



Mike

[Follow](#)

Ruby On Rails Developer, sometimes I say funny things.

Dec 2, 2017 · 4 min read

Real Time feature with Phoenix and Vue.js

Anytime that I need to add a real time feature for some project, my framework of choice is **Phoenix**, a rails like framework written in elixir.

For the front-end I'm going to use **Vue.js**, because is simple and gets the job done. You can use **React** or **Angular2** or even **Jquery** if you like.

Most of the phoenix channels tutorials around the web, put the front-end client inside the framework; for this tutorial in particular I'm going to build the client in a separated app, this is in case you already have your project with a different stack and just want to add real time as micro-service.

This is a very basic Phoenix tutorial, so topics about authentication, database operations are not covered in this article.

Generate a new Vue project using vue-cli

First we install **vue-cli**:

```
$ npm install --global vue-cli
```

Then we generate a new project called `realtime-example` :

```
$ vue init webpack realtime-example
```

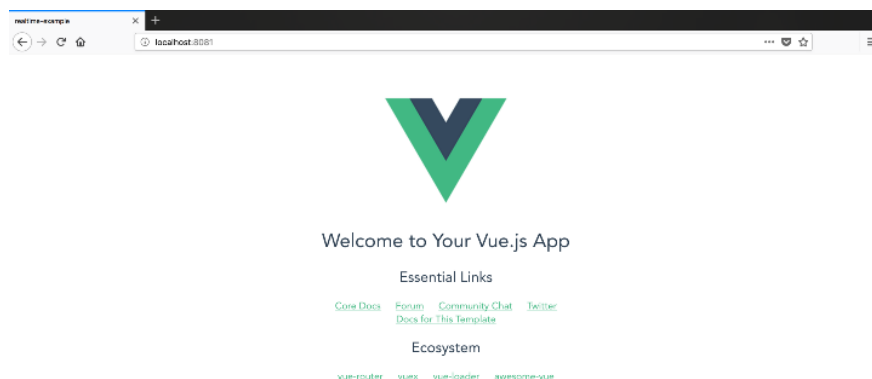
We go to the project folder and install the dependencies:

```
$ cd realtime-example && npm install
```

Now juts run the app to see if everything is working:

```
$ npm run dev
```

Open `http://localhost:8080/`, if you see the “Welcome screen”, you are ready(if you can't the welcome screen, try with `http://localhost:8081/`).



If you see this, you are very good following tutorials.

Adding components

This is the part when I write a bunch of code and you copy that in to your template.

We going to make a simple app that creates links and each link appears in list for all the connected users, so every client can see the new added links in real time(basically is a *Todo* app in disguise with real time functionality).

Now we open the `index.html` file in the root of the project and add a css framework, this ways things don't look that bad, and `animate.css` because animations are cool

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="utf-8">
5     <meta name="viewport" content="width=device-width,init
6     <title>Realtime Example</title>
7     <!-- Add Simple css framework -->
8     <link rel="stylesheet" type="text/css" href="https://c
9     <link rel="stylesheet" href="https://cdnjs.cloudflare.
10  </head>
```

Then we add a new file in inside the components folder:

```
src/components/NewLink.vue
```

this is our component responsible of the link creation. In this file we write the next code:

```

1   <template>
2     <div class="container">
3       <div class="form">
4         <div class="siimple-h3">Add a New Link </div>
5         <div>
6           <label class="siimple-label">Title </label>
7           <input v-model="form.title" type="text" class="si
8         </div>
9         <div>
10          <label class="siimple-label">URL</label>
11          <input v-model="form.url" type="text" class="siimp
12        </div>
13        <div class="save-button-container">
14          <div @click="addLink" class="siimple-btn siimple-b
15        </div>
16      </div>
17    </div>
18  </template>
19
20  <script>
21    export default {
22      name: 'NewLink',
23      data () {
24        return {
25          form : {
26            title: '',
27            url: ''
28          }
29        }
30      },
31      methods: {
32        addLink: function () {
33          let link = {
34            title: this.form.title,
35            url: this.form.url
36          }
37          // we gonna replace this console log with a socke
38          console.log(link)
39        }
40      }
41    }

