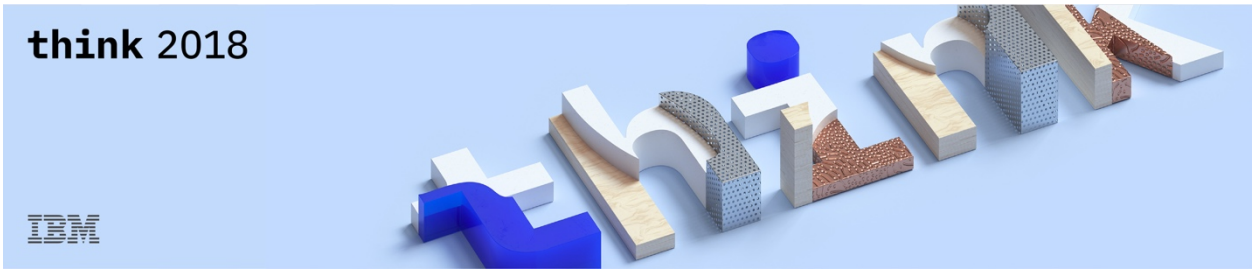


Lab Center – Hands-on Lab

Session 2473

Session Title IBM Connections Customizer : from Zero to Hero

Stefano Pogliani, IBM, stefano.pogliani@fr.ibm.com

Martti Garden, IBM, martti.garden@de.ibm.com

Padraic Edwards, IBM, padraic.edwards@ie.ibm.com

Martin Donnelly, IBM, marti_donnelly@ie.ibm.com

Table of Contents

Disclaimer

IBM's statements regarding its plans, directions, and intent are subject to change or withdrawal without notice at IBM's sole discretion. Information regarding potential future products is intended to outline our general product direction and it should not be relied on in making a purchasing decision.

The information mentioned regarding potential future products is not a commitment, promise, or legal obligation to deliver any material, code or functionality. Information about potential future products may not be incorporated into any contract.

The development, release, and timing of any future features or functionality described for our products remains at our sole discretion I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve results like those stated here.

Information in these presentations (including information relating to products that have not yet been announced by IBM) has been reviewed for accuracy as of the date of initial publication and could include unintentional technical or typographical errors. IBM shall have no responsibility to update this information. **This document is distributed "as is" without any warranty, either express or implied. In no event, shall IBM be liable for any damage arising from the use of this information, including but not limited to, loss of data, business interruption, loss of profit or loss of opportunity.** IBM products and services are warranted per the terms and conditions of the agreements under which they are provided.

IBM products are manufactured from new parts or new and used parts.
In some cases, a product may not be new and may have been previously installed. Regardless, our warranty terms apply."

Any statements regarding IBM's future direction, intent or product plans are subject to change or withdrawal without notice.

Performance data contained herein was generally obtained in controlled, isolated environments. Customer examples are presented as illustrations of how those customers have used IBM products and the results they may have achieved. Actual performance, cost, savings or other results in other operating environments may vary.

References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business.

Workshops, sessions and associated materials may have been prepared by independent session speakers, and do not necessarily reflect the views of IBM. All materials and discussions are provided for informational purposes only, and are neither intended to, nor shall constitute legal or other guidance or advice to any individual participant or their specific situation.

It is the customer's responsibility to insure its own compliance with legal requirements and to obtain advice of competent legal counsel as to the identification and interpretation of any relevant laws and regulatory requirements that may affect the customer's business and any actions the customer may

need to take to comply with such laws. IBM does not provide legal advice or represent or warrant that its services or products will ensure that the customer follows any law.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products about this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products. IBM does not warrant the quality of any third-party products, or the ability of any such third-party products to interoperate with IBM's products. **IBM expressly disclaims all warranties, expressed or implied, including but not limited to, the implied warranties of merchantability and fitness for a purpose.**

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents, copyrights, trademarks or other intellectual property right.

IBM, the IBM logo, ibm.com and [names of other referenced IBM products and services used in the presentation] are trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at: www.ibm.com/legal/copytrade.shtml.

U.S. Government Users Restricted Rights — use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM.

Introduction

This workshop was prepared as step by step instruction. It should lead you through IBM Connections Customizer components and its capabilities. All included exercises are going to support you in creating your first IBM Connections Customizer scripts.

Information

Information about user and password, method of access etc are going to be provided by your teacher.

All exercises are fulfilled with screen shots for easier navigation. However, there may be some small differences based on your environment, browser version, language or user account used.

Icons

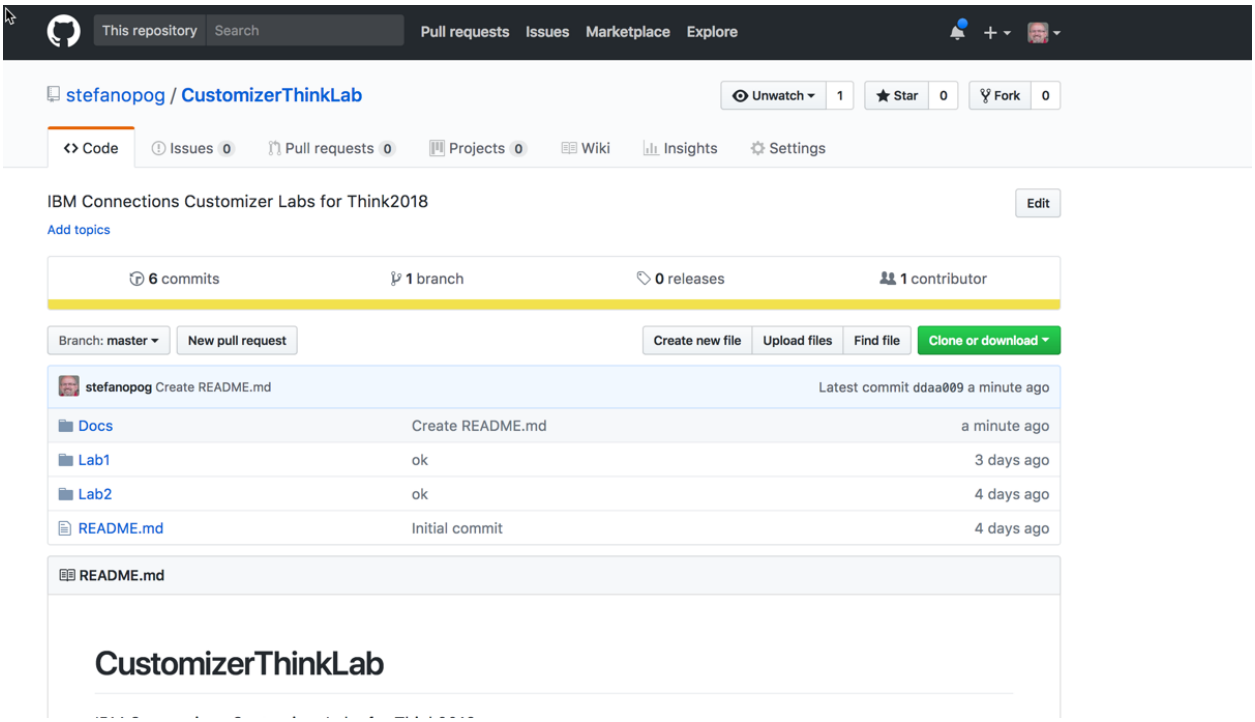
The following symbols appear in this document at places where additional guidance is available.

Icon	Purpose	Explanation
	Important!	This symbol calls attention to a particular step or command. For example, it might alert you to type a command carefully because it is case sensitive.
	Information	This symbol indicates information that might not be necessary to complete a step but is helpful or good to know.
	Trouble-shooting	This symbol indicates that you can fix a specific problem by completing the associated troubleshooting information.

Code, Solutions, Examples

A special Github repository has been created where solutions, examples and the code or the graphics to be used are made available to all the students. We will reference this Github repository throughout this document as "**Reference Repository**".

This Github repository is available at this address:
https://github.com/stefanopog/CustomizerThinkLab



You may want to open a tab of your browser on this address in order to use the material. Feel free to clone this repository for better analyzing the code and the examples.

Documentation

The text of these exercises, together with other useful documentation, is stored in the Docs folder of the Github repository mentioned above.

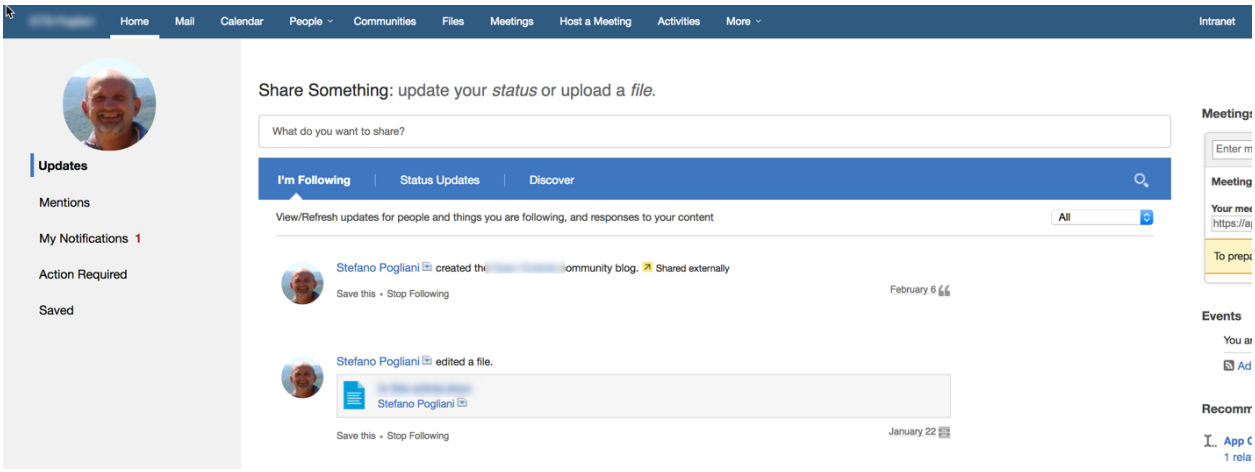
Lab 1 The typical “Hello World” script

This is a very simple exercise, very similar in the approach to the typical “Hello World” exercises you have certainly encountered in other labs.

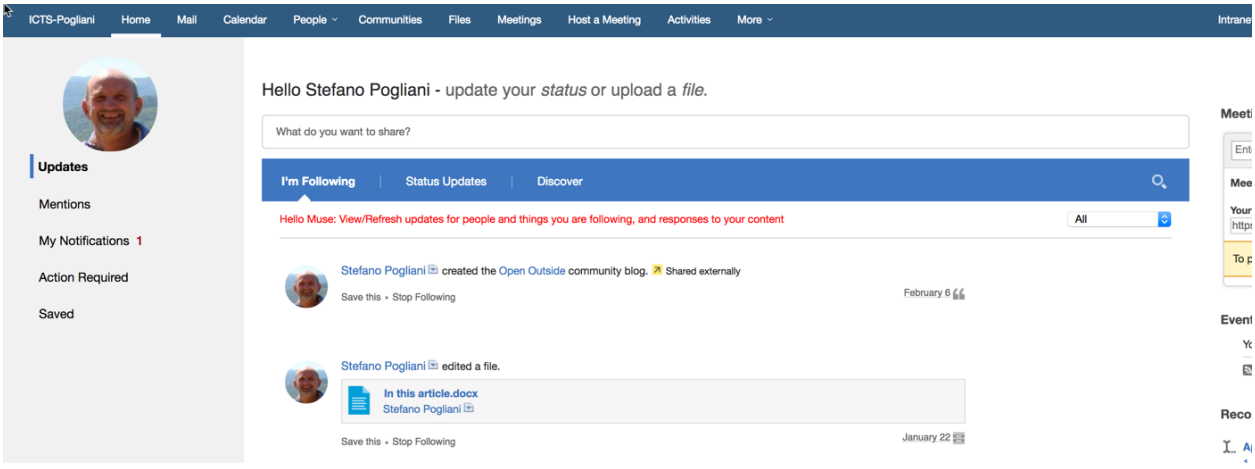
The main goal we want to reach through this exercise is to guide you through the process of creating an IBM Connections Customizer extension; the extension itself is very simple because the focus will be on the process.

What we want to achieve

When accessing the traditional IBM Connections Cloud Homepage, you are likely to see something like this:



In our exercise, we will modify the look-and-feel of this page to be the following one:



So, we simply changed the “Share Something” string with the greeting “Hello Stefano Pogliani” and we changed the color of the font for the string below the blue header of the Activity Stream (and added a little text to it).

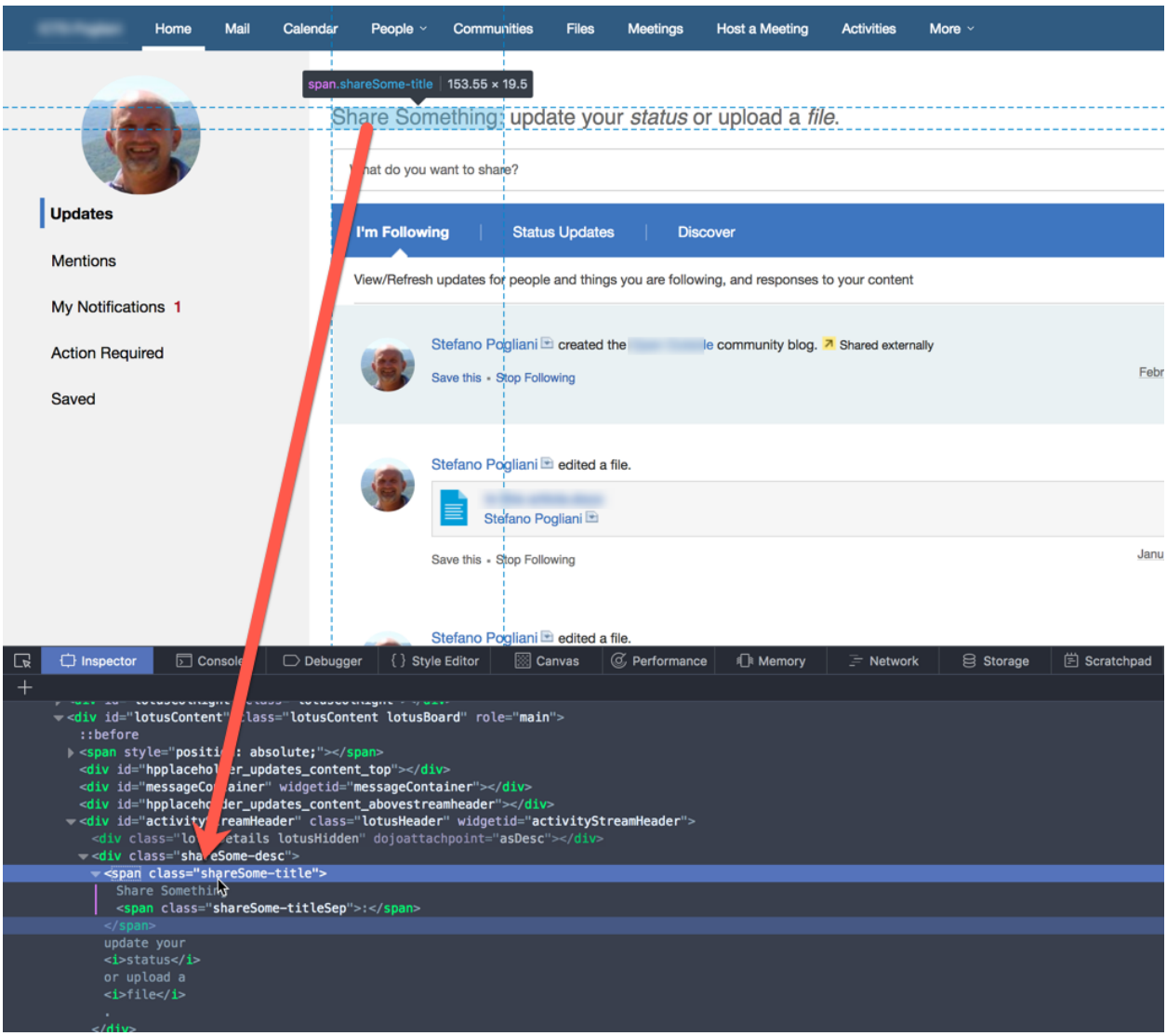
Preparation Activities

In order to do those modifications, we first need to understand which DOM elements we need to modify.

