Java API with Vue.js UI using JWT authentication

Quick Start

This guide is not meant to be a “cut-and-paste” from this document to get your project working. Instead it points out locations in the code to look for examples, while providing some extra tips. Do not cut-and-paste anything from this document, instead find it in the working example project and use the code demonstrated there.

It is highly recommended that you read through this document before starting as it contains some tips and details not available in the projects.

Setup

Before running the example projects you will need to run the schema.sql and data.sql files to add the User table and starting data to the database you intend to use. Once that is complete, you will need to update the DataSource in the springmvc-servlet.xml with the correct database name and credentials for your database.

The Vue.js UI Project

Creating the Vue Project

Select “Manually select features”
Features:

- Bable
- Router
- Linter / Formatter
- Unit Testing
- E2E Testing

Use history mode for router: Y
Pick a linter / formatter config: ESLint with error prevention only
Pick additional lint features: Lint on save
Pick a unit testing solution: Mocha + Chai
Pick a E2E testing solution: Cypress
Where do you prefer placing config for… : In dedicated config files

Copy auth.js into /src

Create a file named .env in the root of the project

- Same directory where package.json is located.

- All keys in the.env file must start with VUE_APP_
- Add a key for your Backend API url
    - VUE_APP_API_URL=http://localhost:8080/api
- You can then access these anywhere in your Vue app's JavaScript using:
    - `process.env.<KEY>`
        - Example: `process.env.VUE_APP_API_URL`
- Anytime you change this file you will need to stop the Vue serve (CTRL-C) and restart it (npm run serve)

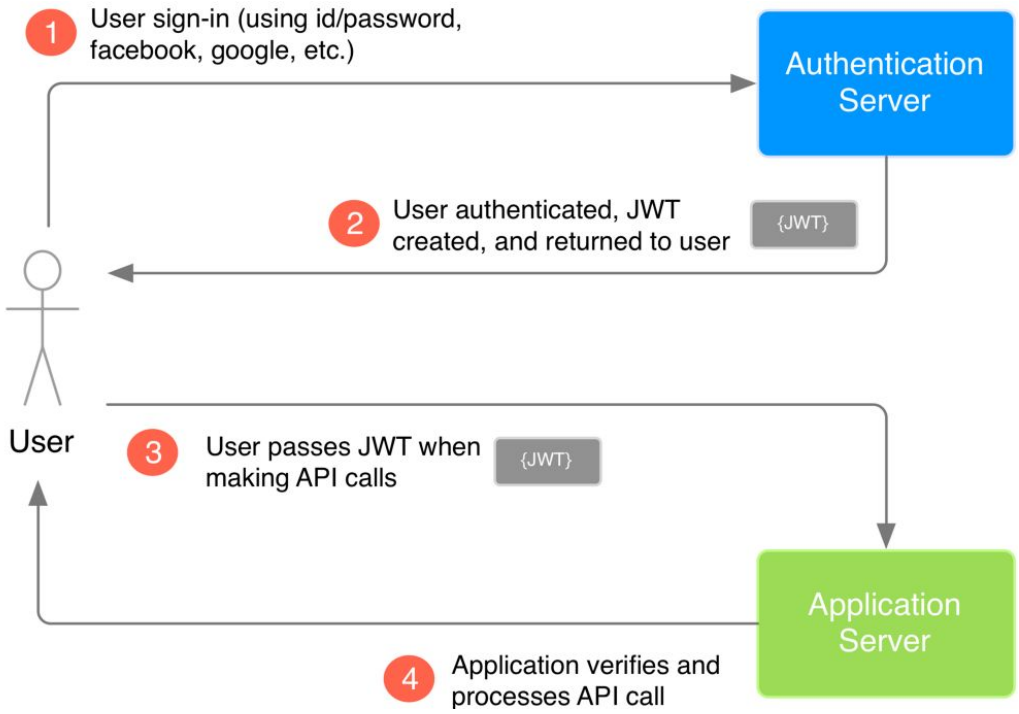
The Project

Since to have multiple pages you need to use the Vue Router, the steps and file locations are different than the full page components built in class.

/components	Still contain .vue files, but these should be smaller components that are added to full pages.
/views	Also contain .vue files, but these represent actual pages in the site.
auth.js	This is the JavaScript for the JWT authentication and User Management. You can add this to any Vue View or Component by including the following in the imports: `import auth from './auth';`
router.js	Every View will need to be configured in this JS file to allow it to be accessed. This file will also be used to determine who can access each page.

JWT Authentication

JWT (JSON Web Token) authentication works by the server creating a Token that is returned to the client at login. The token can include information about the user, such as their username and role. When the client makes future API calls to the server it passes the JWT Token, which the Server verifies was created by it. This allows for an application to work with the server in a stateless, so no need to store the user information in Session.



The provided auth.js will create a User Object from the JWT Token that you can use to do Authorization in your application.

