Axios toolkit for WordPress REST API

As I'm learning the deep workings of the WordPress REST API I'm also learning how to use Axios as a replacement for Fetch.

While I discovered that this will not work with Nuxt and the WordPress REST API, it's still a good starting point if you want to work with vanilla JS and templating engines. I am still researching how to make it work in Nuxt.js.

Another thing that's important to note. Other than `getJWT()` all other functions require authentication and I've chosen to use JSON Web Tokens (JWT) to authenticate the requests. The WordPress side uses the JWT Authentication for the WP REST API so I don't havee to code it myself.

We first import the Axios library, create a constant holding our JWT token and set the default base URL for all Axios calls.

We'll discuss how we generate it and why we're getting from session storage when we talk about the `getJWT()` function.

```
import axios from "axios";

const access_token = sessionStorage.getItem('jwt_token');
axios.defaults.baseURL = 'http://rivendellweb.local';
```

Rather than use the default the password for the administrator account that is making the requests, I strongly suggest you use application passwords as an additional level of security without having to create accounts specific to a project.

We first run a post request to the JWT-Auth plugin's `token` endpoint with the user's credentilas in the body of the request. One thing that tripped me is that because we're getting the token for the first time ***we don't need to pass the JWT token to this method***.

If the request is successful we get back a JWT token that we store in session storage. I chose session storage because it lasts only as long as the current session.

```
async function getJWT() {
  await axios
    .post('/wp-json/jwt-auth/v1/token', {
      username: 'username',
      password: 'password',
    })
    .then((res) => {
      if (sessionStorage.getItem('jwt_token') === null) {
        sessionStorage.setItem('jwt_token', res.data.token);
      } else {
        sessionStorage.removeItem('jwt_token');
        sessionStorage.setItem('jwt_token');
      }
    })
    .catch((err) => {
      console.error(err);
    });
}
```

`getRoot()` returns the root of the site's API. I've added this function for completeness sake. Most of the time we'll skip this and look at `getIndex()` instead.

```
async function getRoot() {
  await axios
    .get('/wp-json/', {
      headers: {
        Authorization: `Bearer ${access_token}`,
      },
    })
    .then((res) => {
      console.log(res.data);
    })
    .catch((err) => {
      console.error(err);
    });
}
```

`getIndex()` queries the basse of the WordPress routes. This is where all the routes we'll discuss below originate from.

This is more specific than `getRoot` as it pertains only to WordPress routes and we can use it to explore what routes the specific instance of WordPress makes available.

```
async function getIndex() {
  await axios
    .get('/wp-json/wp/v2/', {
      headers: {
        Authorization: `Bearer ${access_token}`,
      },
    })
    .then((res) => {
      console.log(res.data);
    })
    .catch((err) => {
      console.error(err);
    });
}
```

We use `getPageCount()` to get the number of pages of posts available on the blog.

So the idea is to capture the value of `X-WP-TotalPages` response header via the HEAD HTTP method and use it to build the pagination.

> The HTTP HEAD method requests the headers that would be returned if the HEAD request's URL was instead requested with the HTTP GET method.
>
> HEAD from MDN

In this example we capture the headers we need and then use them to log a message to the console. In a production application we would use them to generate the individual links to all available pages.

```
async function getPageCount() {
  await axios
    .head('/wp-json/wp/v2/posts/', {
      headers: {
        Authorization: `Bearer ${access_token}`,
      },
    })
    .then((res) => {
      // console.log(res.headers);
      let totalPosts = res.headers['x-wp-total'];
      let totalPages = res.headers['x-wp-totalpages'];

      console.log(
        `there are ${totalPosts} posts divided in ${totalPages} pages`
      );
    })
    .catch((err) => {
      console.error(e'x-wp-total'
}
`there are ${totalPosts} posts divided in ${totalPages} pages`
```

To query the type of resources available, use `getTypes()`. This will provide a list of all types of content available through the REST API, including custom post types

```
async function getTypes() {
  await axios
    .get('/wp-json/wp/v2/types/', {
      headers: {
        Authorization: `Bearer ${access_token}`,
      },
    })
    .then((res) => {
      console.log(res.data);
    })
    .catch((err) => {
      console.log(err);
    });
```

```
}
```

If we know the name of the type of content we're looking for, we can get more information about by using `getTypeBySlug`. This query will return a single type matching the slug we pass as a parameter or the post type if no parameter is passed to the function.

```
async function getTypeBySlug(slugName = 'post') {
  let singleType = `/wp-json/wp/v2/types/${slugName}`;

  await axios
    .get(singleType, {
      headers: {
        Authorization: `Bearer ${access_token}`,
      },
    })
    .then((res) => {
      console.log(res.data);
    })
    .catch((err) => {
      console.log(err);
    });
}
```

Unless we choose to work with a non-default type of content, posts are our main tool.

This will return a page of x number of posts. The defaults for the function are the same defaults that WordPress provides.

```
async function getPosts(numberOfPosts = 10, pageNumber = 1) {
  await axios
    .get('/wp-json/wp/v2/posts/', {
      params: {
        per_page: numberOfPosts,
        page: pageNumber,
      },
```

```
      headers: {
        Authorization: `Bearer ${access_token}`,
      },
    })
    .then((res) => {
      console.log(res.data);
    })
    .catch((err) => {
      console.log(err);
    });
}
```

The next function will get a single post by it's ID or return a default post if there are no parameters in the function call.

```
async function getPostByID(postID = 790634) {
  await axios
    .get('/wp-json/wp/v2/posts/', {
      params: {
        id: postID,
      },
      headers: {
        Authorization: `Bearer ${access_token}`,
      },
    })
    .then((res) => {
      console.log(res.data);
    })
    .catch((err) => {
      console.log(err);
    });
}
```

`getPostBySlug()` will return a single post based on its slug, the string we use in the URL. We also return a default value if there was no parameter passed to the function.

```
async function getPostBySlug(slugName = 'post-from-rest-api-2') {
  await axios
    .get('/wp-json/wp/v2/posts/', {
      params: {
        slug: slugN'/wp-json/wp/v2/posts/'ders: {
        Authorization: `Bearer ${access_token}`,
      },
    })
    .then((res) => {
      console.log(res.data);
    })
    .catch((err) => {
      console.log(err);
    });
}
```

Pages are the second most used content type in WordPress. This function will return the latest pages (in terms of creation) from the site.

```
async function getPages(numberOfPages = 10) {
  await axios
    .get('/wp-json/wp/v2/pages/', {
      params: {
        per_page: numberOfPages,
      },
      headers: {
        Authorization: `Bearer ${access_token}`,
      },
    })
    .then((res) => {
      console.log(res.data);
    })
    .catch((err) => {
      console.log(err);
    });
}
```

We can add queries for a single page by ID and by slug, like we did for posts,

leaving the process as an exercise for the reader.

There are other things we can do and will be covered in later posts:

- We can create functions to work with custom post types created for the site
- We can create an editor and within the editor
    - We can create a function that will update an existing post
    - We can create a function that will create a brand new post