Our best friends here are the “Developer Tools” you find in your browser.

The Greeting

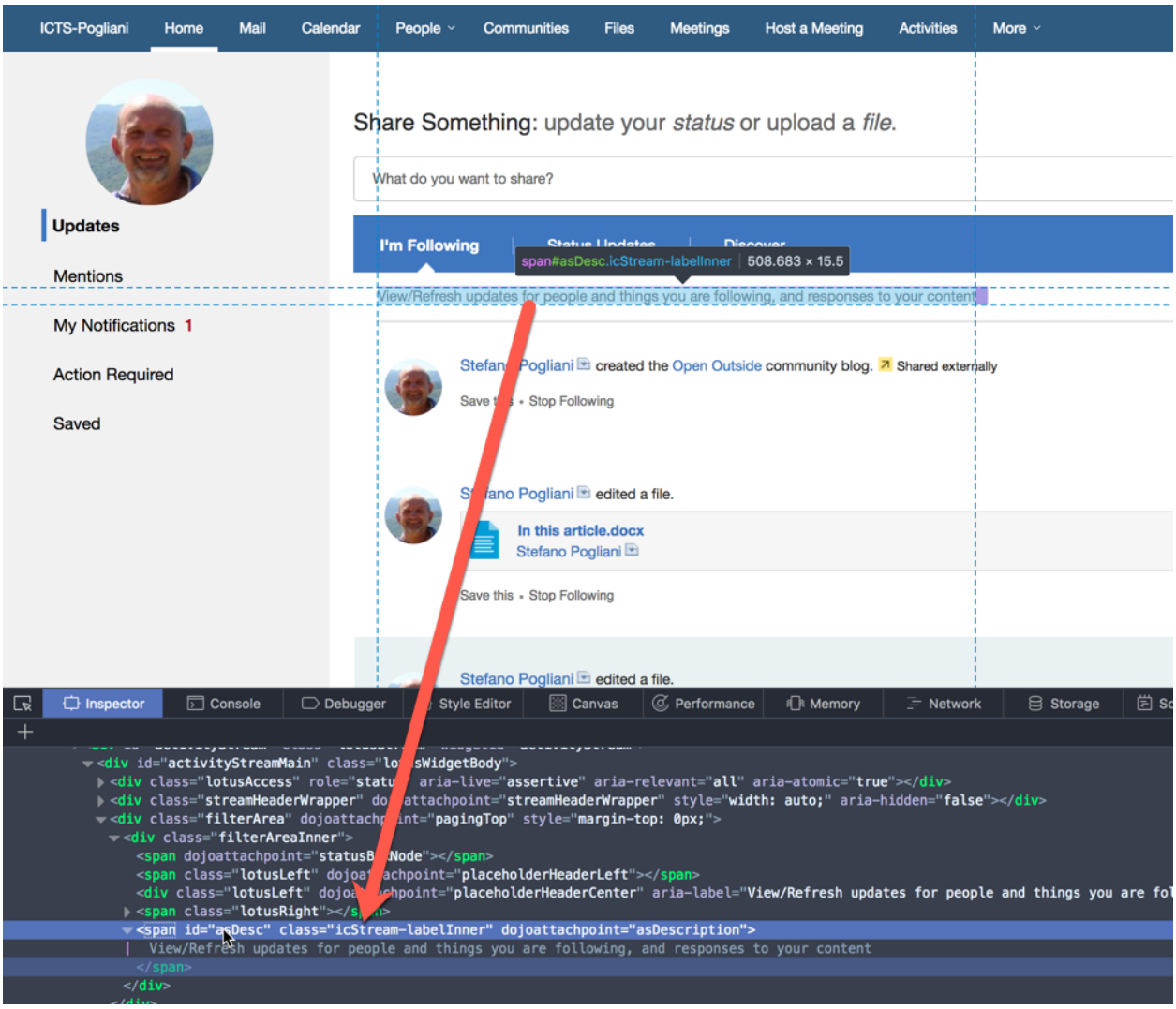
Which is the DOM element in the page holding the string “Share Something” ?



Using the Browser Developer Tools, we quickly find that the “*Share Something*” string is the content of a **** element whose class is “**shareSome-title**”.

The Description

The description string below the Blue Bar on the top of the Activity Stream is easily found also:



Using the Browser Developer Tools, we quickly find that the string is the content of a **** element whose id is "**asDesc**".

IBM Connections page loading mechanism

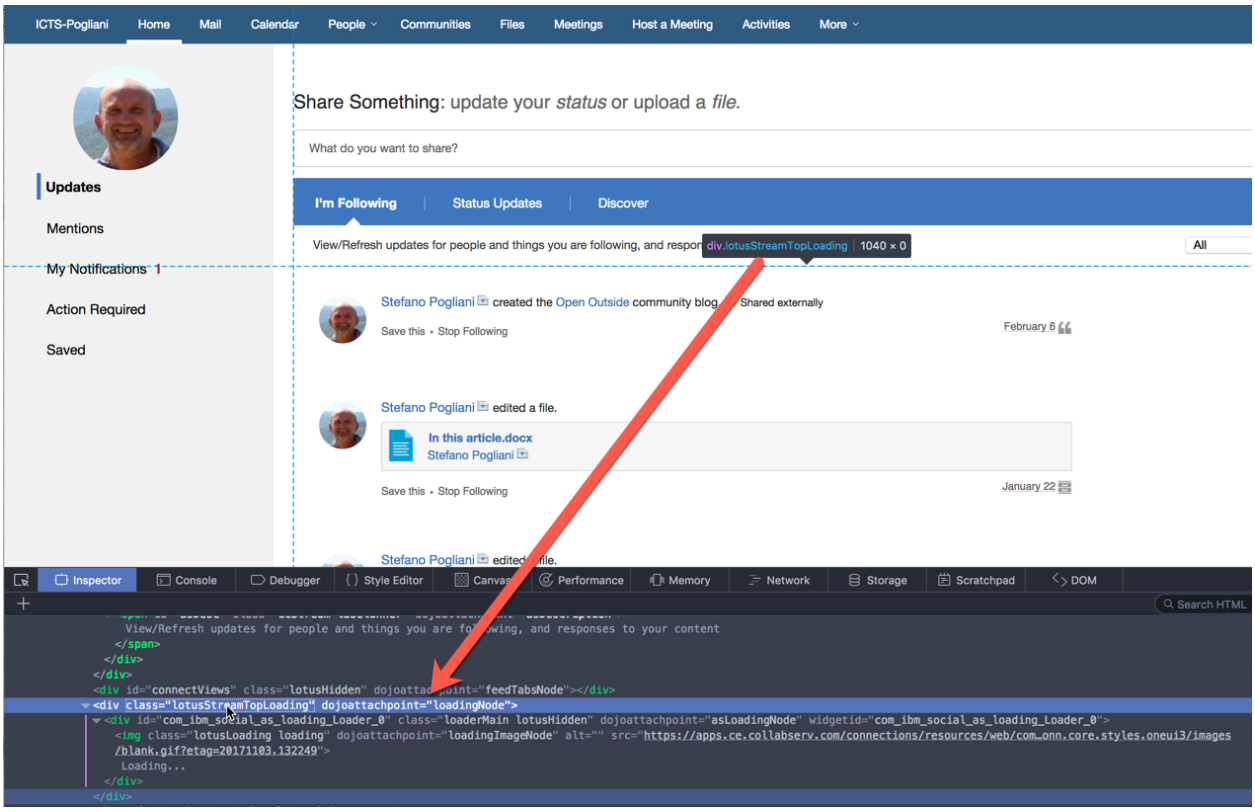
IBM Connections Customizer injects into the page one or more scripts according to the definition of the extension(s). The IBM Connections Customizer scripts that are loaded, are declared as <script> elements at the bottom of the HTML code of the page that is rendered.

If the script will execute immediately, it is likely that it would execute before the page is fully built in your browser.

This means that we cannot assume that the HTML elements that we need to modify are already rendered when the script executes (during the load of the page). Thus, we need to implement a strategy

that would allow the real changes made by the script to happen once all the required elements in the page are there.

We think that it is fair enough to assume that we could start modifying the two **** elements introduced in the previous sections when the ActivityStream starts loading: at that point, the two **** elements are likely to be there:



So, our strategy is going to be the following:

- Wait until the page will start loading the items in the ActivityStream
  This happens once the element whose class is “loaderMain lotusHidden” is rendered on the page.
  In order to be sure we target that very element, we can say that the element must be a child of the other element whose class is “lotusStreamTopLoading”.
  The “dojo” query for this is

```
dojo.query(".lotusStreamTopLoading div.loaderMain.lotusHidden")
```

	We could have used the “id” property of the child element in order to be reasonably sure that we would have targeted the one we wanted. Unfortunately, you will see that the “id” ends with a “_0”. This implies that the value of the “id” property is calculated at runtime by IBM Connections and we cannot be sure there will not be one ending with “_1” or other strings.

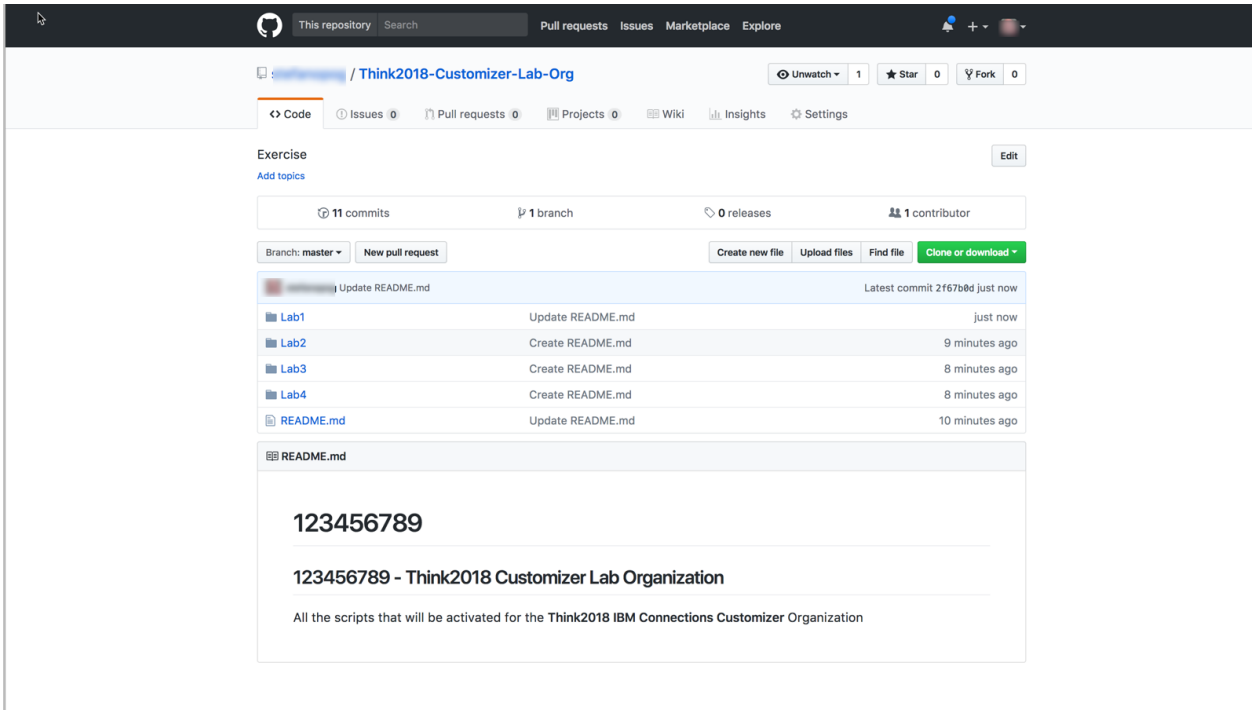
- Once that element is rendered, we can modify the “textContent” property of the two elements and the font color of the second one.

Preparing the Environment

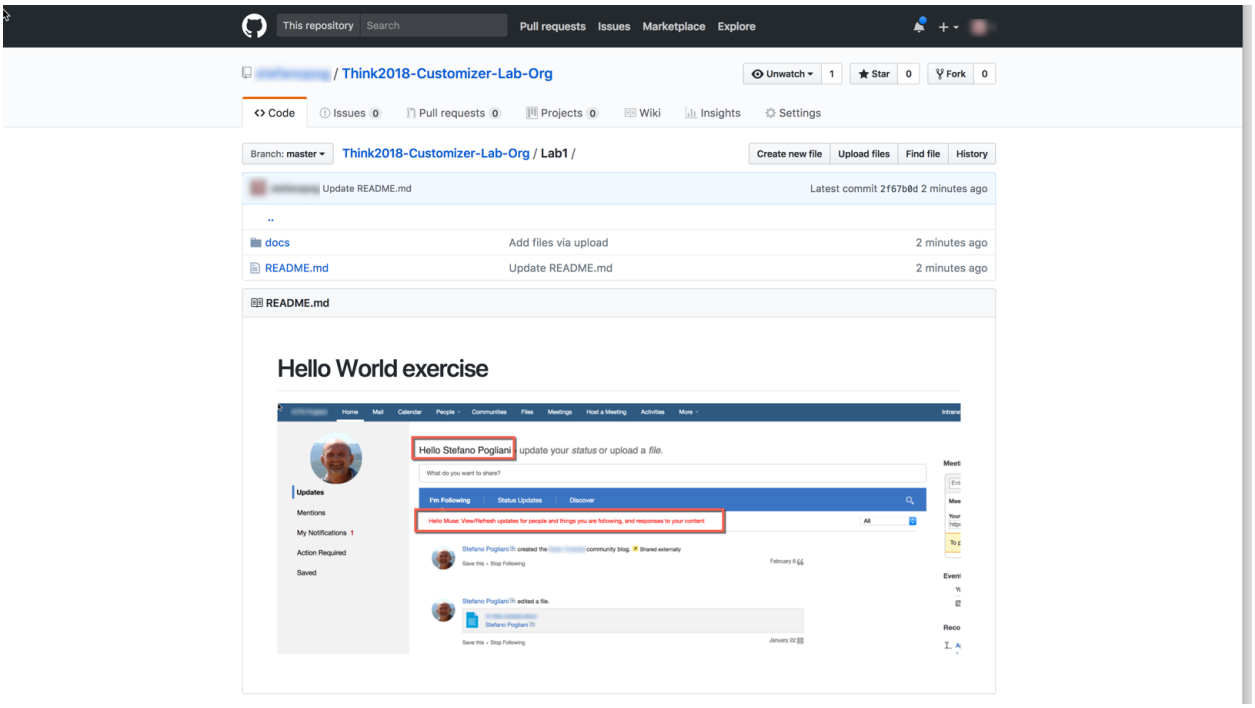
You have been invited as a Contributor to the Github repository associated to this organization.

	The instructor will provide you the URL and the credentials for accessing it.

You will arrive to a page similar to this one.