The User Object

{iat: 1554674683, sub: "Doug", rol: "user", exp: 1554696283}

auth.getUser().rol ← Users Role
auth.getUser().sub ← Username

The auth.js can be added to any Vue component or view by importing: `import auth from './auth';`
Where the path, ./auth, is updated to be the relative path between your component and the auth.js.

Once imported the auth Object has the following methods:

.getUser() ← returns the user object
.logout() ← destroys the JWT token and User Object, effectively logging the user out of the application

An example of using the User object to only show parts of the page if the user is an admin can be found in:

1. About.vue
2. App.vue

User Login

The Login.vue component has a functional example of a Login.

User Logout

The App.vue component has an example of a logout method

User Registration

The Register.vue component has a functional example of a simple User Registration

Using the JWT Token

The JWT Token must be sent with each fetch() request against an API that requires that the user is Authenticated. This is done by adding a configuration object to the fetch, similar as what was used to set JSON for POST and PUT requests. The configuration object must set the Authorization header.

```
headers: new Headers({
    Authorization: 'Bearer ' + auth.getToken(),
  }),
```

And it must include a directive that tells the server what login process created the JWT.

```
credentials: 'same-origin',
```

same-origin directs the server that it created the token.

A working example of a Fetch() with the Token can be found in the About.vue

Routing

Every new page you add to your Vue application must be added to the route.js

For example, if you had a component in /src/viws/NewPage.vue that you wanted to add routing for:

1. Import the component

```
import NewPage from '@/views/NewPage;
```

2. Add an object with where that matches the path from the URL (similar to a RequestMapping) to the component to display:

```
{
    path: '/newPage',
    name: 'newPage',
    component: NewPage,
  }
```

For this route if your app was located at http://localhost:8081, then if the user went to the URL http://localhost:8081/newPage then the component NewPage would be displayed.

In a routed Vue application, pages and components cannot be displayed directly, instead they must have a valid route in the route.js to be reached.

There example application route.js has a router lifecycle hook, beforeEach, that you will want to copy into your route.js to enable Authorization using the JWT token. When doing so, note that when the route is created it is assigned to a variable, router, and then the export default router; is moved to after the beforeEach.

The beforeEach method has comments explaining what each line is doing.

The router can do a lot more, and while this will get you started, you should explore some of its functionality to see what it can do to help you build your site.

More Information about the Router can be found in this video:
https://www.youtube.com/watch?v=Bq2MCYWDndc

Or in the documentation: https://router.vuejs.org/

The Java API Project

com.techelevator.authentication

You will need to copy the entire com.techelevator.authentication package and all its classes into your project.

You shouldn't need to modify them, but in case you do, here is what each does:

AuthProvider	Interface for a class that gives details about the user making the request, such as their role. You can @Autowire it into any Controller There is an example of using it in the ApiController class
JwtAuthInterceptor	Intercepts every request to an API to verify the Token. It returns an unauthorized error if the token is missing or incorrect, and allows access if

	the token is correct
JwtTokenHandler	Helper class to create the Token and parse it into a user object
PasswordHasher	Provides Password Hashing and Salting
RequestAuthProvider	The implementation of the AuthProvider that will be injected anywhere the AuthProvider interface is Autowired
UnauthorizedException and UserCreationExceptions	Custom exceptions used by the other classes in this package

com.techelevator.model

Model classes for the User object. You shouldn't need to change these.
The table can be added to any database and is defined in /database/schema.sql

com.techelevator.controller

AccountController	Provides User Login and registration operations. You can copy this class to your project
ApiController	Provides 2 sample API methods. The first shows how to use the AuthProvider to verify what role a user has. The second is used in the JWT token example on the Vue About page. Notice that in both cases there is nothing done with the JWT Token in the controller classes.

/WEB-INF/springmvc-servlet.xml

There is two new sections you will need to copy into your project:

```
<mvc:interceptors>
      <mvc:interceptor>
            <mvc:mapping path="/**"/>
            <bean class="com.techelevator.authentication.JwtAuthInterceptor">
                  <property name="excludedUrls">
                        <list>
                        <!-- Every url in the app must be authenticated
                              except /login -->
                              <value>/login</value>
```

```
                                <value>/register</value>
                            </list>
                        </property>
                    </bean>
                </mvc:interceptor>
            </mvc:interceptors>
```

This section configures the JwtAuthInterceptor and makes it so any requests to the server will first go through it, except for the paths listed in the excludedUrls list. If there are other paths that should be available to the public, meaning that they should not require authorization, then the paths will need to be added the list. It is currently to allow access to /login and /register to the public.

```
        <mvc:cors>
            <mvc:mapping path="/**" allowed-methods="*"/>
        </mvc:cors>
```

Sets a site-wide CORS policy. However, you still must add @CrossOrigin to your controllers.

/WEB-INF/web.xml

The session should be set to only be tracked via cookies, which will disable URL-Rewriting, where it sends the Session Token in the URL.

```
  <session-config>
    <tracking-mode>COOKIE</tracking-mode>
 </session-config>
```