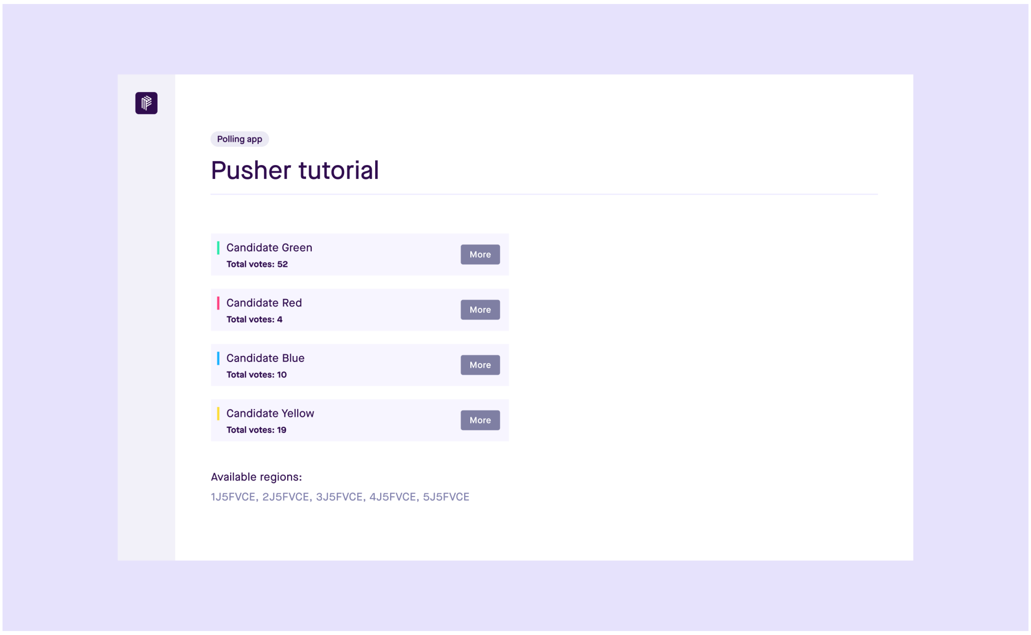
For this tutorial, you need a basic knowledge of React and Node.js. We will also use the following :

- React hooks and functional components for the frontend
- A Node js environment for the backend with the following packages:
    - Express.js
    - pusher-js
    - CORS
    - @pusher/push-notifications-server

- A Pusher account; you can create one here if you haven't done that already. You can sign up for free Sandbox plans to get you started with Pusher and Beams, and easily grow your plan with your usage by following the self-serve tiers.

Objective

At the end of this tutorial, our demo app will stream live votes being fed from the server in real time. You'll also be able to subscribe to receive web notifications in your browser via Beams, to inform you of major events, like the result of a particular poll.