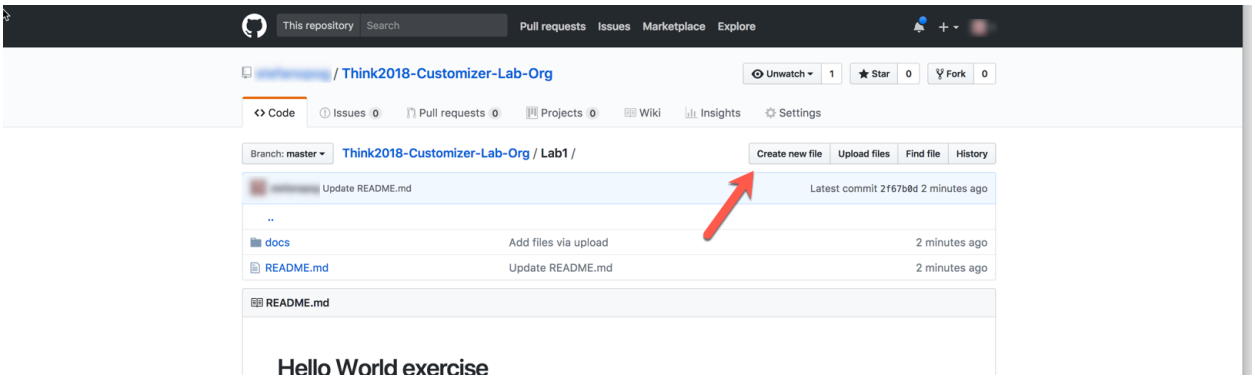
Click on the “**Lab1**” item and you will access this page:



Now you are ready to code!

Creating your script

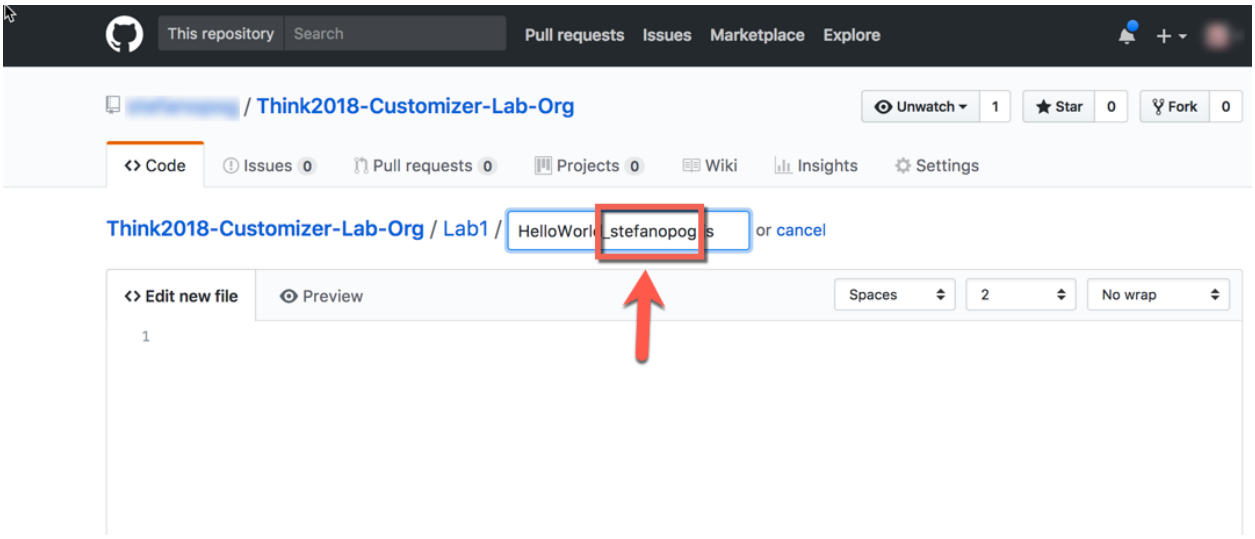
Create the Javascript file that will contain the code for your extension. Click on “Create File” as shown in the following image:



Now, you have to enter the name of the file. Since there are many people doing the exercise at the same time, we need to avoid conflict in filenames. For this reason, enter the following for the filename:

HelloWorld_**<identifier>**.js

<identifier> is a unique label that will be provided to you by the instructor (it is going to be the first part of the email address associated with your IBM Connections Cloud account for this lab).



Let's now proceed step-by-step in building our code together.

- Changing the Greeting.
  As we previously said, we need to modify the content of a **** element whose class is "**shareSome-title**"
  The code that we need to use is the following:

```
//
// Change the Greetings String
// Note that the username of the currently Logged in user can be obtained by means of the
// global variable "lconn.homepage.userName". This variable is defined by the
// IBM Connections Home page
//
var thisUser = lconn.homepage.userName;
dojo.query("span.shareSome-title")[0].textContent = "Hello " + thisUser + " -";
```

  We use the "dojo.query" function to retrieve the **** element. Since the "query" returns an array, we need to select the very first element of this array.

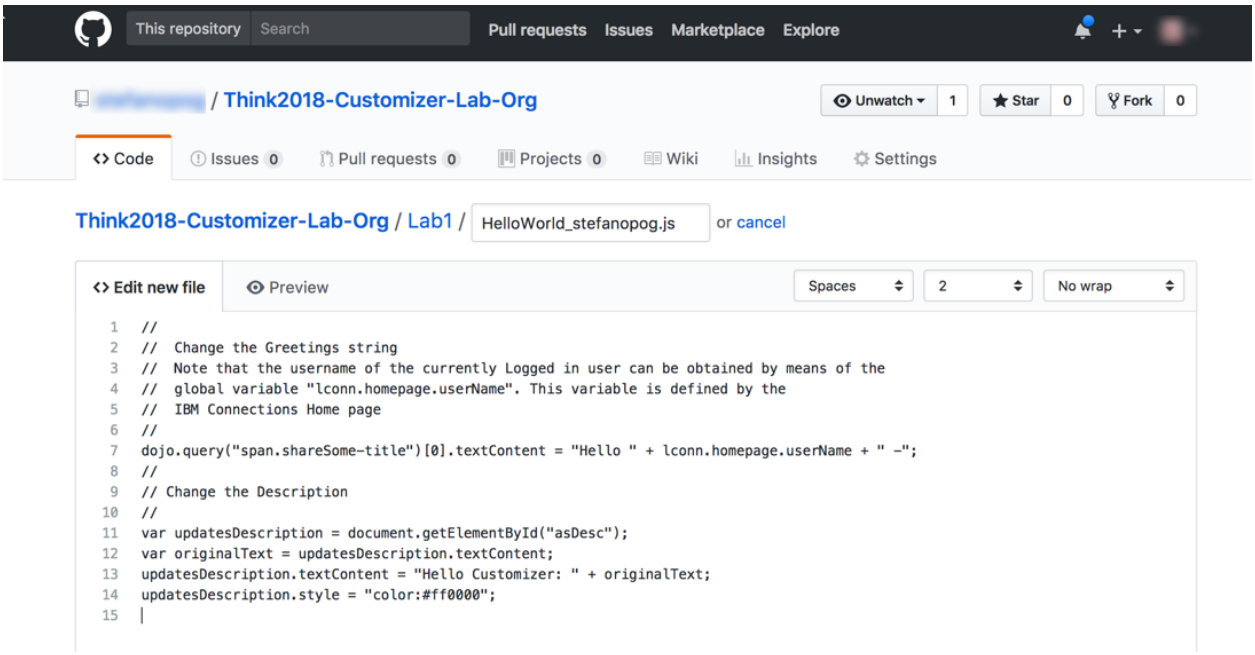
	In production code, we would certainly need to validate that the "query" actually returned the result that we were looking for.

- Changing the Description.
  As we previously said, we need to modify the content of a **** element whose id is "**asDesc**".
  The code that we need to use is the following:

```
//
// Change the Description
//
var updatesDescription = document.getElementById("asDesc");
var originalText = updatesDescription.textContent;
updatesDescription.textContent = "Hello Customizer: " + originalText;
updatesDescription.style = "color:#ff0000";
```

In this case we just used the “document.getElementById” function to retrieve the whose id is “asDesc”. We could have used the “dojo.byId” function.

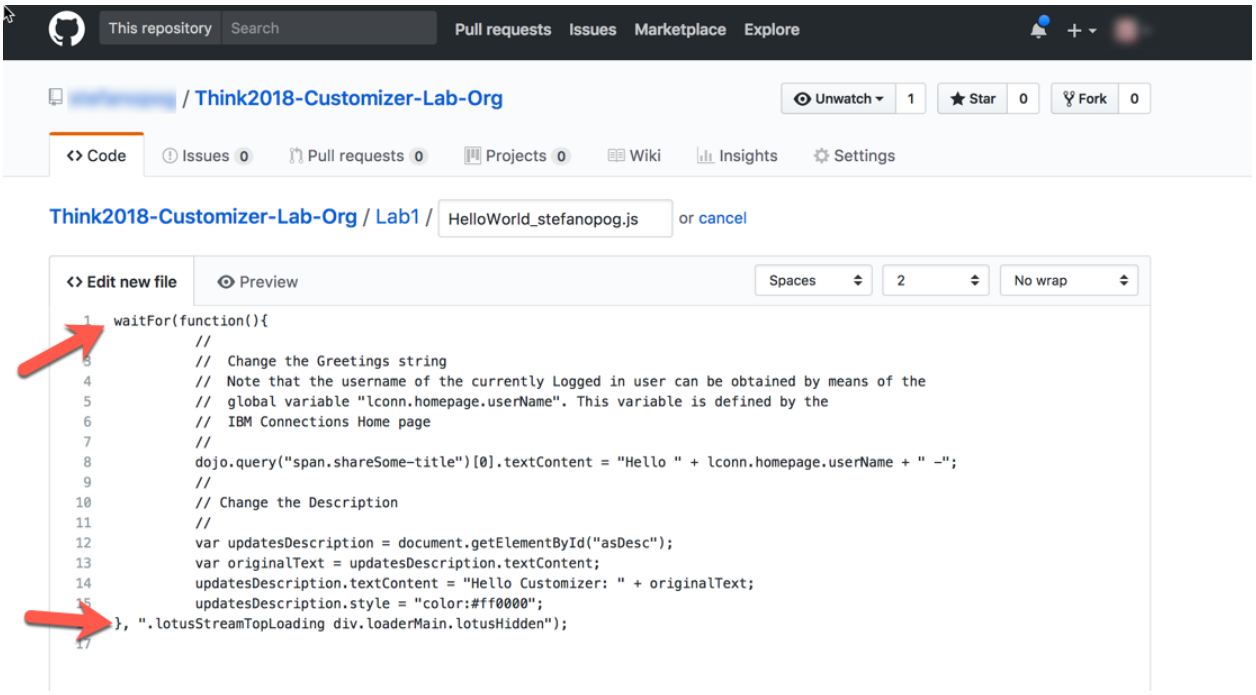
- At this point our script looks like this:



We have now to make sure that the code we just entered would be executed when the elements we are modifying are properly rendered (as mentioned previously).
To implement this behavior, let’s suppose that we have a function, called “waitFor”, which waits for the condition to be met and, then, will execute our code.
The way we would describe this in Javascript is by using the callback mechanism: the code we just entered becomes the body of a callback function executed by our “waitFor” once the condition is met.
At high level :

```
//
// wait for the required elements are rendered
//
waitFor(function() {
      …. Our code here ….
}, “.lotusStreamTopLoading div.loadMain.lotusHidden”);
```

So, change the code in your editor to look like what is shown in the following picture:

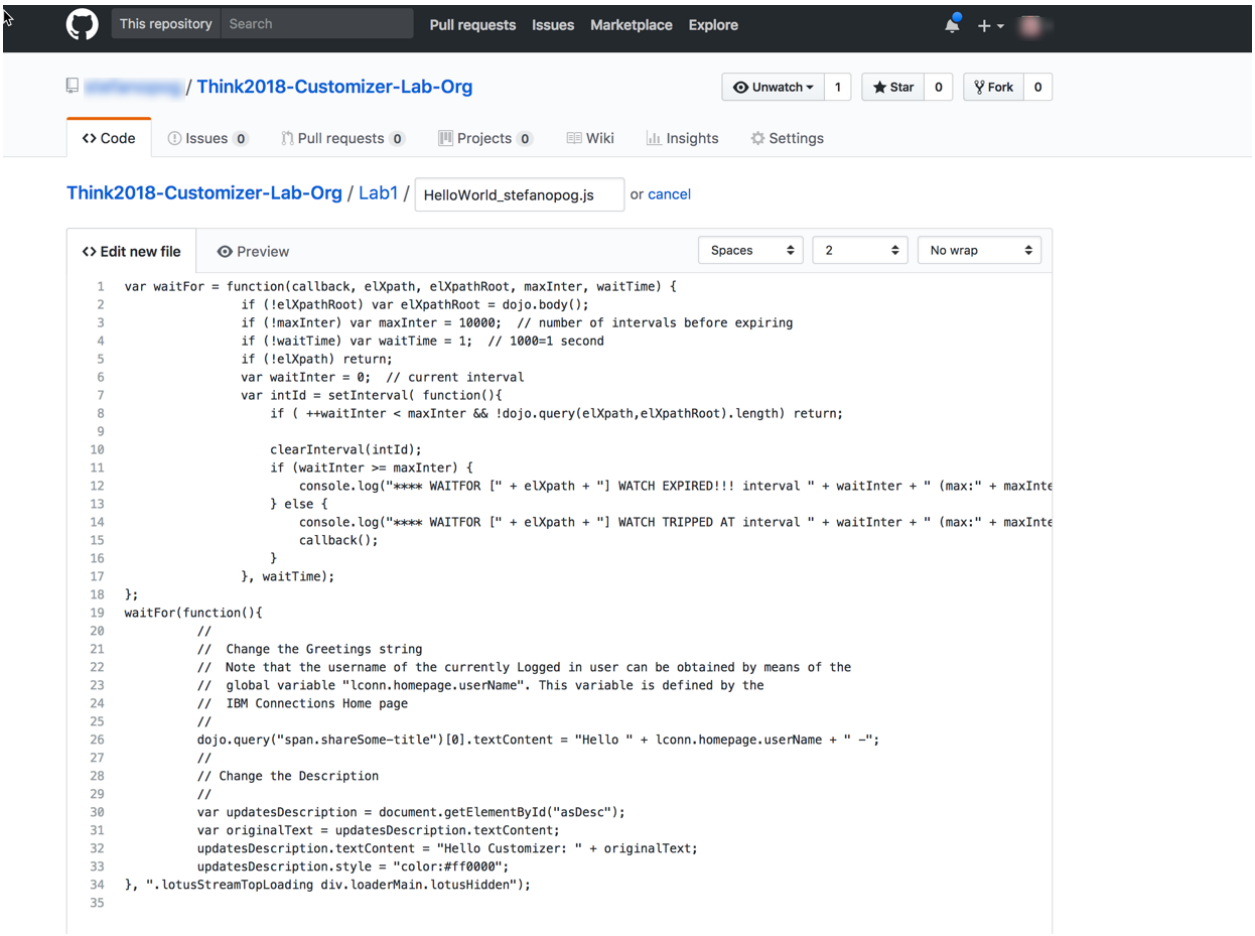


- Of course, we now need to provide the implementation of our “waitFor” function, right ?
  You can copy the code here and paste before the first line of the script:

```
var waitFor = function(callback, elXpath, elXpathRoot, maxInter, waitTime) {
    If (!elXpathRoot) var elXpathRoot = dojo.body();
    If (!maxInter) var maxInter = 10000;  // number of intervals before expiring
    If (!waitTime) var waitTime = 1;  // 1000=1 second
    If (!elXpath) return;

    var waitInter = 0;  // current interval
    var intId = setInterval( function(){
          if ( ++waitInter < maxInter && !dojo.query(elXpath,elXpathRoot).length) return;
          clearInterval(intId);
          if ( waitInter >= maxInter) {
                   console.log("**** WAITFOR [" + elXpath + "] WATCH EXPIRED!!! interval
" + waitInter + " (max:" + maxInter + ")");
           } else {
                   console.log("**** WAITFOR [" + elXpath + "] WATCH TRIPPED AT interval
" + waitInter + " (max:" + maxInter + ")");
                   callback();
            }
      }, waitTime);
};
```

This code simply implements a polling mechanism using the Javascript “setInterval” function.
We do not use the last 3 parameters of the function and the code defaults them to some value.
What is to be noticed is the way in which “waitFor” uses the **callback** function (which is actually the code we wrote in the previous part of the exercise).
At the end, the code in your editor should look like this:



- We are still missing one point though.
  In the code we wrote we assumed that the **dojo toolkit** would be available in the page in order for your script to use it. We are confident that the IBM Connections page will load the **dojo toolkit** at a certain point, but we cannot assume that it will be loaded by the time we first use it (look at the line 8 in the previous picture: our “waitFor” function uses the **dojo toolkit** to poll the presence of the element…).
  We certainly do not want our script to load the **dojo toolkit** another time…. So we need to wait for it to be ready on the page.

