

User Registration

In our previous lesson, we toured our example app's starting code and saw that any user off the internet was able to access our dashboard, which displays a list of events from our mock database. We only want users who have an account with our app to be able to access those events.

If you download the starting code in the Lesson Resources of this page, you'll see that our **server.js** file now only returns those events when the `/dashboard` endpoint is accessed by an API call that includes a token.

server.js

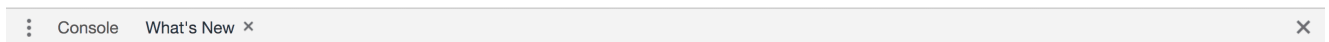
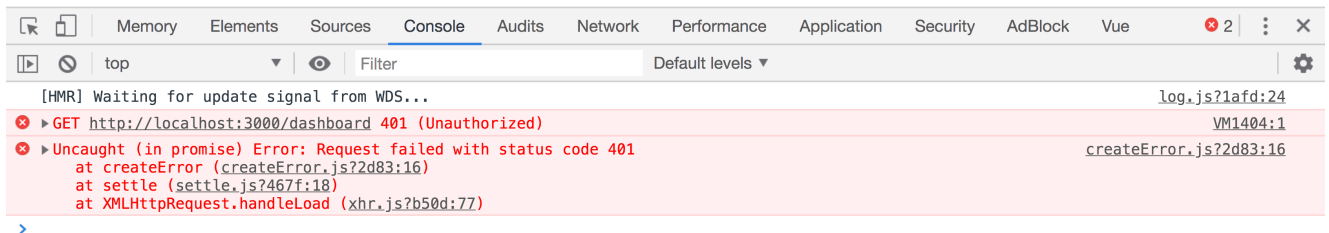
```
app.get('/dashboard', verifyToken, (req, res) => {  
  //verifyToken is middleware  
  jwt.verify(req.token, 'the_secret_key', err => { //  
    verifies token  
    if (err) { // if error, respond with 401 code  
      res.sendStatus(401)  
    } else { // otherwise, respond with private data  
      res.json({  
        events: events  
      })  
    }  
  })  
})
```

As a reminder, the **server.js** file is our mock authorization server running Express.js. We will run API calls against this server, and it is already configured for proper token authentication. If we run our app now (`npm run start`) and try to access the dashboard, we'll get a 401 error in our console.



Dashboard

Loading events



So we need to add to our app so that when a user is registered, they receive a token, which we can send along as we make a call out for that protected data. The `/dashboard` endpoint will then verify that token. If everything is good, it will respond with the private data we are requesting.

The steps we'll take in this lesson include:

1. Create a new **RegisterUser.vue** component
2. This will accept a name, email, and password from the user, and use Vuex to make a call out to our server's `/register` endpoint
3. Add to our Vuex store
4. We need an Action to register users and a Mutation that stores the response received from our server's `/register` endpoint and adds the received token to our Axios header
5. Add **RegisterUser** as a route

Let's get started.

Creating a RegisterUser component

So what does a component look like that registers users? You've seen a lot of them before in other apps. We're essentially building a small form that takes in a user's information, which we use to create their new account.

Name:



Email:

Password:



Register

[Already have an account? Login.](#)

So what kind of information are we gathering? We'll need a `name`, `email` and `password` from our new user, which means we'll need to add those as fields in our component's data, like so:

`src/views/RegisterUser.vue`

```

data () {
  return {
    name: '',
    email: '',
    password: ''
  }
},

```

Now in the component's template we can create a form and add an `input` for each of those fields and bind them to the data with `v-model`. While we're at it, we'll give each input a `label`, and add a button that will eventually trigger our submit behavior.

 **src/views/RegisterUser.vue**

```

<template>
  <div>
    <form>
      <label for="name">
        Name:
      </label>
      <input v-model="name" type="text" name="name"
value>

      <label for="email">
        Email:
      </label>
      <input v-model="email" type="email" name="email"
value>

      <label for="password">
        Password:
      </label>
      <input v-model="password" type="password" name
value>

      <button type="submit" name="button">
        Register
      </button>
    </form>
  </div>
</template>

```

Now our data fields will be populated as the user types into them. But what do we want to do with that user data we've collected? We need to send it as credentials to our backend, to

the `/register` endpoint, when a user clicks our button. Since the button is of type `submit` we need to add `@submit.prevent="register"` to the `form` element to prevent the default behavior. That `register` method name? Let's write that method now.

We need the `register` method to take the user's credentials and `dispatch` a Vuex Action (which we'll write in a minute).