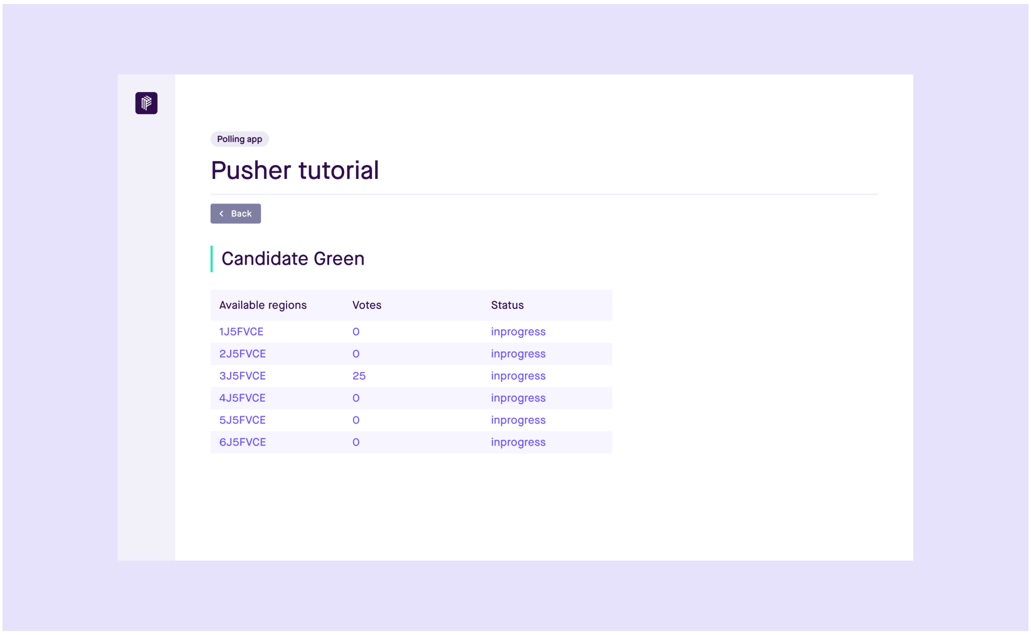
Candidate Green
Total votes: 52

Candidate Red
Total votes: 4

Candidate Blue
Total votes: 10

Candidate Yellow
Total votes: 19

Available regions:
1J5FVCE, 2J5FVCE, 3J5FVCE, 4J5FVCE, 5J5FVCE



Candidate Green

Available regions	Votes	Status
1J5FVCE	0	inprogress
2J5FVCE	0	inprogress
3J5FVCE	25	inprogress
4J5FVCE	0	inprogress
5J5FVCE	0	inprogress
6J5FVCE	0	inprogress

Housekeeping

First things first, there are some basic things we need to put in place or take note before moving on to creating our live poll app.

Using the Global State Concept

Since we are building a relatively small web app for this tutorial, we won't use a state management module like redux or mobx for the app. Instead, we'll use the global state concept where we will cascade all relevant states at the top level component; in this case, App.js. Afterwards we'll change and pass them to other child components. There is also no routing module like redux used in this tutorial.

Getting started with the Front End

Let's get started with the front end and bootstrap with "create react app".

In the terminal, run:

```
create-react-app polling_app
```

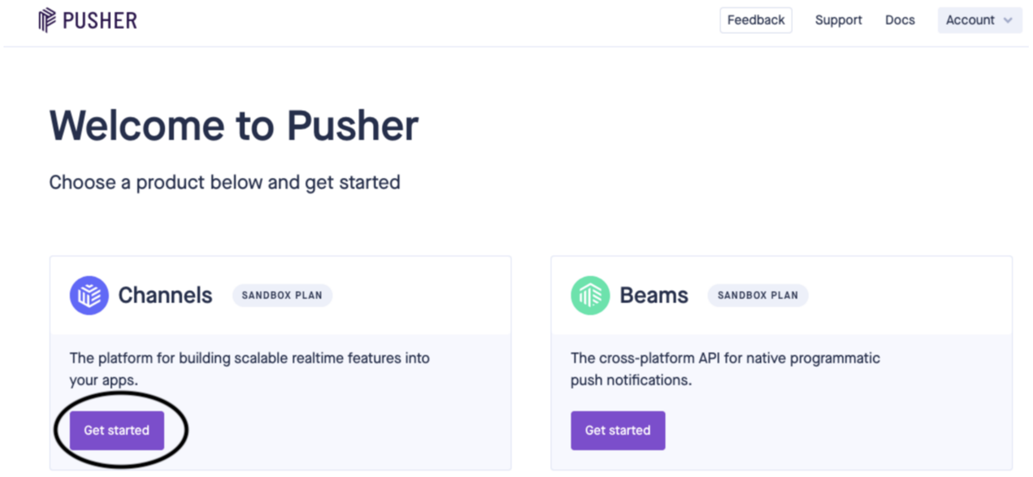
Wait for all the dependencies to be installed. Afterwards, we'll use the pusher-js module to connect and receive data from the server in real time. That is, we will install pusher-js and and @pusher/push-notifications-web to handle the web notifications. To do this, navigate to the polling_app directory in your terminal and run:

```
npm install pusher-js @pusher/push-notifications-web
```

Now, let's create a Channels app.

Creating a Channels App

Navigate to your Pusher account dashboard and click on "Get started" under Channels.