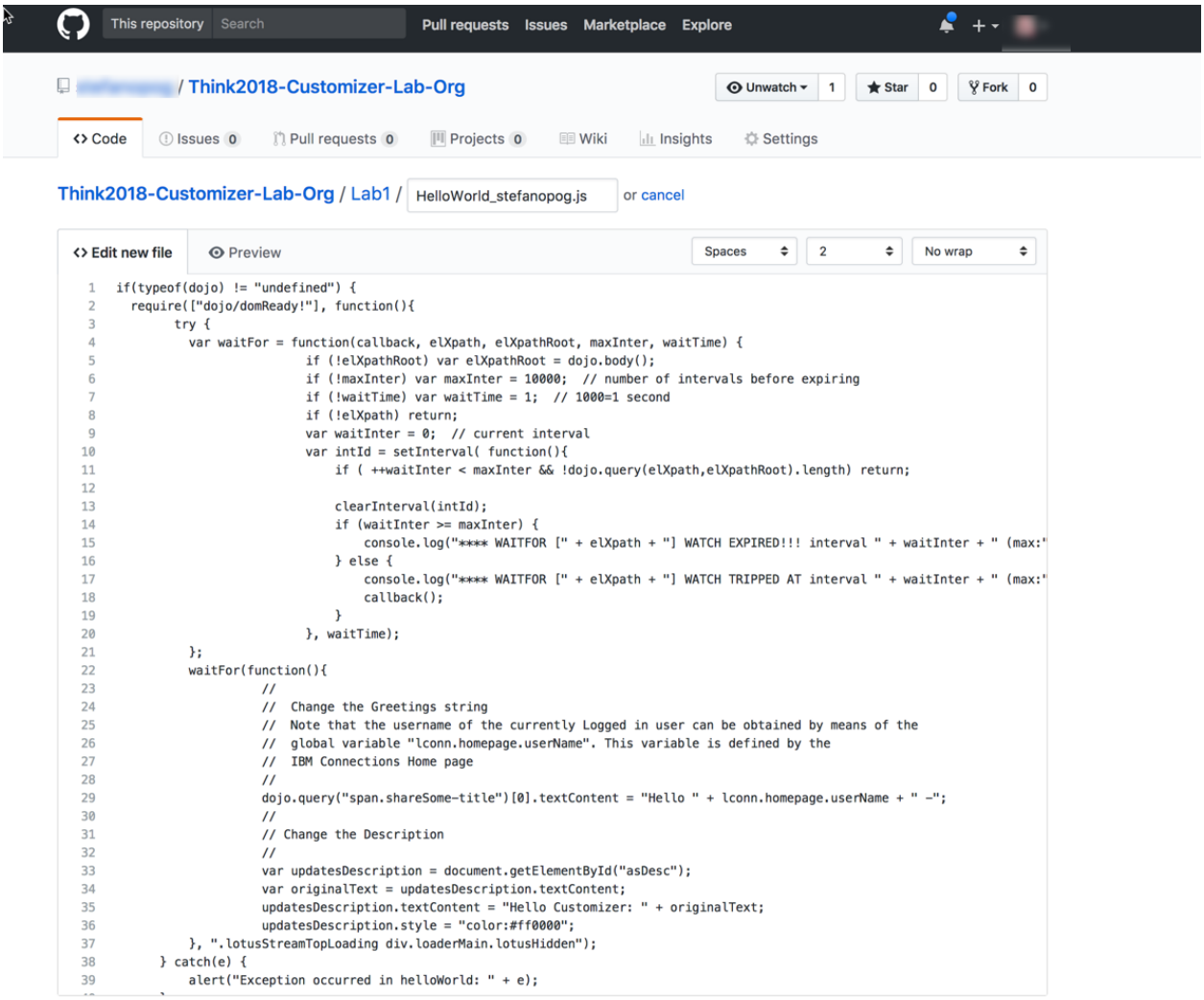
  Fortunately the **dojo toolkit** provides a standard mechanism to deal with this point: the “dojo/domReady!” plugin (described here https://dojotoolkit.org/reference-guide/1.10/dojo/domReady.html).

  So, we will simply need to wrap the code we wrote up until now in this way:

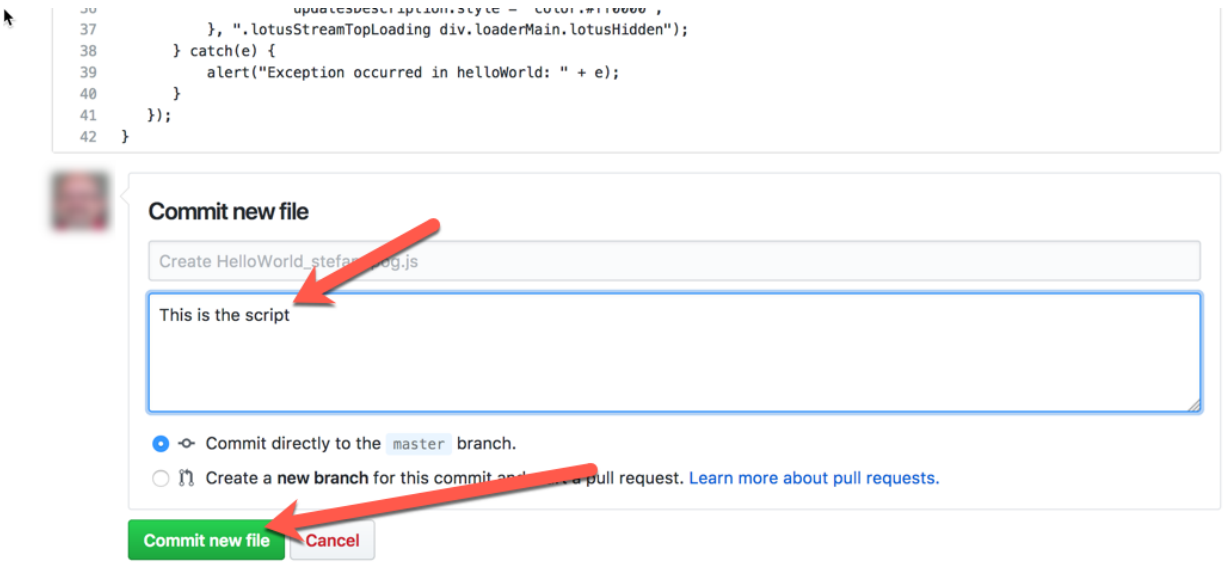
```
if (typeof(dojo) != “undefined”) {
    require([“dojo/domReady!”], function () {
          try {
                … all our previous code ….
       }  catch(e) {
             alert(‘exception occurred in HelloWorld : ‘ + e);
```

```
        }
    });
}
```

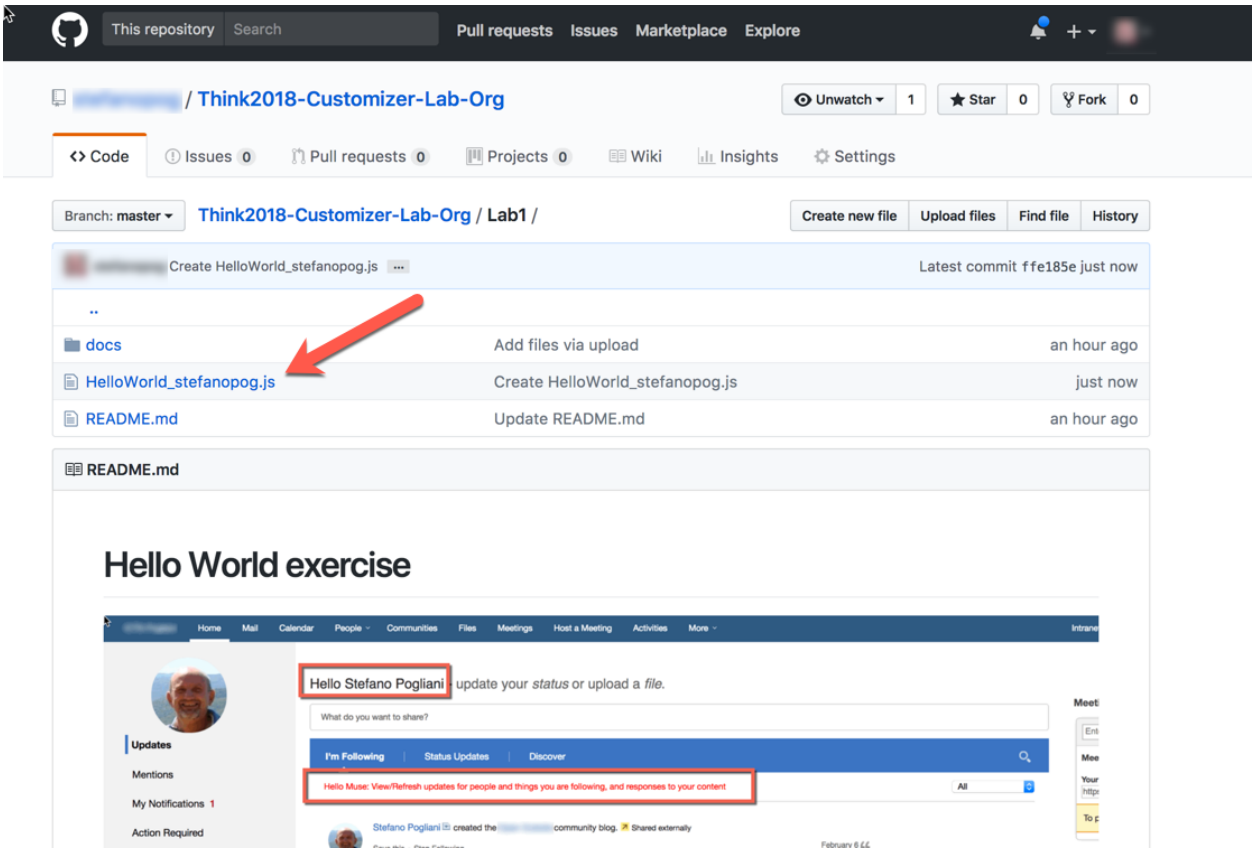
We have introduced a “try… catch” statement to globally catch any exception we may encounter.
At the end, the code in your editor should look like this:



- You can now commit your code to the repository. It is always good practice to add, as a comment, what you did.



- Your script is now safely recorded in the Github repository.



Creating your extension

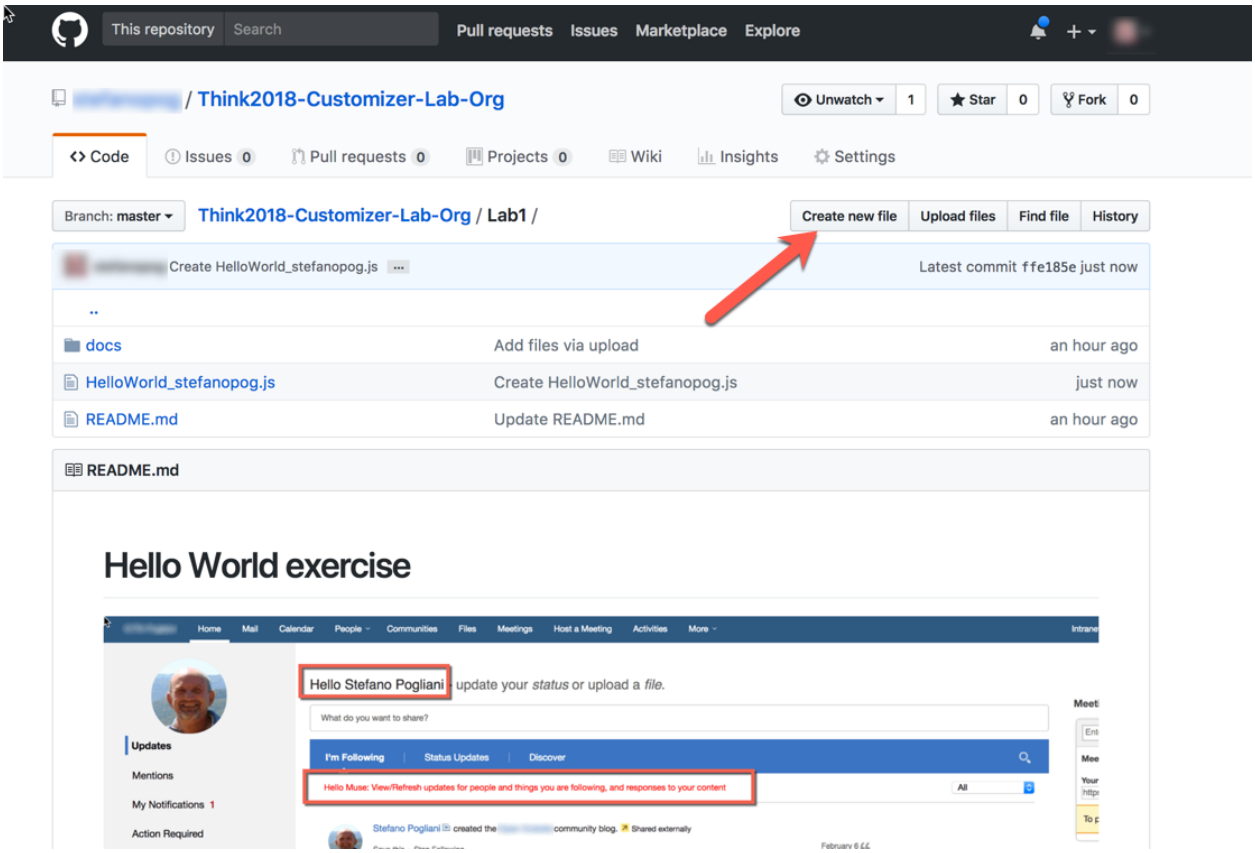
Whilst the extension can be directly created in the IBM Connections Cloud Administration panel, we suggest that you save the extension definition as a JSON file in the same Github repository.

	In this way, the extension may be used in other situations and may become part of the overall documentation of the project

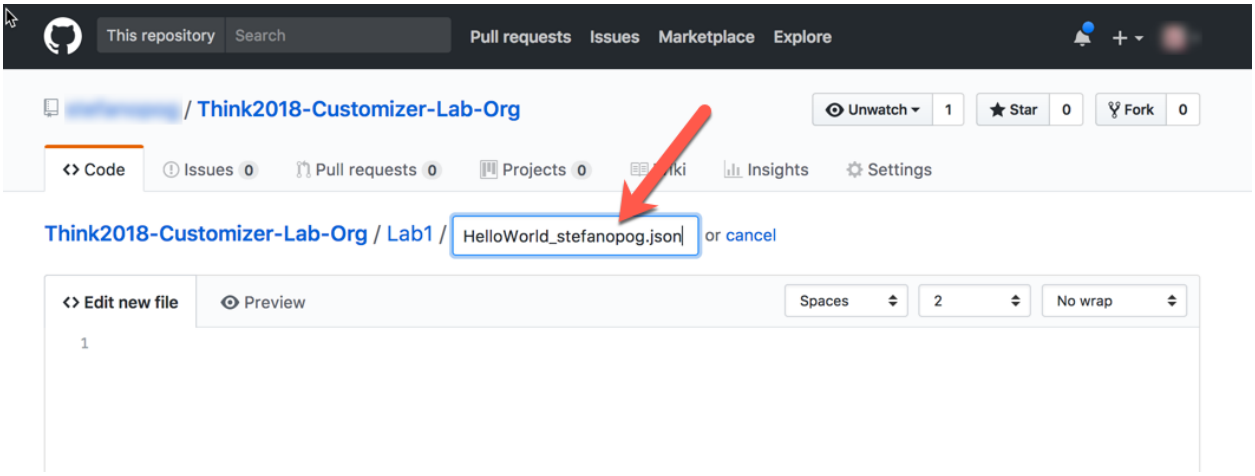
In order to create the extension, we need to create a new file inside our Github repository. The file will follow this naming convention

HelloWorld_**<identifier>**.json

Where <identifier> is a unique label that will be provided to you by the instructor (it is going to be the first part of the email address associated with your IBM Connections Cloud account for this lab). In this way, everybody will immediately understand that this is the extension definition associated with the HelloWorld_**<identifier>** script.