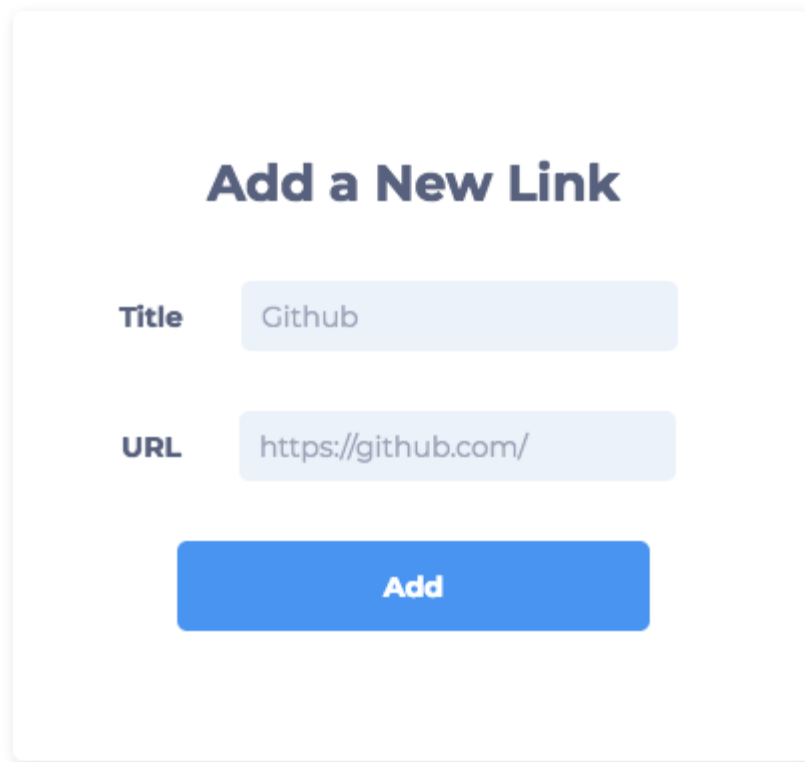
```

```
42     </script>
43
44     <!-- Add "scoped" attribute to limit CSS to this component
45     <style scoped>
46         .container {
47             display: flex;
48             flex-direction: column;
49             align-items: center;
50         }
51
52         .form {
```

Now we need to render this component in the root component, so we import the **NewLink** component and add the tag in the template section:

```
1   <template>
2     <div id="app">
3       <div class="siimple-h2">
4         {{ title }}
5       </div>
6       <div class="body">
7         <NewLink/>
8       </div>
9     </div>
10  </template>
11
12  <script>
13    import NewLink from '../src/components/NewLink'
14
15    export default {
16      name: 'app',
17      data () {
18        return {
19          title: 'Share links in realtime, because why not?'
20        }
21      },
22      components: {
23        NewLink,
24        Feed
25      }
26    },
```

If we check the browser now, the Link form should be on the left.



Add a New Link

Title Github

URL https://github.com/

Add

Phoenix Project

For the phoenix project we going to use a template with docker support, clone the repository in another folder, I recommend to install **Docker** and **Docker-Compose** if you do not have them installed.

Now we clone the repository:

```
$ git clone https://github.com/miguejs/phoenix-start-template
```

In order to get the project running, first we get inside the container with:

```
$ docker-compose run web bash
```

Once inside the container we run the following commands:

```
# Install dependencies
$ mix deps.get

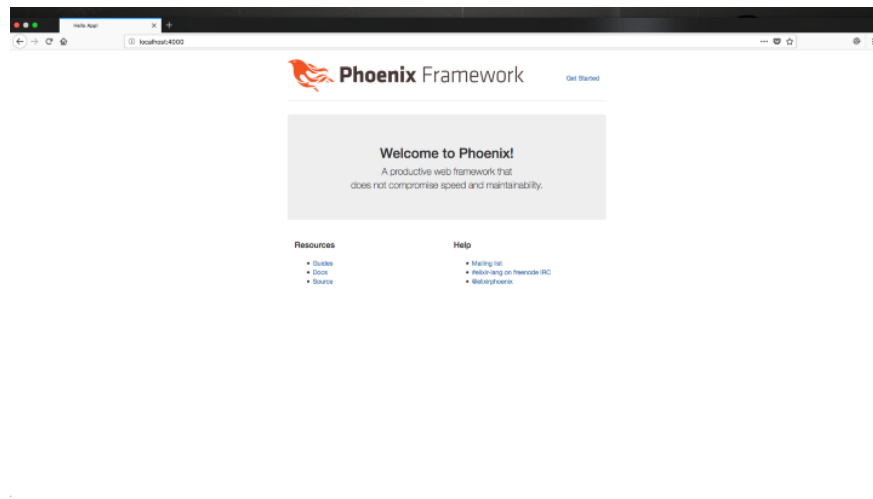
# Create and migrate your database
$ mix ecto.create && mix ecto.migrate
```