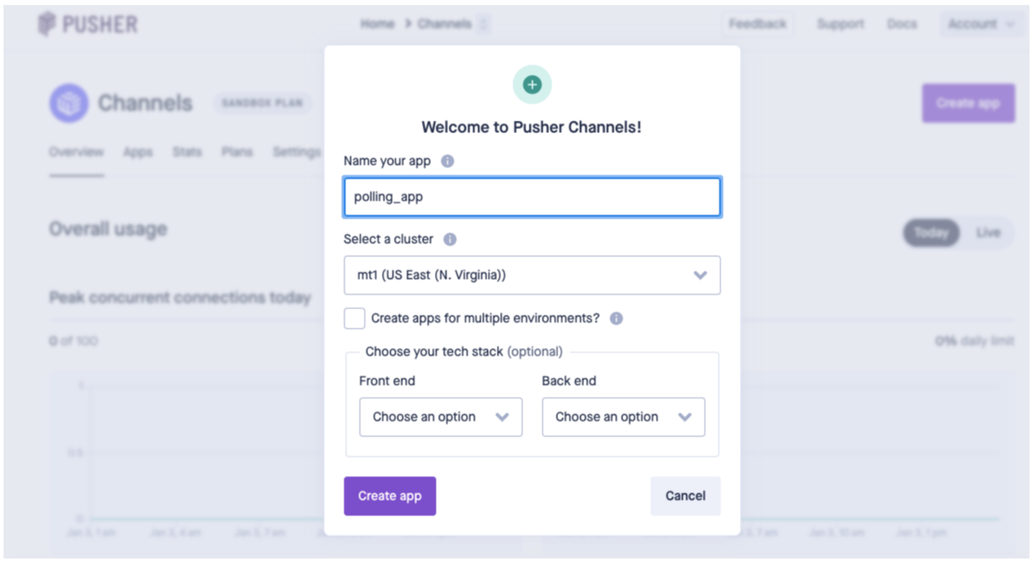
Welcome to Pusher

Choose a product below and get started

Channels — The platform for building scalable realtime features into your apps.

Beams — The cross-platform API for native programmatic push notifications.

Follow the steps in the dialog box to name your app and select your preferred cluster location. Then, click on **Create app**.



Welcome to Pusher Channels!

Name your app: polling_app

Select a cluster: mt1 (US East (N. Virginia))

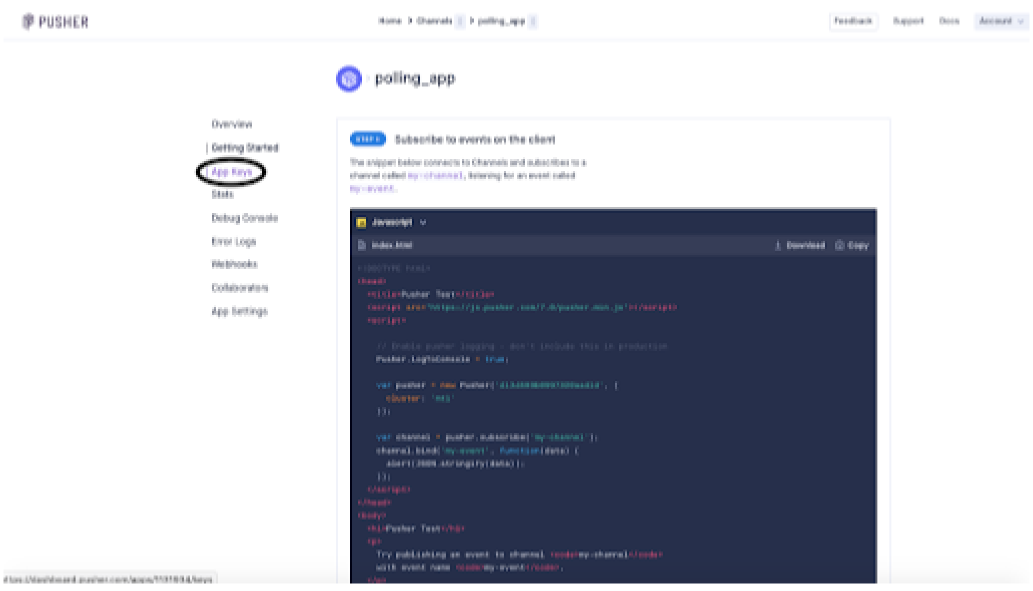
☐ Create apps for multiple environments?

Choose your tech stack (optional)

Front end: Choose an option

Back end: Choose an option

Next you'll see the following page. Here, the details you need for connections can be found under **App Keys.**



Subscribe to events on the client

The snippet below connects to Channels and subscribes to a channel called my-channel, listening for an event called my-event.