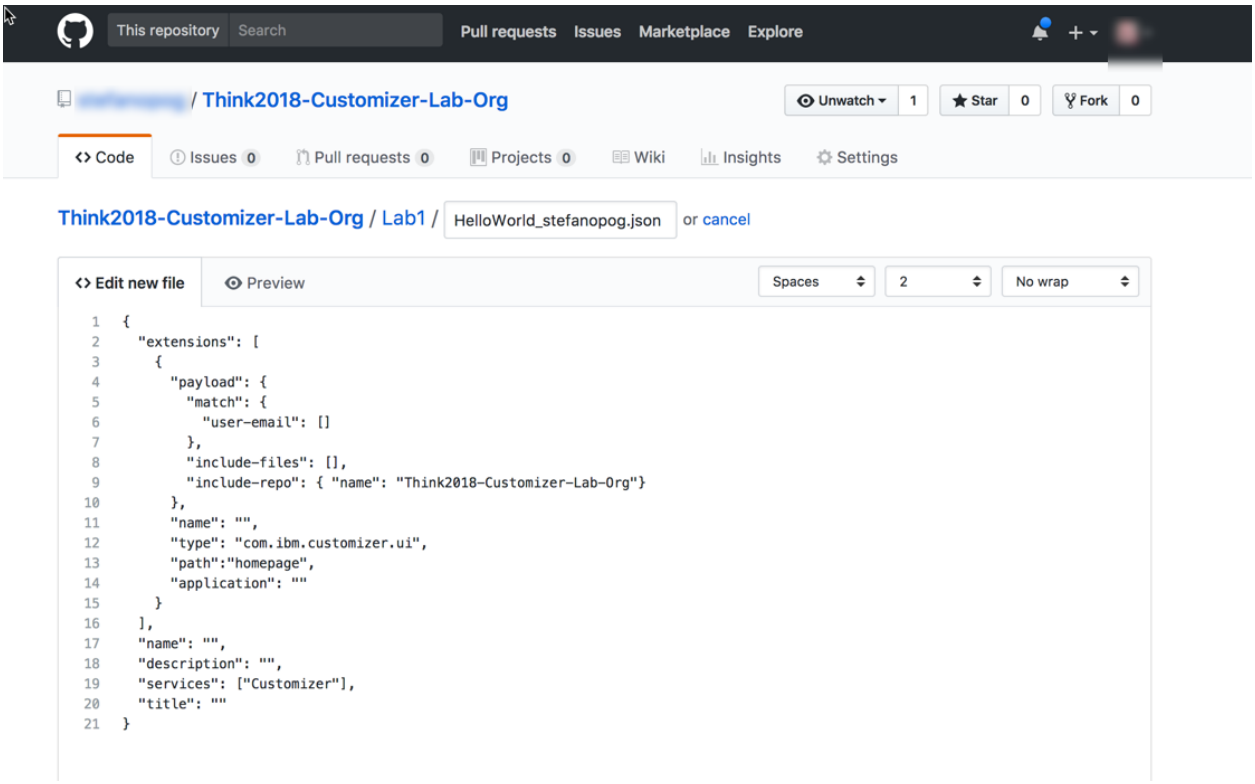
And, then,



Let's first copy this empty template inside our new json file (a copy of this template is available as the `Lab1/emptyTemplate.json` under "**Reference Repository**"):

```
{
    "extensions": [{
                "payload": {
                        "match": {
                                "user-email": []
                         },
                        "include-files": [],
                        "include-repo": {
                               "name": "Think2018-Customizer-Lab-Org"
                        }
                 },
                 "name": "",
                 "type": "com.ibm.customizer.ui",
                 "path":"homepage",
                 "application": ""
          }],
      "name": "",
      "description": "",
      "services": ["Customizer"],
      "title": ""
}
```

So you will get this result:



Some of the values of the JSON descriptor are already provided. A complete documentation for the attributes is available at this url :
https://github.com/ibmcnxdev/customizer/blob/master/docs/IBMConnectionsCustomizer.pdf

- "include-repo": (line 8)
  This MUST be the name of the Github repository that we are using. It is the "root" directory that the IBM Connections Customizer engine will use in order to retrieve the scripts that the extension uses
- "type": (line 13)
  This is a string whose value "com.ibm.customizer.ui" will tell IBM Connections Cloud which type of IBM Connections Customizer service will be used by the extension
- "path": (line 14)
  The value of "homepage" will instruct the IBM Connections Customizer engine to add the scripts referenced by the extension only to the "homepage" service of IBM Connections
- "services": (line 19)
  This is a string whose value "Customizer" will tell IBM Connections Cloud which type extension it needs to register in the Application Registry.

Let's now fill the other values that are required:

- "user-email": (line 5)
  We can restrict the injection of the scripts associated to the extension only to users whose email address will match the string.
  Since we do not want to interfere with other people working at the same time on the same IBM

Connections Cloud organization, we need to provide a value for this attribute.
The value needs to be the email address you have been provided by the instructor to log into IBM Connections Cloud (enclosed in double-quotes). So something like :

```
"user-email": ["stefanopog@think2018-Customizer.com"]
```

In real life, you may not need to use this attribute if the extension you are creating will be used by **all the people** in your organization.
See the documentation for discovering all the possibilities associated to the "match" clause.

- "include-files": (line 7)
  This is where you will actually reference the script that you created earlier in the exercise.
  The path to reference the script is relative to the Gituhub repository you are using (see the "include-repo" clause described above).
  So in your situation you need to provide the following value :

```
"include-files": ["Lab1/HelloWorld_<identifier>.js"]
```

- "name": (line 12)
  this is the name of the Extension.
  You can use the following:

```
"name": "HelloWorld Ext Lab1 <identifier>"
```

- "name": (line 17)
  This is the name of the Application we are going to declare to IBM Connections. Typically, an "Application" can declare one or more extensions (in our example, we have one extension only associated with the application).
  This name **must be unique** for the IBM Connections Organization, since it is the name that will be recognized be the IBM Connections Application registry. You can use the following name:

```
"name": "HelloWorld App Lab1 <identifier>"
```

- "application": (line 15)
  This entry (which is within the "extension") actually needs to reference the correct "Application"). So, the value you will provide for this attribute **MUST BE the same** as the value you provided for the "name" attribute of the Application (line 17). You can use the following name:

```
"application": "HelloWorld App Lab1 <identifier>"
```

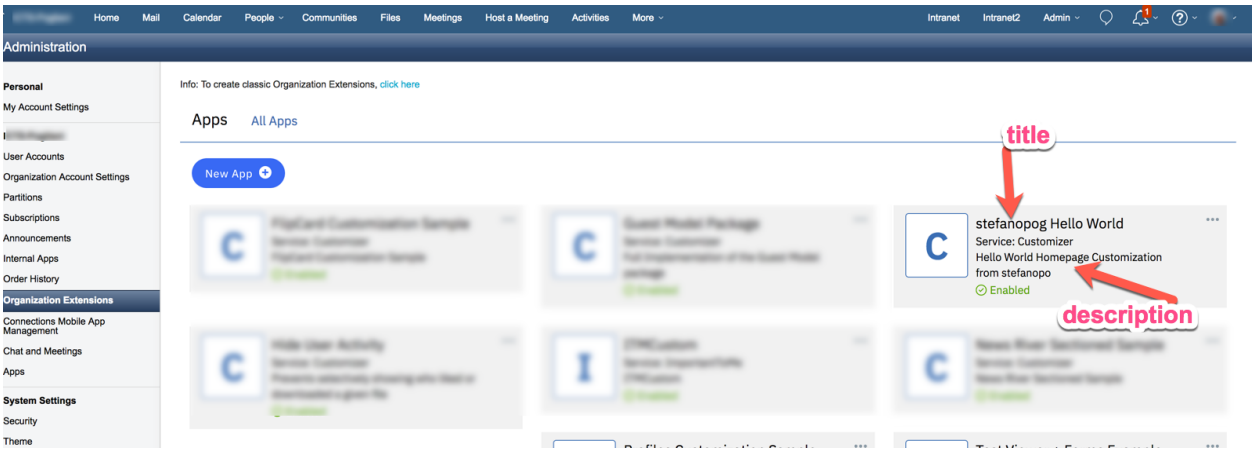
- "description": (line 18)
  This is a free text description of what the Application (and its Extensions) provide. You can use the following:

```
"description": "Hello World Homepage Customization from <identifier>"
```

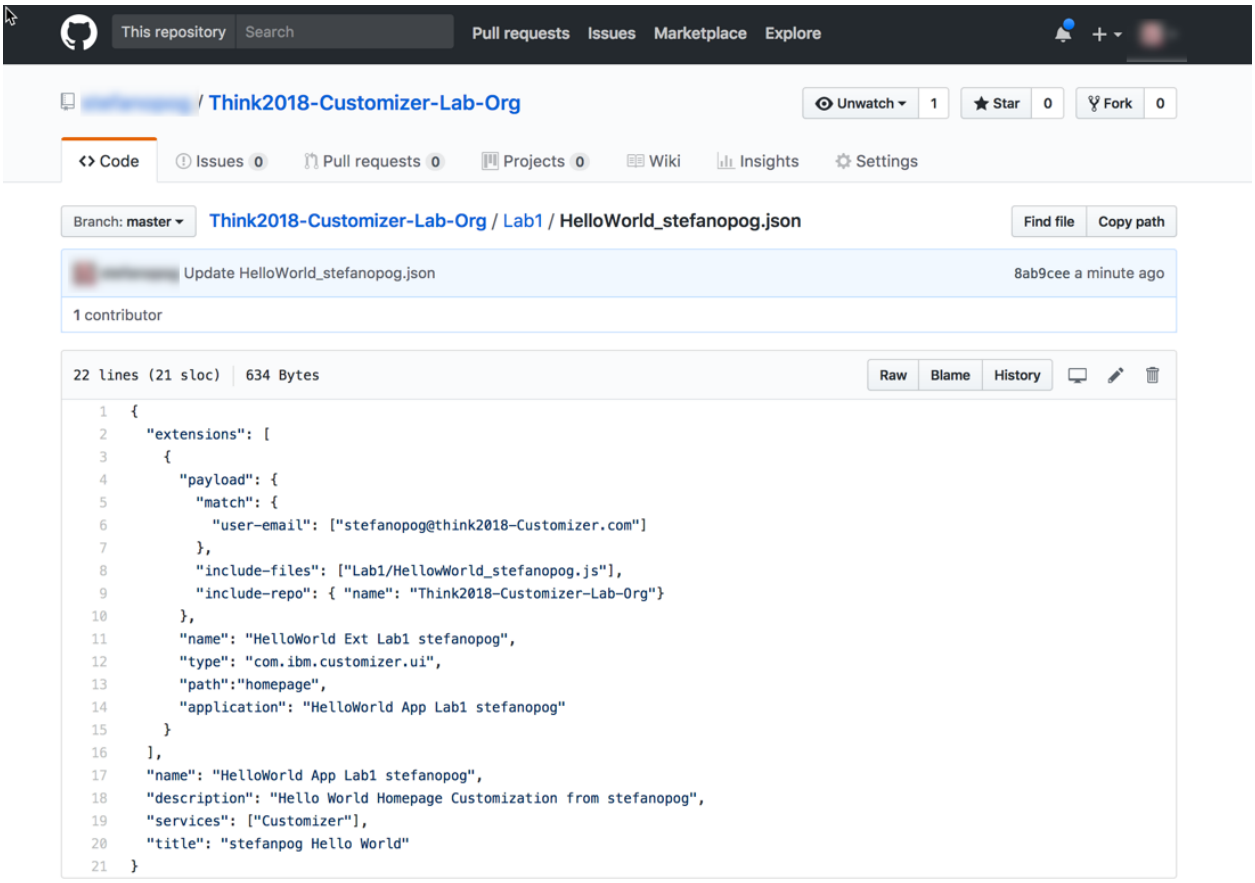
- "title": (line 20)
  This is the title that will appear in the IBM Connections Cloud application registry. You can use the following title:

  ```
  "title": "<identifier> Hello World"
  ```

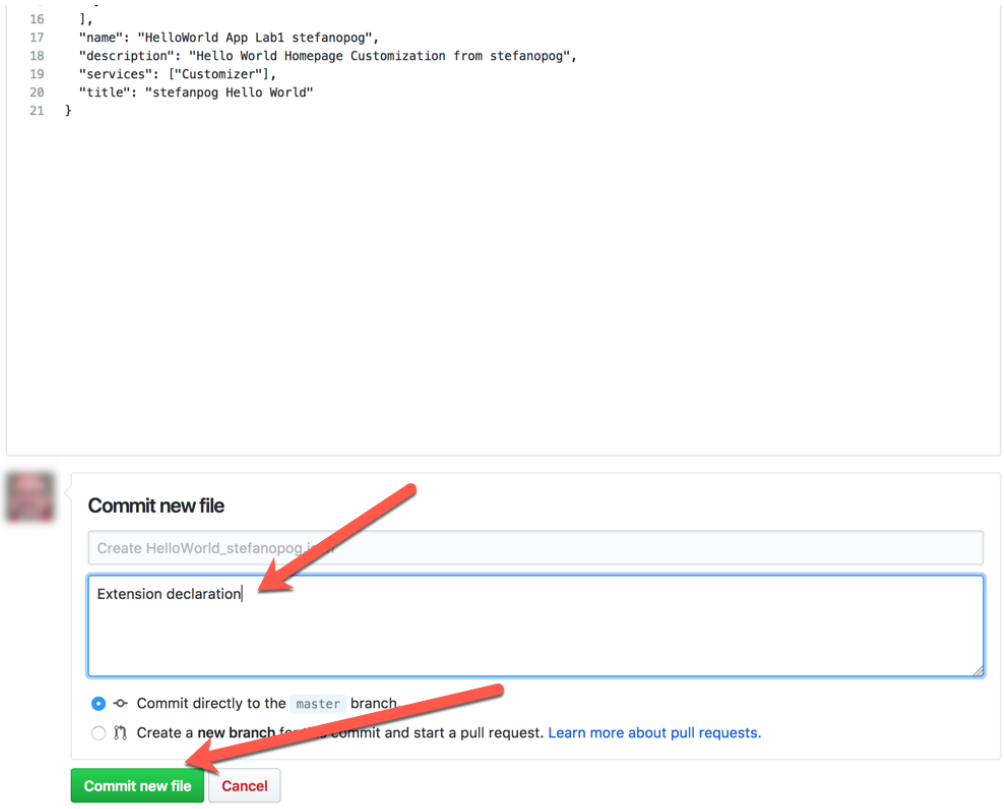
  Once the Organization admin will use your JSON file to declare the extension, this is how it will appear in the IBM Connections Cloud Application Registry:



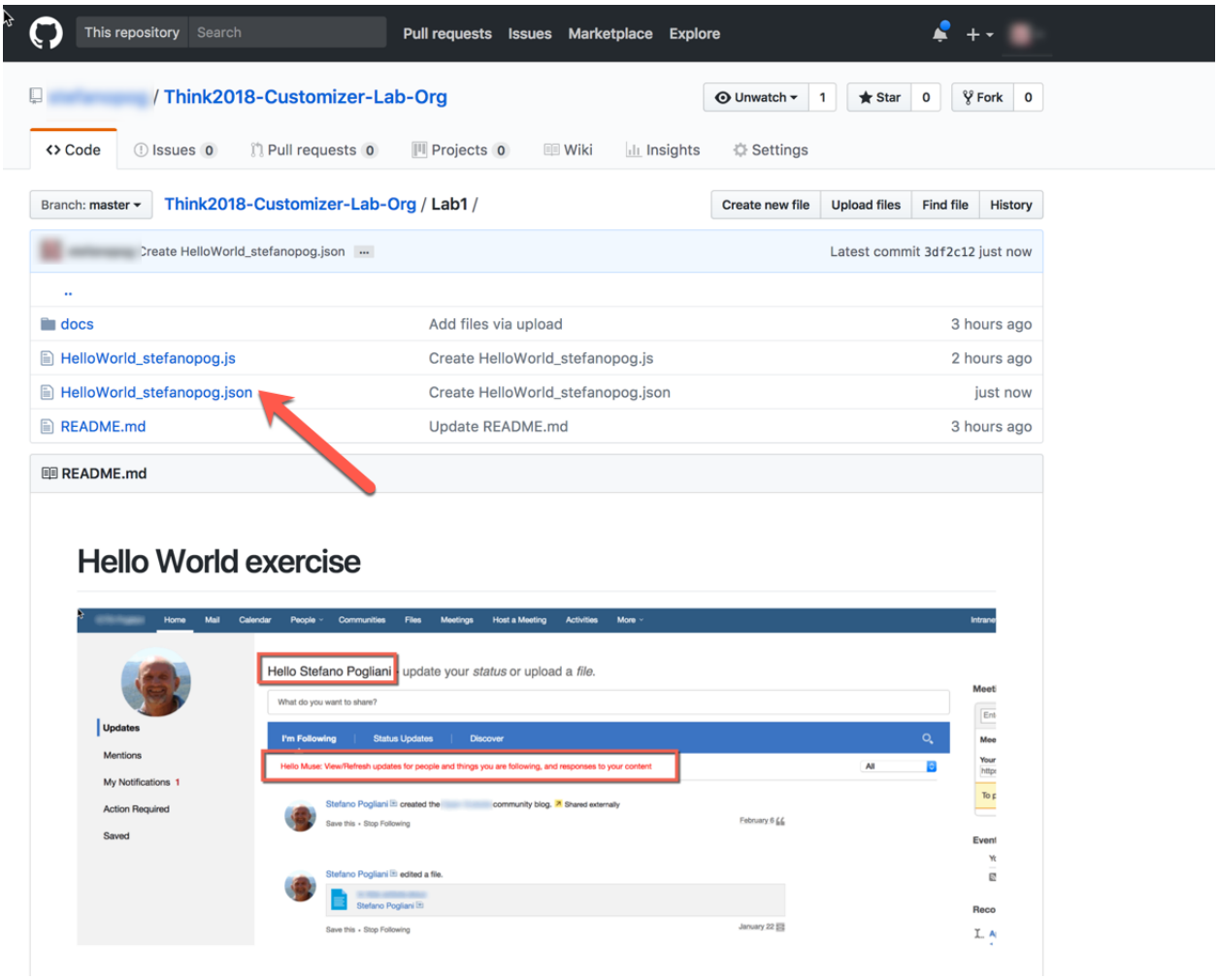
At this point, the JSON file you are editing in Github will look like this:



You can Commit the change to the Github repository:



And your Repository will look like this:



Making your extension available

At this point, everything is ready to be put in production. This is what the documentation states:

a. Share your repo with IBM – add "ibmcndev" as a collaborator

b. IBM (ibmcnxdev) then creates a fork of your repository under github.com/ibmcnxdev and grants you read access by default.

c. You can continue to work on your extension using your original repo for your source code activity, but once you are ready to deliver to IBM Cloud you must issue a pull request to IBM.

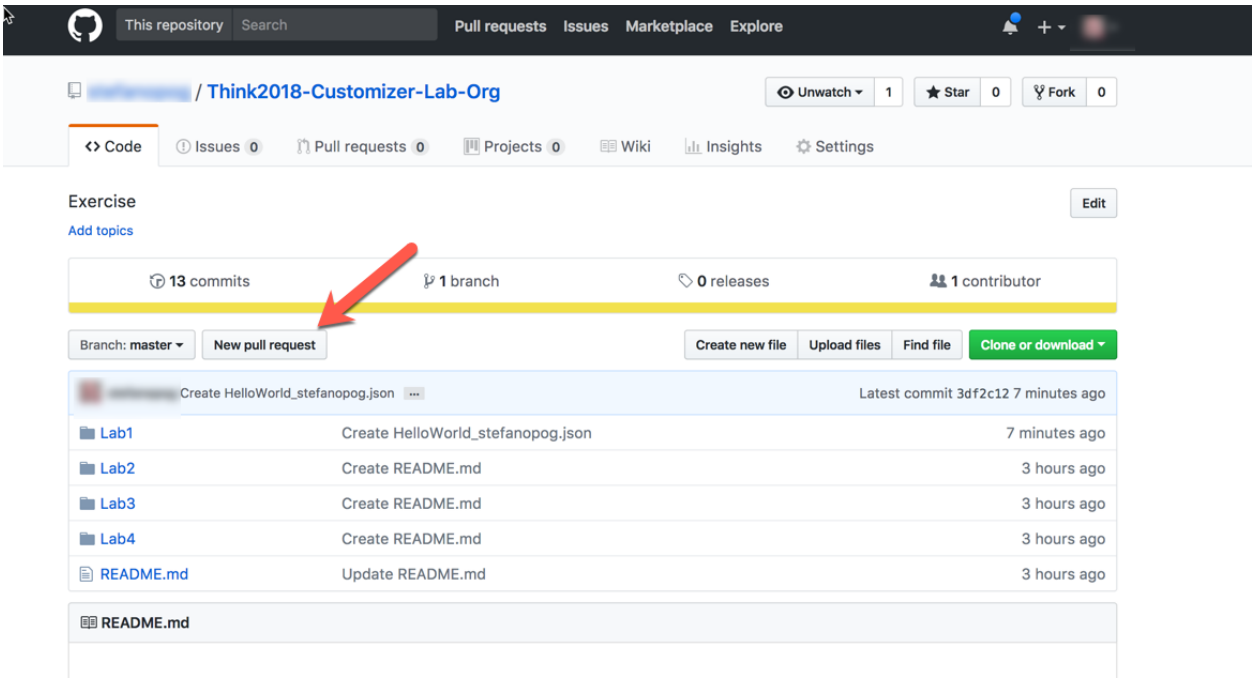
d. IBM merges your pull request once acceptance criteria are met.

e. Upon merge, the repo files are automatically pushed to IBM Customizer via a webhook.

f. Rinse & repeat starting at Step (c) for extension updates.

Steps **a.** and **b**. have already been done for you before you started the exercise.

Now, you need to execute **Step c**.
Go back to the root directory of your Github repository and click on the “Pull Request” button:



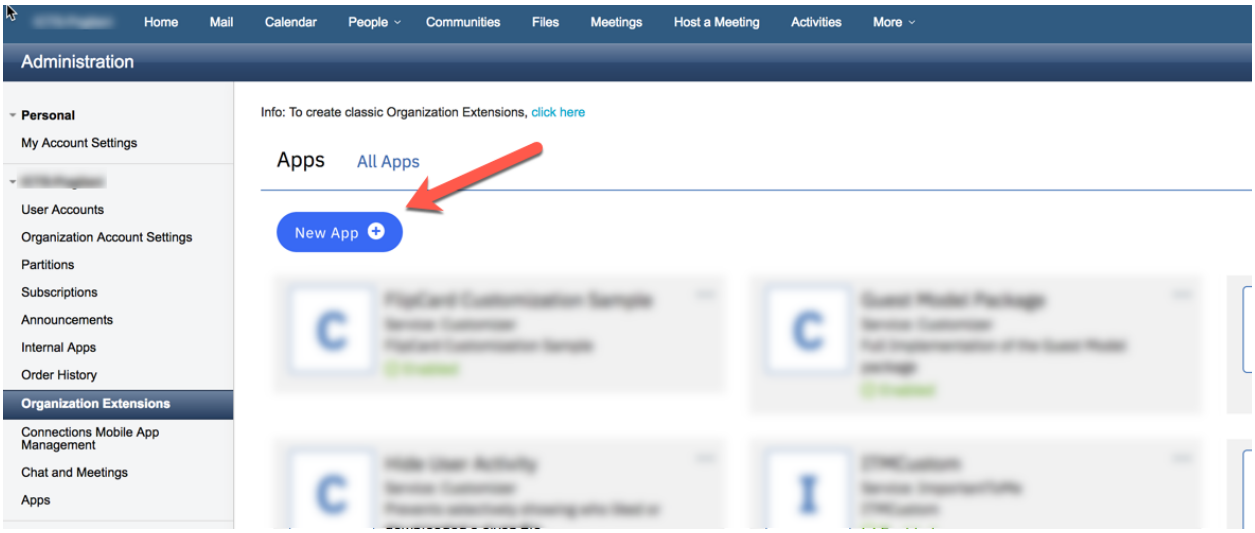
Activating the extension

Now your extension has been validated. The validation process (from **IBMCNDEV**), if successful, will make the files you created in this exercise available to IBM Connections Cloud and to IBM Connections Customizer.

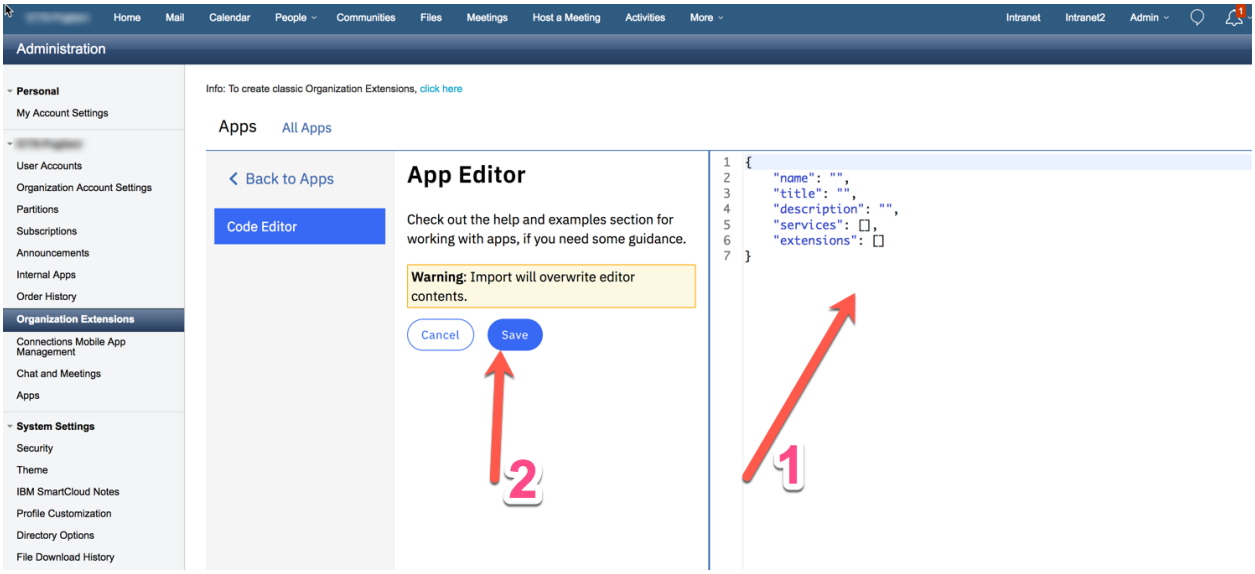
The only thing that is missing is to actually define the Extension in the IBM Connections Cloud Application Registry. This is the step that informs IBM Connections Customizer of what to do with the scripts that you just created.

The IBM Connections Cloud organization administrator needs to be given the JSON file which contains the extension (the one you created here).

He will go to the IBM Connections Application Registry and will declare a new extension:

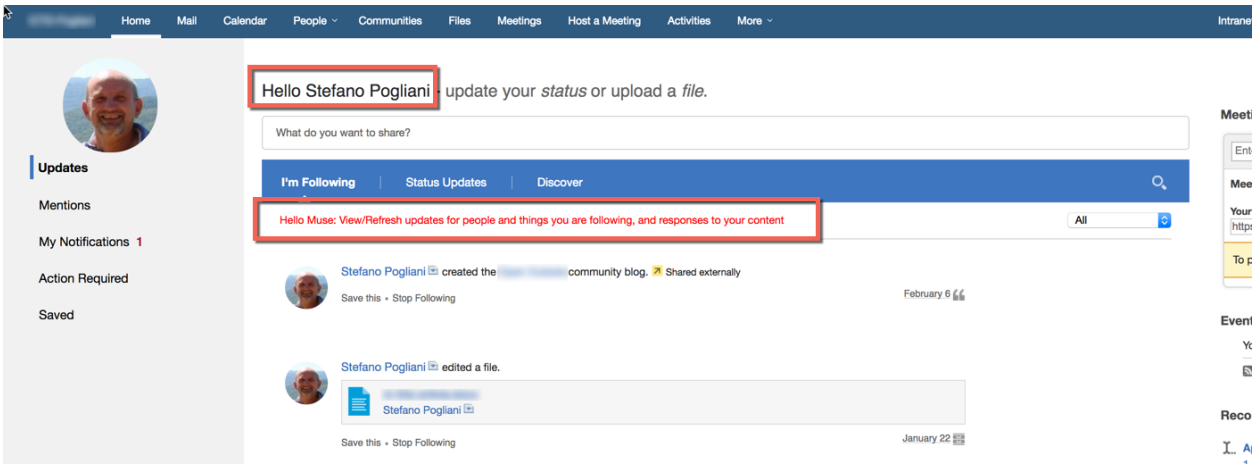


A new screen is shown:



The IBM Connections Cloud organization admin will replace the content pointed by the right arrow (1) with the content of the JSON file you created and, then, **saves** (2) the extension.

You can now go to the homepage and see the final result:



This ends Lab 1.
Congratulations!

Lab 2 Playing with CSS

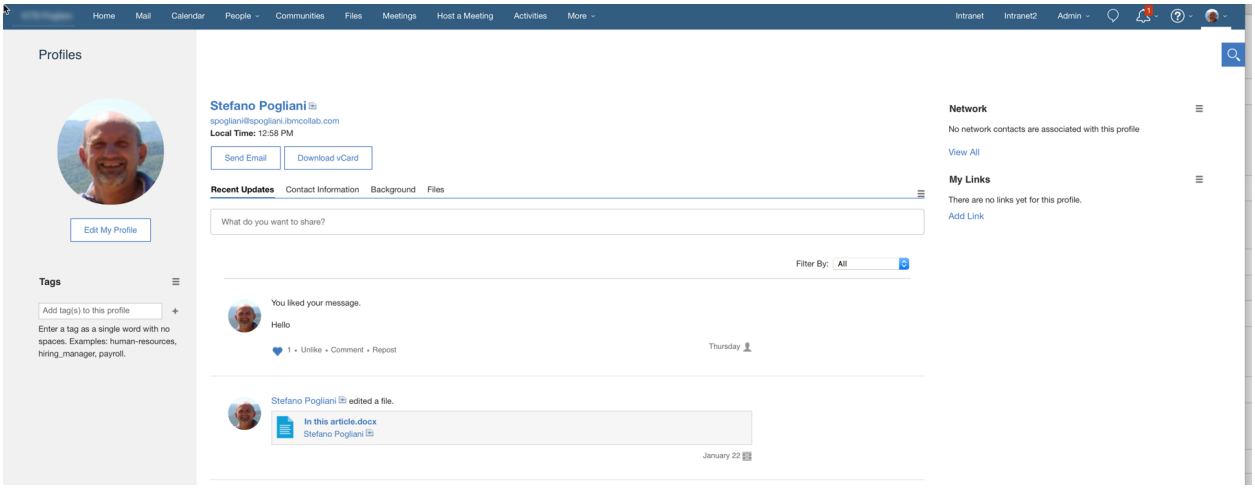
In the previous exercise we played with some DOM Elements using IBM Connections Customizer. We just saw how, programmatically, we could modify the style of an HTML element (when we colored the “Description” in red).