 **src/views/RegisterUser.vue**

```
methods: {
  register () {
    this.$store
      .dispatch('register', {
        name: this.name,
        email: this.email,
        password: this.password
      })
  }
}
```

We are about to add the `register` Action to our Vuex store, but I wanted you to first see that we're taking an object with the `name`, `email` and `password` from our **RegisterUser** component, and passing it into the `register` Action as the payload. Now let's head into our Vuex store and write out that Action.

Setting up Vuex

Since our `register` Action will be making an API call, we'll need to import the Axios library into our **store.js** file. The Action will then look like this:

 **src/store.js**

```
import axios from 'axios'
...
actions: {
  register ({ commit }, credentials) {
    return axios
      .post('//localhost:3000/register', credentials)
      .then(({ data }) => {
        console.log('user data is', userData)
        commit('SET_USER_DATA', data)
      })
  }
}
```


```
}  
}
```

We're making an Axios `post` request to our server's `register` endpoint. While making this call, we're sending along the user's `credentials` (name, email, password), which we passed in as the payload from the **RegisterUser** component when we dispatched this Action.

Our dummy server's `register` endpoint will take those user credentials, add the user to our **user.json** file, then send us back a response object that includes a JWT `token`, along with the user's `name` and `email`. Since we are console.logging the `data` we get back, we can see that response in our browser's devtools. But first we need to add RegisterUser as a route so we can access it, so let's quickly do that now.

Adding RegisterUser to router.js

We'll just import and add as a route inside the **router.js** file.

 **src/router.js**

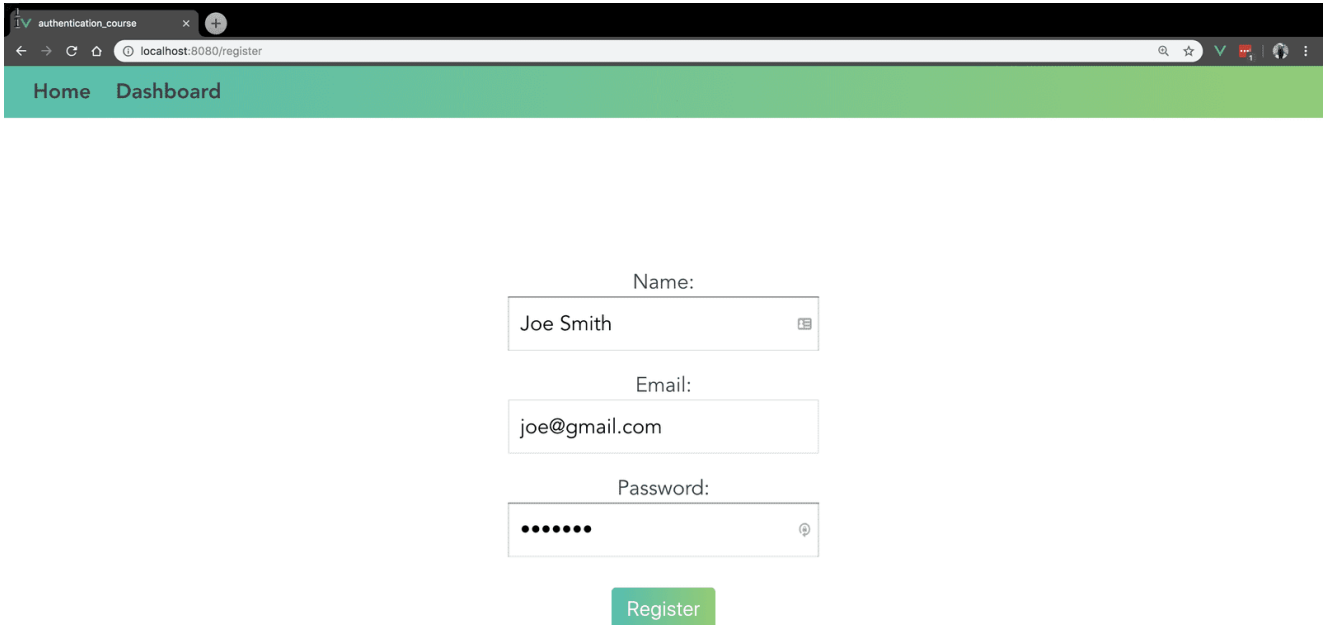
```
import Vue from 'vue'  
import Router from 'vue-router'  
import Home from './views/Home.vue'  
import Dashboard from './views/Dashboard.vue'  
import RegisterUser from './views/RegisterUser.vue'
```

```
Vue.use(Router)  
const router = new Router({  
  mode: 'history',  
  base: process.env.BASE_URL,  
  routes: [  
    {  
      path: '/',  
      name: 'home',  
      component: Home  
    },  
    {  
      path: '/dashboard',  
      name: 'dashboard',  
      component: Dashboard  
    },  
    {  
      path: '/register',  
      name: 'register',  
      component: RegisterUser  
    }  
  ]  
})
```

```
}  
]  
})
```

Storing the Server's Response

Now we can head to the app running in the browser, pull up the `/register` route, and create a new account.



authentication_course x

localhost:8080/register

Home Dashboard

Name:
Joe Smith

Email:
joe@gmail.com

Password:
.....