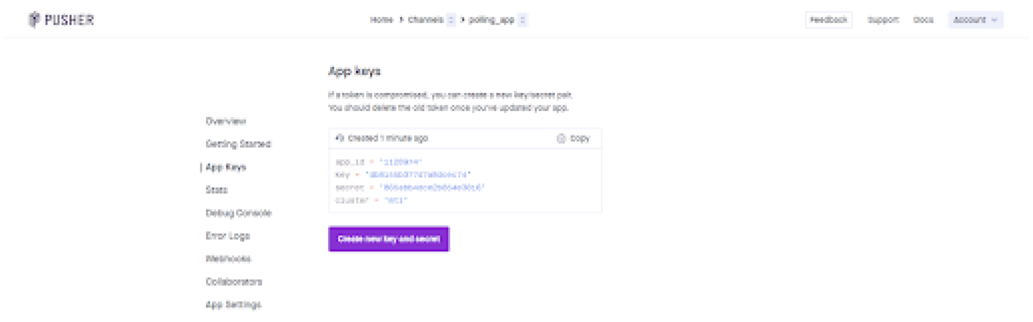
```
<!DOCTYPE html>
<head>
  <title>Pusher Test</title>
  <script src="https://js.pusher.com/7.0/pusher.min.js"></script>
  <script>

    // Enable pusher logging - don't include this in production
    Pusher.logToConsole = true;

    var pusher = new Pusher('[illegible]', {
      cluster: 'mt1'
    });

    var channel = pusher.subscribe('my-channel');
    channel.bind('my-event', function(data) {
      alert(JSON.stringify(data));
    });
  </script>
</head>
<body>
  <h1>Pusher Test</h1>
  <p>
    Try publishing an event to channel <code>my-channel</code>
    with event name <code>my-event</code>.
  </p>
```

Click on **App Keys** in the left hand menu and copy the information provided (including `app_id`, `key`, `secret`, and `cluster`) as shown below. Keep the information safe so that you can use it soon.



App keys

If a token is compromised, you can create a new key/secret pair. You should delete the old token once you've updated your app.

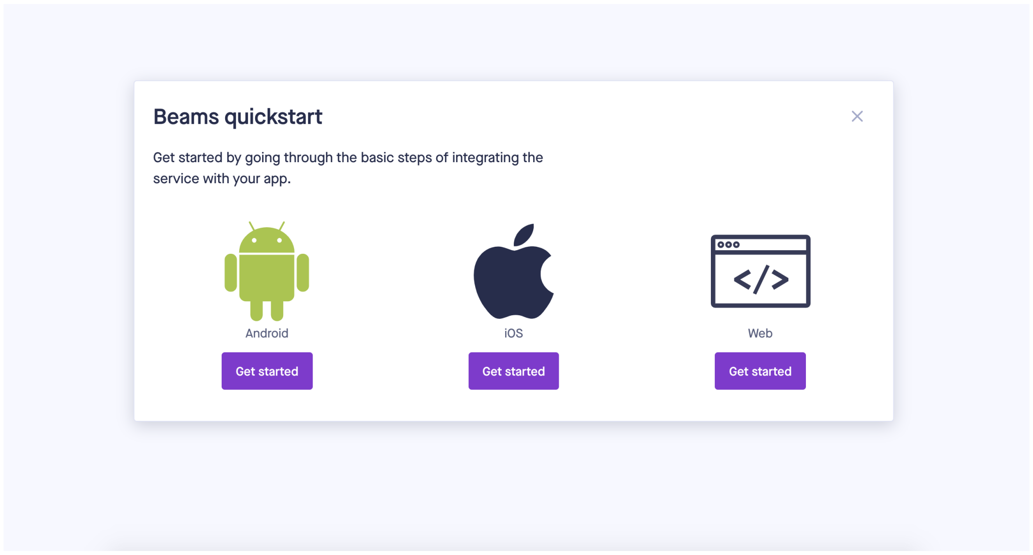
Created 1 minute ago

```
app_id = '[illegible]'
key = '[illegible]'
secret = '[illegible]'
cluster = '[illegible]'
```

Creating a Beams App

Follow the same steps but this time create a Beams App. Once you have created an instance, it's time to select your preferred option from Android, iOS and Web. We're going to select Web, of course, since we are writing a web application.

Go through the 4-step Quick Start guide that is displayed. These steps include creating a Service Worker, installing the SDK and registering your first web device.