What if we would like to apply extensive changes to the CSS of one or more IBM Connections pages? Probably using Javascript code to wrap CSS directives is going to be tedious and error-prone. It would be much easier if we could compile all of our style changes into a CSS file and have IBM Connections Customizer apply that style with a single command.

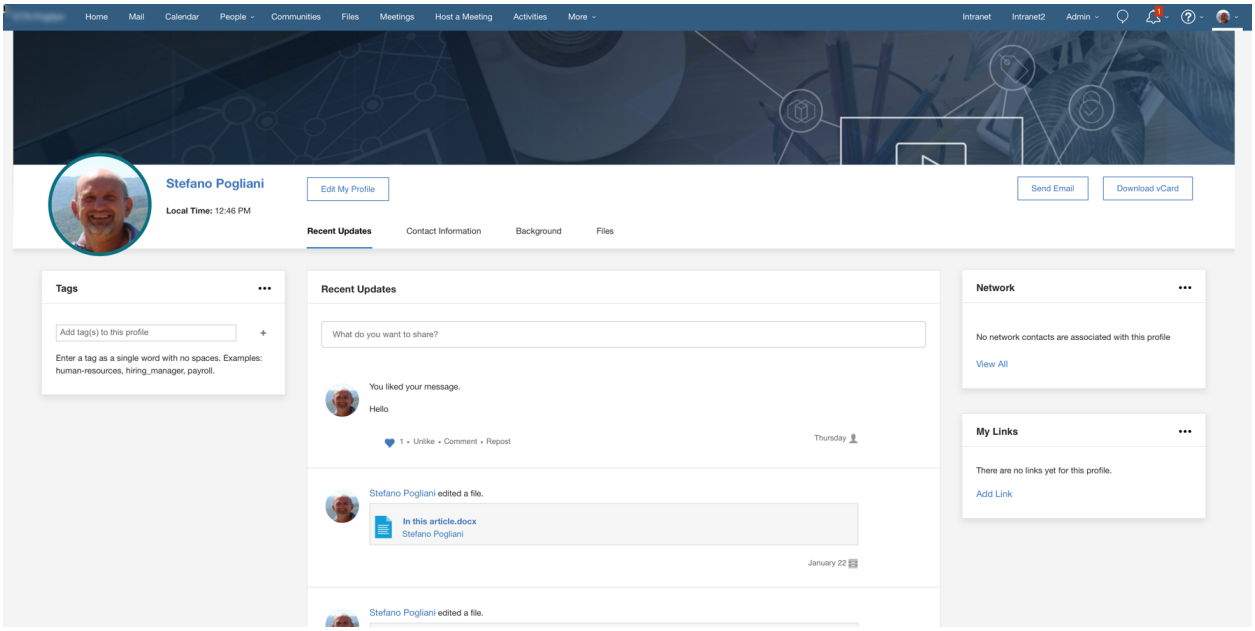
This is the goal of this second exercise. Once again, the extension is really simple by itself, and the focus is really on showing how a CSS file can be used by an IBM Connections Customizer extension.

What we want to achieve

When accessing the traditional IBM Connections Cloud Profile page, you are likely to see something similar to this:



In our exercise, we will modify the look-and-feel of this page to be the following one:



We are not going to modify any functionality from the standard Profile page, but simply changing the way in which the standard information is shown.

Two different approaches

We will present two ways to this usecase:

1. The first one makes use some Javascript code which will inject the CSS file containing the stylesheet we want to apply
2. The second uses a new functionality, recently introduced in IBM Connections Customizer, which allows the developer to directly reference CSS files in the Extension definition.

We thought that, whilst the second one is, by far, the quickest and the most efficient from a performance perspective, the first would be a useful exercise that would allow students to further understand the way in which IBM Connections Customizer works.

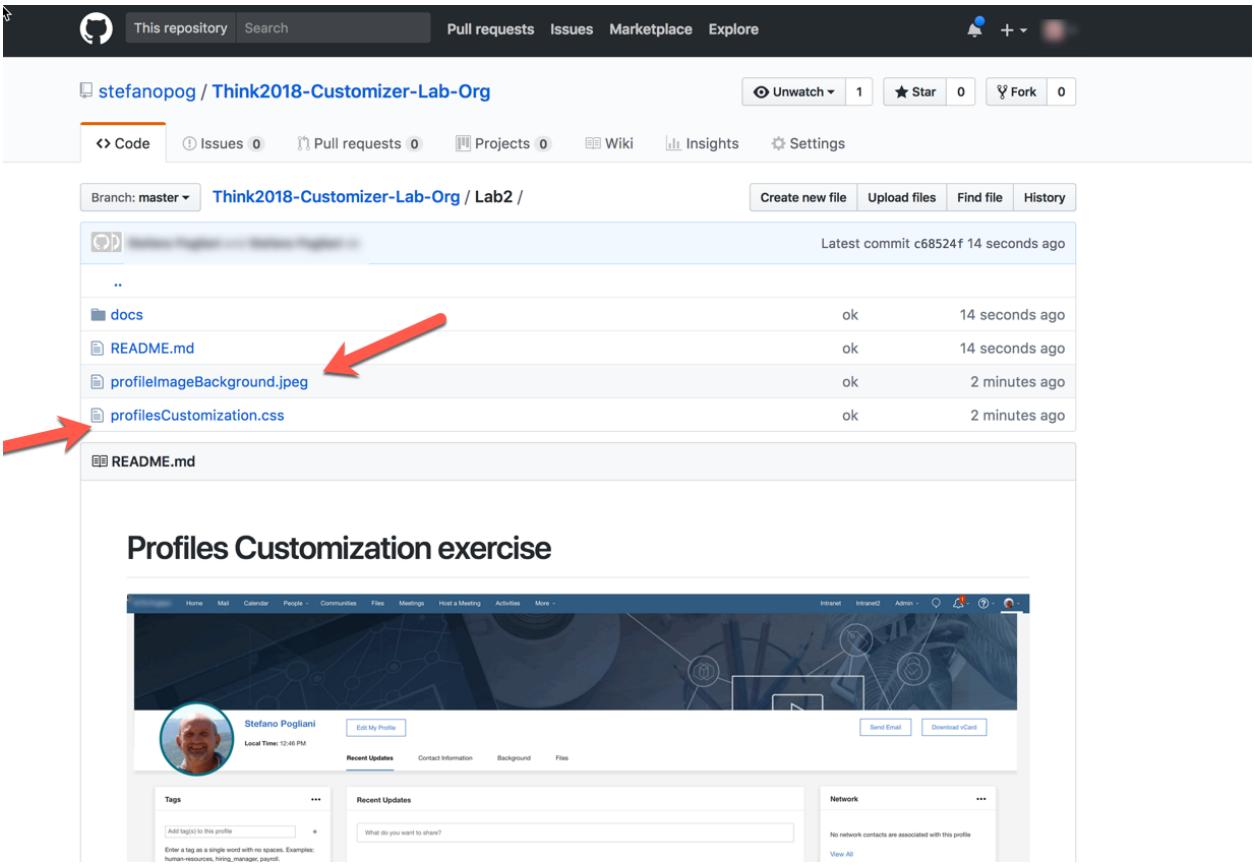
The CSS File

Both approaches would require the definition of the CSS file.

	The objective here is not to explain the Stylesheet itself. You may experiment with the CSS file which is provided as an example in order to better understand the way the CSS directives have been applied in order to produce the desired result.

The CSS file that we will be using, and the associated image used as a background are already available under the Lab2 directory of the Github repository associated with the IBM Connections Cloud organization we are using.

Please do not touch nor modify those two files as they will be used by the code we are going to write.



	For your own convenience, the CSS file and the associated background image are also available under the Lab2 directory of the "**Reference Repository**"

Using Javascript injection

We have seen in our first Lab how the IBM Connections Customizer injection engine works. So, we know that if we create a Javascript script, a reference to that script will be placed at the end of the HTML code that displays the page.

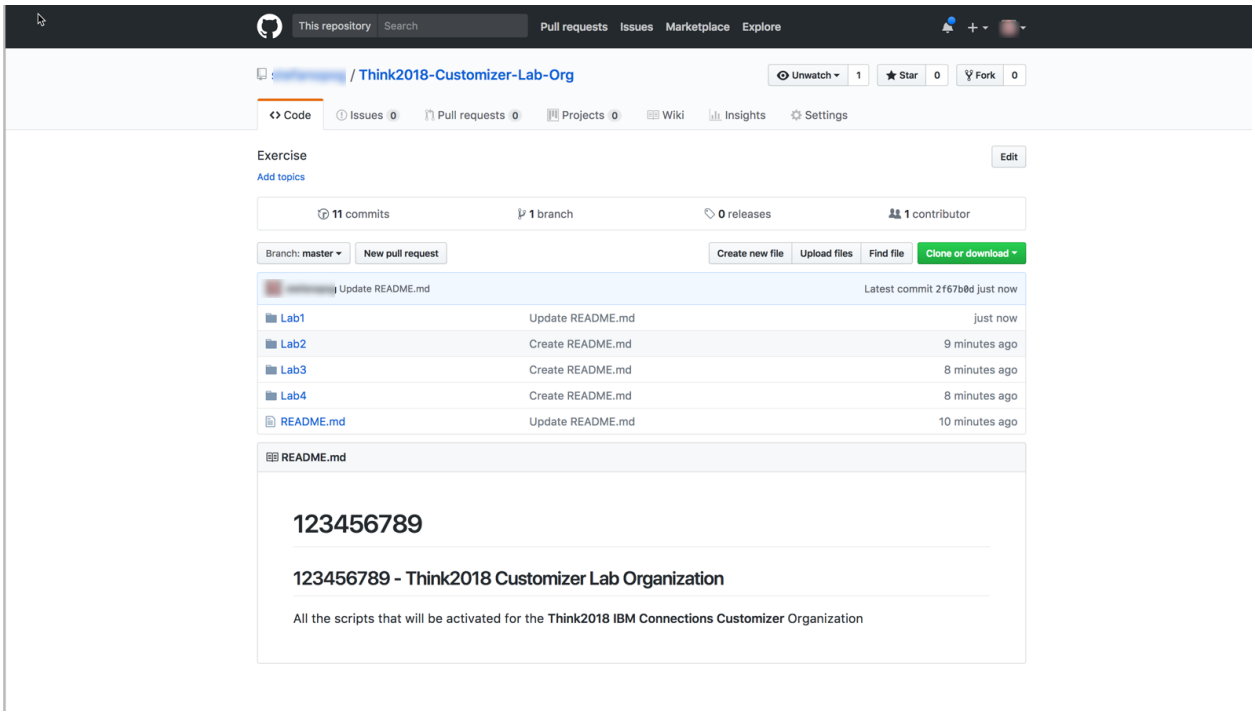
If the script will contain executable statements, they will then be executed immediately as the HTML finishes loading.

Preparation Activities

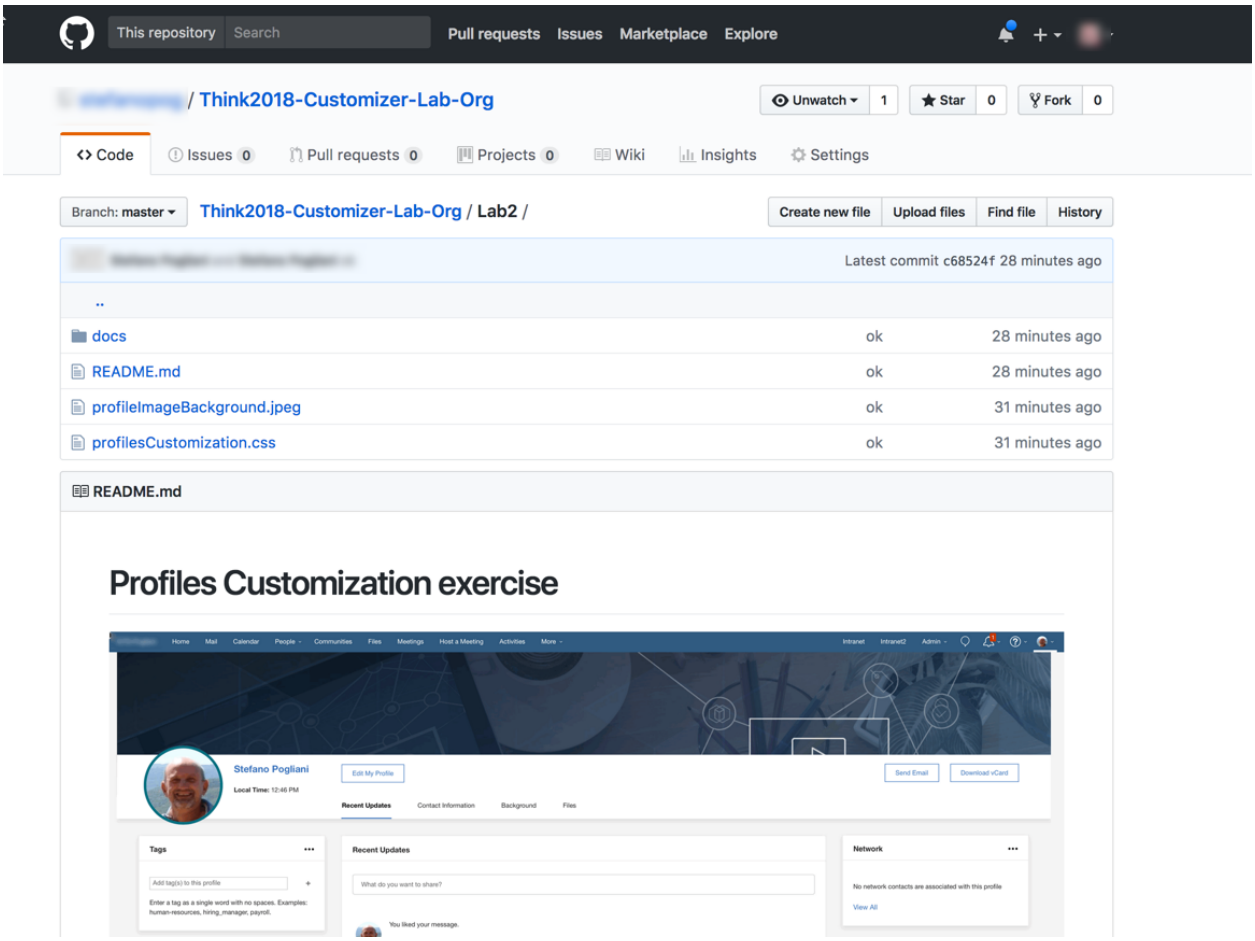
You have been invited as a Contributor to the Github repository associated to this organization.

	The instructor will provide you the URL and the credentials for accessing it.

You will arrive to a page similar to this one.



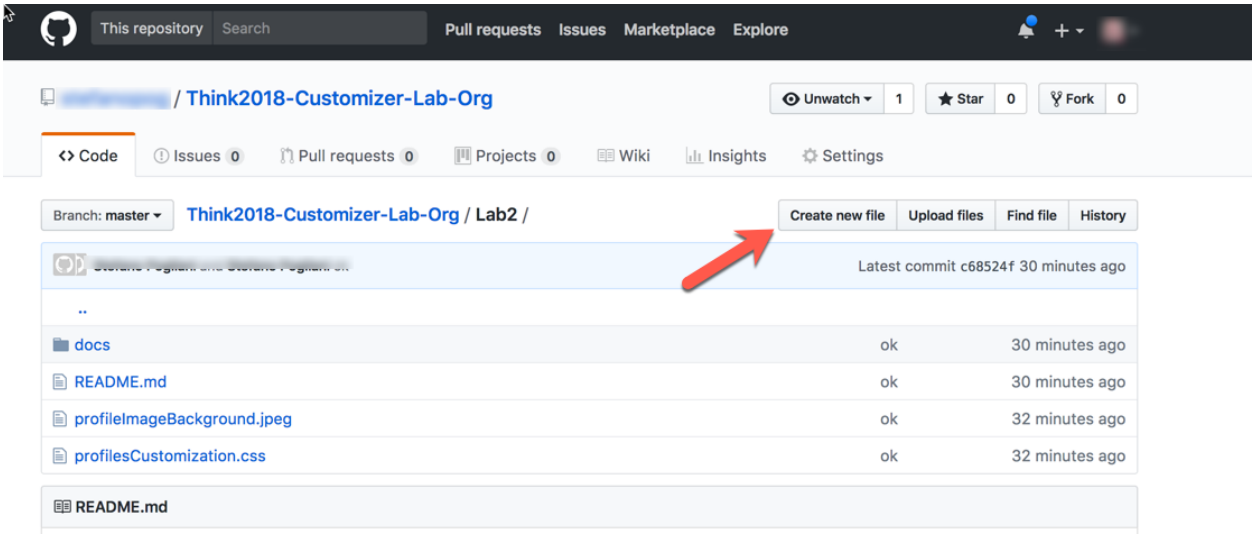
Click on the "**Lab2**" item and you will access this page:



Now you are ready to code!

