Github Pages and authentication, we are not that far

Created on Thursday, December 12, 2019 by Romain Manni-Bucau

Github Pages are a great tool to publish quickly a website but it has some (https://twitter.com/intent/tweet) pitfalls. Today, if you have a private repository and enable Github Pages, you will publish publicly your website whereas you could naturally expect it to stay private as the repository. In any httpd server we would just add a .htaccess or use a solution close to that but here we don't have a real way to do that on the server.

However don't abandon yet, with a little effort you can make it possible.

Long story short, the trick is to generate the website twice:

1. Once for public Github Pages
2. Once for **private** Github Pages

The private one is not handled by Github, it is just a workaround we'll use to be able to add authentication and the public one is not really your website but a limited copy of it.

If we push a **private** website on github, we likely want to control the authentication through Github settings. A trivial way to do that is to go through Github API which has this feature built-in.

Concretely, our public pages will just contain a login form which will fetch the private page through Github repository API and replace its own content with the private content which is the actual content you want to share.

Prepare the public website

There are several ways to do the replacement, either you replace all the page, just the body, just a div etc...

To illustrate this post, you can consider you just replace the content of the HTML *body* and all the public pages will contain the form to log in with Github authentication:

```
<div id="loginForm"> <!-- 1 -->
    <span>Enter your github credentials:</span>
    <form onsubmit="event.preventDefault(); onSubmit(this);">
        <input type="text" id="login" name="login" placeholder="Github username" required>
        <input type="password" id="password" name="login" placeholder="Github password/token"
required>
        <input type="submit" value="Access">
    </form>
</div>
<script>
window.page = '${page}'; // 2
</script>
<script src="js/github.js"></script> <!-- 3 -->
```

1. A standard HTML form to let the user enter its username and password (or Github token if you are using 2FA),
2. A placeholder we'll replace on the fly by the actual page url (for instance *index.html* or *getting-started.html*),
3. A generic github script which will call the Github API and replace the content of the current page by the private content.

To make it working here are the steps you must follow:

1. Generate the normal site, push it to *gh-pages-private* branch,
2. Generate the public site using the template we just explained (the login form), this generation just visits the private site and does three things:
    1. It copies all needed assets (note that if you have private assets as images, you can use the API to fetch them too but it would make this post too long), generally it is the site theme which can be shared by both generations,
    2. For each html page, it creates a new version replacing its content by the login form. Depending your generation engine, it is either the same site with a different template or a real visitor with a replace (personally I code my generator in Java so visiting the private website and creating the new website replacing the content of the visited pages is quite trivial)
    3. Add in assets the *github.js* file
3. Finally both branches are pushed upstream and github will automatically update the public site

Javascript magic to show the private content

The first thing to ensure you identify is the organisation/account, repository and branch you will query:

```
const org = 'rmannibucau';
const repo = 'demo';
const branch = 'gh-pages-private';
```

Then we just need to implement the form login callback - *onSubmit* - we set in the template:

```
function onSubmit(form) {
  // 1
  const login = form.username || form.querySelector('#login').value;
  const password = form.token || form.querySelector('#password').value;

  // 2
  const token = btoa(`${login}:${password}`);
  const request = new Request(
    `https://api.github.com/repos/${org}/${repo}/contents/${page}?ref=${branch}`,
    {
      method: 'GET',
      credentials: 'omit',
      headers: {
        Accept: 'application/json',
        Authorization: `Basic ${token}`
      },
    });

  // 3
  fetch(request)
    .then(function (response) {
      if (response.status !== 200) { // 4
        document.querySelector('#loginForm').innerHTML = `Failed to load document (status: 
${response.status})`;
      } else {
        response.json()
          .then(function (json) { // 5
            const content = json.encoding === 'base64' ? atob(json.content) : json.content;

            // 6
            const startIdx = content.indexOf('<body');
            document.body.innerHTML = content.substring(
                content.indexOf('>', startIdx) + 1,
                content.indexOf('</body>'));
          });
      }
    });

    return false;
}
```

1. We extract from the form the username and password,
2. We create the github request ensuring to force the credentials to be the ones the user entered,
3. We finally call the API to retrieve the content,
4. If the status is not 200 we show an error message,
5. If the status is 200 we can read the content as a JSON then decode it (it is in base64 normally),
6. And finally we can replace our body content by the page we just retrieved content

At this point you should be able to use your Github credentials to access the public website which has no sensitive content.

Going further

There are a lot of improvements you can do to that logic and, as already mentionned, you can even generalize this trick to assets.

However, a small trick which can be nice is to store in the local storage the credentials. In a real implementation it can need to cipher it but to share the overall logic, just add after step 6 of previous snippet the following code:

```
localStorage.setItem('githubPagesAuth', JSON.stringify({ username: login, token: password
 }));
```

And after the *onSubmit* function, in the global javascript part:

```
const existingAuth = JSON.parse(localStorage.getItem('githubPagesAuth'));
if (existingAuth) {
    onSubmit(existingAuth);
}
```

What it does is to trigger a fake authentication if the credentials are already present in the *localStorage*, in other words, no more need to log in on each page to see the content, it is now done automatically, as soon as you managed to authenticate once.

Conclusion

This is just a few tricks to deploy a private website on Github Pages but this works quite well and allows to not find yet another HTTP server just to limit some page access to friends or colleagues.

Of course, this kind of logic can be customized a lot and for a blog, it can even be used to make not yet published pages available only to the people able to access the blog sources (the author generally) which creates a kind of staging website on Github Pages.

Home
(/)

Back to Other
(/category/other/53)