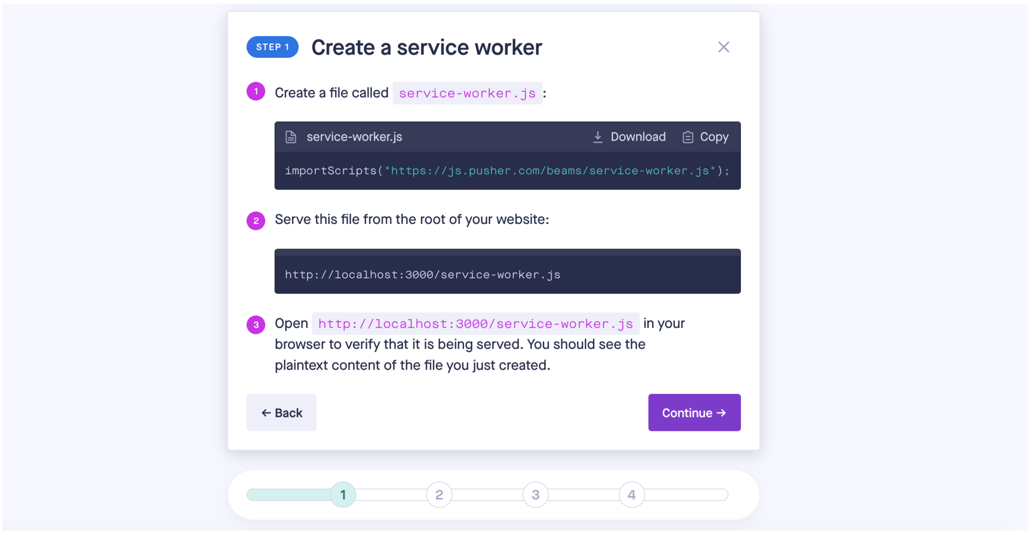


Get started by going through the basic steps of integrating the service with your app.

Creating a Service Worker

Following the steps shown below, create a service-worker.js in the public folder and make sure it registers before proceeding to write your code.

The service worker file is run separately from the web pages in the background. It is the main reason developers can create interactions like push notifications which we will be using in this tutorial and for background sync.



1 Create a file called `service-worker.js`:

```
importScripts("https://js.pusher.com/beams/service-worker.js");
```

2 Serve this file from the root of your website:

```
http://localhost:3000/service-worker.js
```

3 Open `http://localhost:3000/service-worker.js` in your browser to verify that it is being served. You should see the plaintext content of the file you just created.

Building the Realtime Poll Results App

Front End

Navigate to the src folder and create the userComponent folder. This folder will house all the components we will use. Under userComponent, create the following components:

1. States component which will show the general states of the votes for all candidates (red, yellow, blue and green)
2. State component which will show the state for a candidate
3. A modal component which will show the details of voting events happening at different regions for each candidate
4. A header
5. A footer

The structure of the folder once all components are created should look like the following:

∨ src

∨ userComponent

> footer
> header
> modal
> state
> states

Now, let’s edit our App.js file. First, import Pusher and Pusher web notification modules and create a functional component.

To use Pusher, you need to create a new instance of the class “Pusher” and subscribe to a channel. The channel is then bound to an event, so that when an event is fired (i.e when a vote is cast), an action can be performed in the front end.

```
const pusher = new Pusher(Key, {
   cluster: cluster
  });
const channel = pusher.subscribe(‘votes’);
channel.bind(‘vote-event’, function(data) {
   return data
  });
```

For the web notification, the Pusher push notification client class is called with the `instance_ID` you generated earlier when creating your Beams app in the Pusher dashboard.

Subscribe the user’s device to an event:

```
const beamsClient = new PusherPushNotifications.Client({
   instanceId: instanceId,
  })
beamsClient.start()
.then(() => beamsClient.addDeviceInterest(‘votes’))
.then(() => {

});
```

The code can be combined together as below:

```
import * as PusherPushNotifications from “@pusher/push-notifications-web”
import Pusher from ‘pusher-js’;
function App(){
   const pusher = new Pusher("APP_KEY", {
      cluster: "APP_CLUSTER"
```

```
    });
  const channel = pusher.subscribe(‘votes’);
  channel.bind(‘vote-event’, function(data) {
    return data
    });
  const beamsClient = new PusherPushNotifications.Client({
    instanceId: INSTANCE_ID,
    })
  beamsClient.start()
  .then(() => beamsClient.addDeviceInterest(‘votes’))
  .then(() => {

    });
return (
  <div>
    <div style={{display:”flex”, justifyContent:”space-around”}}><States da
    {data} /></div>
  </div>
);

}
export default App;
```