```
# Install Node.js dependencies
$ cd assets && npm install
```

Then we get out of the container with:

```
$ exit
```

Now we run the app with `docker-compose up web` and open `http://localhost:4000/` in your browser. If you see the Phoenix Welcome page, you are ready to go.



Create Our first channel

Now we going to edit `/lib/app_web/channels/user_socket.ex` , lets add channel below the “Channels” comment:

```
channel "links", AppWeb.LinksChannel
```

Then we add a new file called `links_channel.ex` inside the **channels** folder with the following content:


```
1  defmodule AppWeb.LinksChannel do
2    use Phoenix.Channel
3
4    def join("links", _payload, socket) do
5      {:ok, socket}
6    end
7
8    def handle_in("new_link", payload, socket) do
9      broadcast! socket, "new_link_added", payload
```

Basically every time we push a link, the channel will broadcast that link to all the clients.

Add Socket client to the client

In order to connect to the phoenix channels with our client, we going to use the **phoenix-socket** package, same plugin installed in the phoenix node dependencies:

Inside the client app root folder we run:

```
$ npm install --save phoenix-socket
```

And then we need to go back to our **NewLink** component and add the socket plugin in the script tag:

```
1  <script>
2  import { Socket} from 'phoenix-socket'
3
4  export default {
5    name: 'NewLink',
6    mounted() {
7      // Connect to the websocket server
8      let socket = new Socket("ws://localhost:4000/socket")
9      socket.connect();
10     // Join in the links channel
11     this.channel = socket.channel("links", {});
12     this.channel.join()
13     .receive("ok", resp => { console.log("NewLink Joined")
14     .receive("error", resp => { console.log("NewLink Una
15
16   },
17   data () {
18     return {
19       form : {
20         title: '',
21         url: ''
22       },
23     },
24   },
25 }
```

7- Add Feed component

We almost done, we need to add the Feed component, lets create a new file inside the components folder:

```
src/components/Feed.vue
```

with the following content:

```

1   <template>
2     <div class="container">
3       <div class="feed-container" id="feed">
4         <div class="siimple-h3">Feed</div>
5         <div v-for="link of links" :key="link.url" class="si
6           {{ link.title }}: <span class="siimple-link"> {{ l
7         </div>
8       </div>
9     </div>
10  </template>
11
12  <script>
13  import { Socket} from 'phoenix-socket'
14  export default {
15    name: 'Feed',
16    mounted() {
17      let socket = new Socket("ws://localhost:4000/socket"
18      socket.connect();
19
20      this.channel = socket.channel("links", {});
21
22      this.channel.join()
23      .receive("ok", resp => { console.log("Feed Joined su
24      .receive("error", resp => { console.log("Feed Unable
25
26      this.channel.on('new_link_added', payload => {
27        console.log("from Phoenix", payload)
28        this.links.push(payload.link)
29      });
30
31    },
32    data() {
33      return {
34        links: []
35      }
36    }
37  }
38  </script>
39
40  <!-- Add "scoped" attribute to limit CSS to this component
41  <style scoped>

```

```
| 42      .container {
```

And then we import the Feed component in the Root component:

```
1  <template>
2    <div id="app">
3      <div class="siimple-h2">
4        {{ title }}
5      </div>
6      <div class="body">
7        <NewLink/>
8        <Feed/>
9      </div>
10   </div>
11 </template>
12
13 <script>
14 import NewLink from '../src/components/NewLink'
15 import Feed from '../src/components/Feed'
16
17 export default {
18   name: 'app',
19   data () {
20     return {
21       title: 'Share links in realtime, because why not?'
22     }
23   },
24   components: {
25     NewLink,
26     Feed
27   },
```

And is done, we can see how the app works in the next video:

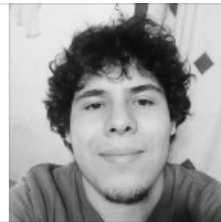
Vue.js in real time example



Here are the repository links from this tutorial, let me know in the comments if you have any problem with the tutorial or any mistake done in this article (I'm not a native English speaker, so any feedback in this department would be appreciated).

[miguejs/realtime-example](#)

realtime-example - Vue js Real time Example
[github.com](#)



[miguejs/phoenix-start-template](#)

phoenix-start-template - Phoenix template with
docker support
[github.com](#)