Register

When we click the register button, we'll see that our server's response was logged to the console:

```
user data is: store.js?c0d6:15  
{token: "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyIjp7IiwudXN0bG9iLWVudC00d6:15", email: "joe@gmail.com",  
  name: "Joe Smith"}  
  email: "joe@gmail.com"  
  name: "Joe Smith"  
  token: "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyIjp7IiwudXN0bG9iLWVudC00d6:15", email: "joe@gmail.com",  
  __proto__: Object
```

As we learned in the previous lesson, we then need to take that data and store it in our Vuex State, store a copy of it in the browser's local storage, and add the token into our Axios header. We'll accomplish those steps in a Mutation called `SET_USER_DATA`, which we're about to write.

But first, since that Mutation will be adding the user to our Vuex State, we need to add a `user` field to our State.

 **src/store.js**

```
state: {  
  user: null  
},
```

Now let's write the Mutation that will add it to the State, along with adding a copy to local storage.

 **src/store.js**

```
mutations: {  
  SET_USER_DATA (state, userData) {  
    state.user = userData  
    localStorage.setItem('user',  
JSON.stringify(userData))  
  },
```

Notice how we're using the `JSON.stringify` method to convert the JavaScript object into a JSON string.

Next, we need to add the token that lives in the `userData` into our Axios header, so that when we make API calls with Axios, our server will have the "key" (JWT token) to unlock our locked `/dashboard` endpoint.

 **src/store.js**

```
import axios from 'axios'  
...  
mutations: {  
  SET_USER_DATA (state, userData) {  
    state.user = userData  
    localStorage.setItem('user',  
JSON.stringify(userData))  
    axios.defaults.headers.common['Authorization'] =  
`Bearer ${  
      userData.token  
    }`  
  },
```

This code may look confusing at first, but we're simply setting the default `Authorization` header of our Axios instance so that it includes the JWT token. If you're wondering what `Bearer` means, it's just the *type* of Authentication being used. You can

read more about it [here](#). You just need to know that we're giving Axios the JWT token so the server can use it to unlock its protected `/dashboard` endpoint.

Great. Now we have `user` State, which our `SET_USER_DATA` populates when our server returns the `userData` in response to our API call. We've stored a copy of it in our local storage so we can reset our State in case of a browser refresh (we haven't yet implemented that reset code, but you can expect it in a future lesson), and we've given Axios the key it needs to unlock our server's dashboard route so we can access that private data: the events that live in our mock database.

Redirecting to Dashboard

We're about to check to see if this all works, but we need to consider where we send the user once they successfully register with our app. Currently, when they click the register button, they aren't taken anywhere. We should be redirecting them to the dashboard once they register successfully, so let's implement that now.

Back in our RegisterUser component, we'll add that redirect:

 **src/views/RegisterUser.vue**

```
methods: {
  register () {
    this.$store
      .dispatch('register', {
        name: this.name,
        email: this.email,
        password: this.password
      })
      .then(() => {
        this.$router.push({ name: 'dashboard' })
      })
  }
}
```

Now, once the user successfully registers, they will `then` be routed to the `dashboard`. Great! Now we're ready to check this out in the browser.

Checking the Browser

If we head into the browser, create an account, and hit the register button, we should now be getting back our private events.

Dashboard

@12:00 on Feb 22, 2022

Puppy Parade

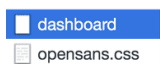
@9:00 on March 4, 2022

Cat Cabaret

@1:00 on June 12, 2022

Doggy Day

It works! If we pop open the devtools and inspect the network tab, we can see that the register call was made, as well as a call to the dashboard, where we can indeed see our token was added in the Authorization header.



Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyIjp7Im5hbWUiOiJLYXRlIE1hcnciLCJlbWFPbCI6ImthdGVAZ21haWwY29tIiwicGFzc3dvcmQiOiJwYXNzMtIzIn0sImVudCI6MTU1NDQ5NTUwMH0._y7Kbndh3RkIrvfulygCnq42c960TQptVozQYE0Fkkg

Remember when we covered the structure of a JWT in the first lesson? Notice the three parts of that JWT token, each separated by a period.

Awesome. We did it!

What's next?

Now that we've started registering users, in the next lesson we'll look at logging a user back into the account they've already created. As we progress through the course we'll also handle what to do when a user tries to access the dashboard route when they aren't logged in.

[Download Video](#)

[Share Lesson](#)



Lesson Resources

- Starting Code

- Ending Code