The States Component

The states component will be in the index.js file inside the state folder we created earlier. This functional component provides the current state of votes for all the candidates. This is written as:

```
Import State from “../state/index”
Import states from “./states”
Export default function States (props) {
  return (
  <main style={{display:"flex", flexWrap:"wrap", justifyContent:"space-
  {states.map((state, i) =>{
    return <State index={i} key={i} src={state.src} desc={state.desc} data=
   })}
  </main>
  )
}
```

The states data is located in a states file. It is an array written as:

```
const states = [
    {
        id:"1",
        name: 'Green',
        src:"https://via.placeholder.com/150",
           },
    {
        id:"2",
        name: 'Red',
        src:"https://via.placeholder.com/150",
           },
    {
        id:"3",
        name: 'Yellow',
        src:"https://via.placeholder.com/150",
      },
    {
        id:"4",
        name: 'Blue',
        src:"https://via.placeholder.com/150",

    }
]

export default states
```

The State Component

The State component provides details about each candidate:

```
import { useState } from "react"

import Modal from "../modal/index"
export default function State(props) {
    console.log(props)
    const [display, setDisplay] = useState('none')
  return (
      <main style={{ width:"50%", marginBottom:"15px", marginTop:"5px"}}>
        <div className="">
            <div>
                 <img src={props.src} alt=""/>
            </div>
            <div>
                <p>{props.price}</p>
                <p>Total votes: {props.data.message[props.index + 6]} </p>
            </div>
        </div>
        <div>
            <button onClick={()=>setDisplay("")}>More</button>
            {/* <button>bid</button> */}
        </div>
        <Modal index={props.index} display={display} regions={props.data.me
      </main>
  )
}
```

The Modal Component

The modal component shows what is going on with the idividual regions:

```
import { useState } from "react";
export default function Modal(props){
    // const x = props.status
    const sampleData = [
      [ 1, '1J5FVCE', 0, 0, 0, 0, 0, 0, 0, 0, 'inprogress' ],
      [ 1, '2J5FVCE', 0, 0, 0, 0, 0, 0, 0, 0, 'inprogress' ],
      [ 1, '3J5FVCE', 0, 0, 0, 0, 0, 0, 0, 0, 'inprogress' ],
      [ 1, '4J5FVCE', 0, 0, 0, 0, 0, 0, 0, 0, 'inprogress' ],
      [ 1, '5J5FVCE', 0, 0, 0, 0, 0, 0, 0, 0, 'inprogress' ],
      [ 1, '6J5FVCE', 0, 0, 0, 0, 0, 0, 0, 0, 'inprogress' ]
    ]
    const regions = [['regx', [0]], ['regx', [0]]]
    const [reg, updateReg] = useState(sampleData)
    return (
        <main style={{position:"fixed", display: props.display, zIndex:2, ba
               <p className="" style={{float:"right", color:"black"}} onCli
               <table style={{textAlign:"center", color:"red", width:"100%"
               <table>
  <thead>
    <tr>
      <th>Region Name</th>
      <th>Votes</th>
      <th>Status</th>
    </tr>
  </thead>
  <tbody>
    {reg.map((region)=>{
        return <tr>
        <td>{region[1]}</td>
        <td>{region[props.index + 2]}</td>
        <td>{region[10]}</td>
      </tr>
    })}
  </tbody>
</table>
</table>
        </main>
    )
}
```

Server

The server is very straightforward. The “vote cast” is already provided as elements in an array. The general representation is given as:

`batchnumber`, `region`, `red`, `green`, `yellow`, `blue`, `redtotalvotes`, `greentotalvotes`, `yellowtotalvotes`, `bluetotalvotes`, `status`]

A `setInterval` method will be used as a means through which each “vote is cast”.

Our server is written in a Node.js environment. To get started, we’ll create a folder named server. Navigate to the folder in the terminal and run:

```
npm init
```

You need to answer all the prompts and install the dependencies needed. That is, pusher-js express @pusher/push-notifications-server. To do this, navigate to the server folder on the terminal and run:

```
npm install pusher-js express @pusher/push-notifications-server.
npm install pusher
```

Inside the index.js file, you will instantiate a new Pusher class with `app_ID`, `key`, `secret`, `cluster`, and `useTLS` passed into the object argument.

```
const pusher = new Pusher({
  appId: "app_id",
  key: "key",
  secret: "secret",
  cluster: "cluster",
    useTLS: true
  });
```

We're going to use the `setInterval` method to trigger a "vote" every 60 seconds.

```
setInterval(() => {
  if(i == 5){
    i = 0
  }
  pusher.trigger(“votes”, “vote-event”, {
    message: sampleData[i]
  }).then(console.log).catch(e=> console.log(e))
  i++
}, 60000);
```

The complete code on the server should be as below:

```
const express = require("express")
const Pusher = require("pusher")
const PushNotifications = require('@pusher/push-notifications-server');
const generateVotes = require("./votegenerator");
const mySampleRegions = [
  {regionname: '1', population:2189499, turnoutpercent: 51, totalvotes: 111
  {regionname: '2', population:2189499, turnoutpercent: 51, totalvotes: 111
  {regionname: '3', population:2189499, turnoutpercent: 51, totalvotes: 111
  {regionname: '4', population:2189499, turnoutpercent: 51, totalvotes: 111
  {regionname: '5', population:2189499, turnoutpercent: 51, totalvotes: 111
  {regionname: '6', population:2189499, turnoutpercent: 51, totalvotes: 111
  {regionname: '7', population:2189499, turnoutpercent: 51, totalvotes: 111
]
const sampleData = [
  [ 1, '1J5FVCE', 52, 4, 19, 10, 52,4, 19, 10, 'inprogess',1 ],
  [ 1, '2J5FVCE', 15, 6, 10, 12, 67,10,29,22 , 'inprogess', 2 ],
  [ 1, '3J5FVCE', 25, 6, 3, 21, 92,16,32,43, 'inprogess' ,3],
  [ 1, '4J5FVCE', 35, 14, 8, 12,127,30,40,55 , 'inprogess',4 ],
  [ 1, '5J5FVCE', 4, 16, 8, 21,131,46,48,71 , 'inprogess',5 ],
  [ 1, '6J5FVCE', 12, 36, 8, 22, 143, 82,56,93, 'completed',6 ]
]
const allRegions = [
  '1J5FVCE',
  '2J5FVCE',
  '3J5FVCE',
  '4J5FVCE',
  '5J5FVCE',
  '6J5FVCE',
]
const app = express()
const pusher = new Pusher({
    appId: "1124234",
    key: "b691171de5f8ac605664",
    secret: "9b22dfa2b49d99cabeb2",
    cluster: "mt1",
    useTLS: true
  });
(async function(){
  const votes = await generateVotes()
  const spliced = votes.splice(0, 35)
  console.log(spliced[0], votes[4])
})()
var i = 0

setTimeout(() => {
  if(i == 5){
    i = 0
  }
  pusher.trigger("my-channel", "my-event", {
    message: sampleData[0]
}).then(console.log).catch(e=> console.log(e))
  i++
}, 50);

let beamsClient = new PushNotifications({
  instanceId: instanceid,
  secretKey: secretKey
});

beamsClient.publishToInterests(['hello'], {
   web: {
     notification: {
       title: 'Votes completed',
       body: 'The election has been completed and the winner is red,
     }
   }
 }).then((publishResponse) => {
   console.log('Just published:', publishResponse.publishId);
 }).catch((error) => {
   console.log('Error:', error);
 });
```

Running the App

Now, you can run your app. All you need to do is run `npm start` in the terminal to start the front end. To start the backend, run `node index.js`.

It's that simple and straightforward! Follow this tutorial closely and build your own live poll app using Pusher. You can take a look at the code repository here and check out the demo app.

Find out more about **delivering live results with Pusher** or read the full case study on **how The Washington Post broke realtime records** by using WebSockets with Pusher Channels to send 1 million messages per second to global news readers during the 2020 US Presidential Election.

Node.js React JavaScript

January 19, 2021

by Pusher team

Share article

Clone the project repository

Contents

PUSHER
A **MessageBird** company

Products

Channels

Beams

Developers

Docs

Tutorials

Status

Support

Sessions

Company

Contact Sales

Customer stories

Terms of Service

Security

Careers

Blog

Legal