Creating your script

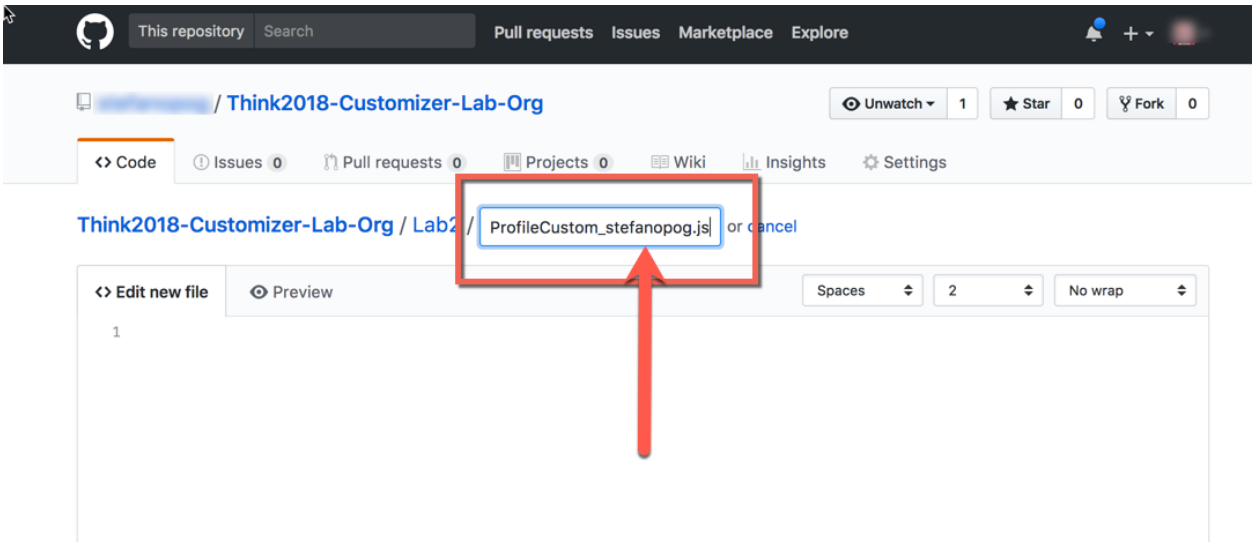
Create the Javascript file that will contain the code for your extension. Click on “Create File” as shown in the following image:



Now, you have to enter the name of the file. Since there are many people doing the exercise at the same time, we need to avoid conflict in filenames. For this reason, enter the following for the filename:

ProfileCustom_<identifier>.js

<identifier> is a unique label that will be provided to you by the instructor (it is going to be the first part of the email address associated with your IBM Connections Cloud account for this lab).



Let's now proceed step-by-step in building our code together.

- The script will need to upload a CSS file.
  The CSS will need to be placed in the <head> element of the HTML page so that the browser will load it immediately and will apply it to the page.
  The dojo way to do it is the following:

  ```
  dojo.place('your HTML element', dojo.doc.head, 'last');
  ```

  where:
  - "your HTML element" will be the <link> element referencing the script.
    In our case it will be :

    ```
    '<link rel="stylesheet" type="text/css"
    src="/files/customizer/Lab2/profilesCustomization.css">'
    ```

    Note the path for the "src" qualifier: "`/files/customizer/Lab2`" is the URL where the artefacts we create in our Github repository will be stored after the validation phase.

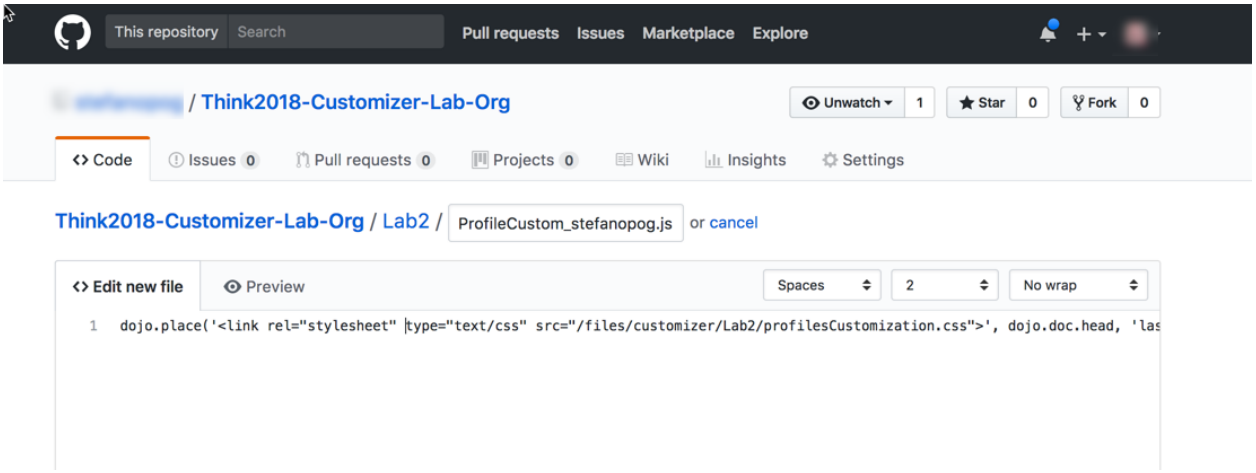
	Note the use of the file "`profileCustomization.css`". As previously stated this file has already been placed in your Directory. It contains the CSS classes that will build the new User Interface for the Profiles page. Feel free to explore it in order to understand how the CSS classes have been used, but **please do not modify** it.

- dojo.doc.head is the handle to the <head> element provided by dojo
- "last" informs the dojo.place to place the new element as the last child of the <head> element

The final form of our statement will be:

```
dojo.place('<link rel="stylesheet" type="text/css"
src="/files/customizer/Lab2/profilesCustomization.css">', dojo.doc.head, 'last');
```

- At this point, your script looks like this:



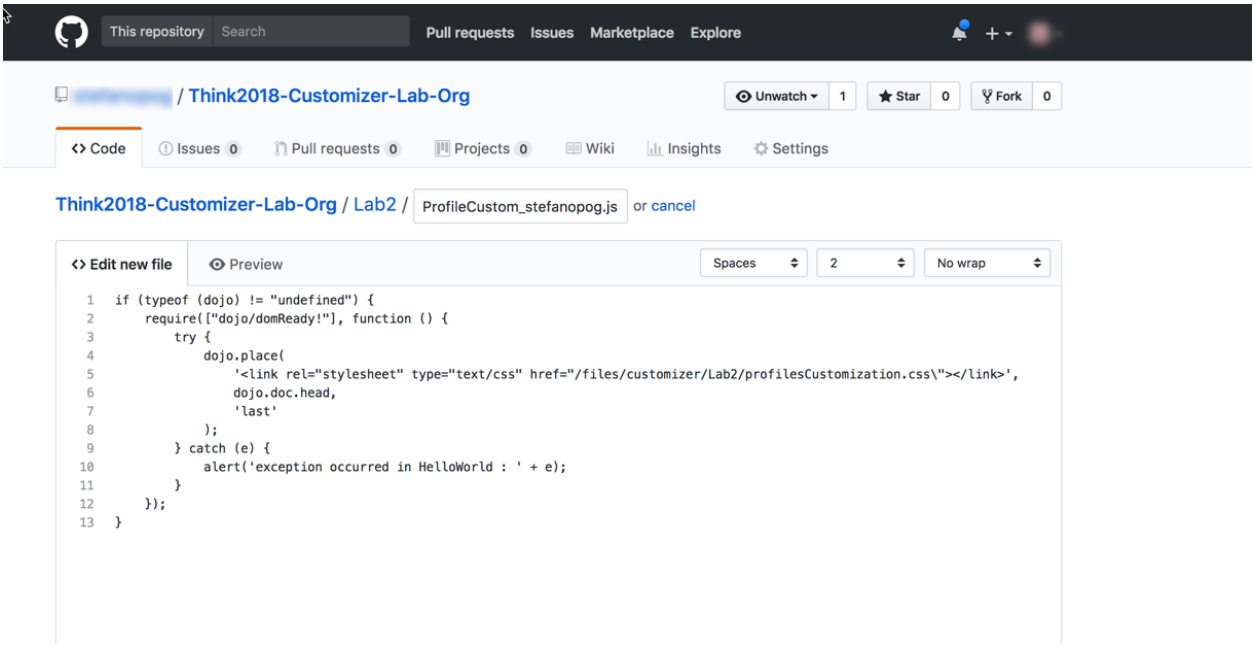
- We are still missing one point though.
  In the code we wrote we assumed that the dojo toolkit would be available for the page in order for your script to use it. We are confident that the IBM Connections page will load the dojo toolkit at a certain point, but we cannot assume that it will be loaded by the time we first use it
  We certainly do not want our script to load the dojo toolkit another time…. So we need to wait for it to be ready on the page.

  Fortunately the dojo toolkit provides a standard mechanism to deal with this point: the "dojo/domReady!" plugin (described here https://dojotoolkit.org/reference-guide/1.10/dojo/domReady.html).

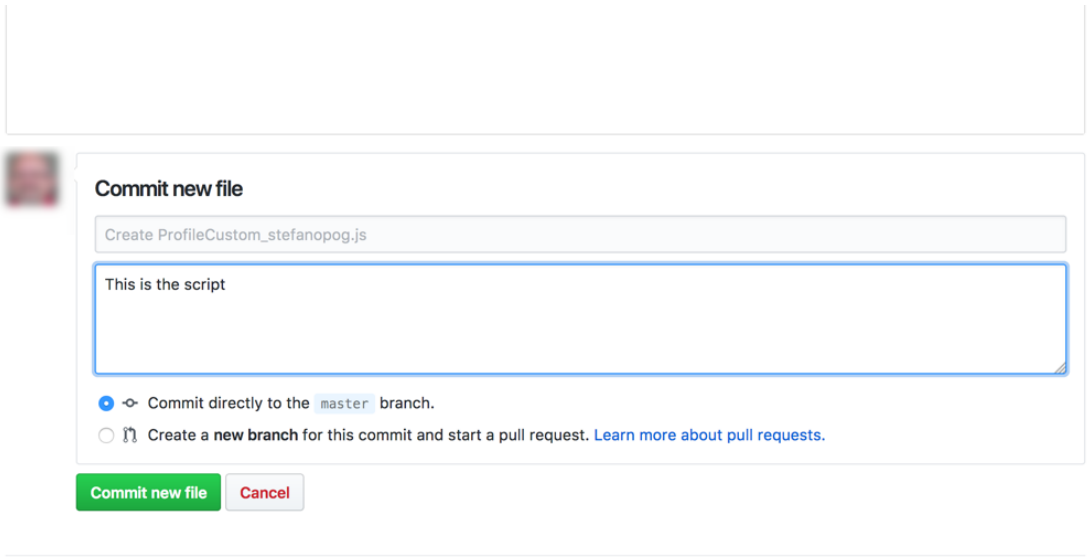
So, we will simply need to wrap the code we wrote up until now in this way:

```
if (typeof(dojo) != “undefined”) {
    require([“dojo/domReady!”], function () {
          try {
                … all our previous code ….
        }  catch(e) {
              alert(‘exception occurred in HelloWorld : ‘ + e);
        }
   });
}
```

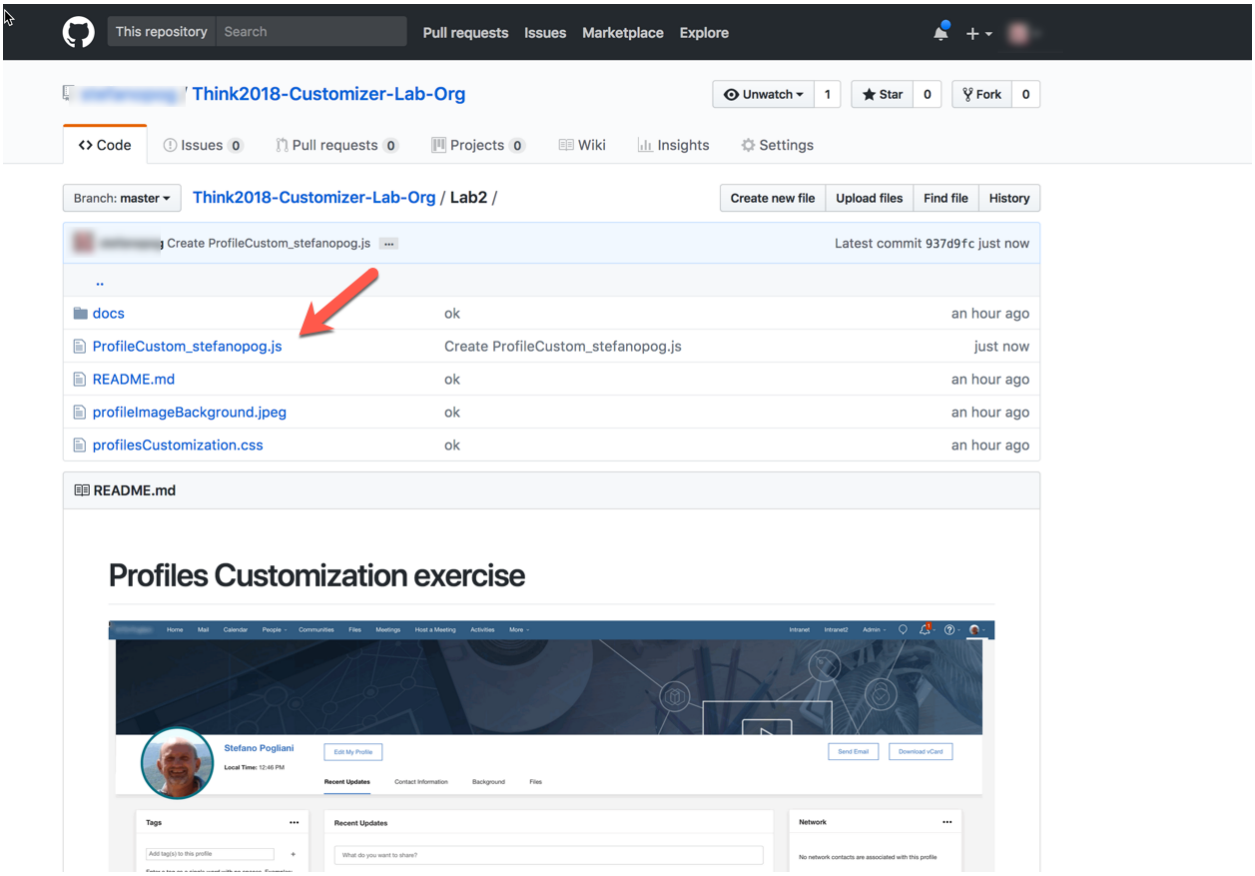
We have introduced a “try… catch” statement to globally catch any exception we may encounter.
At the end, the code in your editor should look like this:



- You can now commit your code to the repository. It is always good practice to add, as a comment, what you did.



- Your script is now safely recorded in the Github repository.



Creating your extension

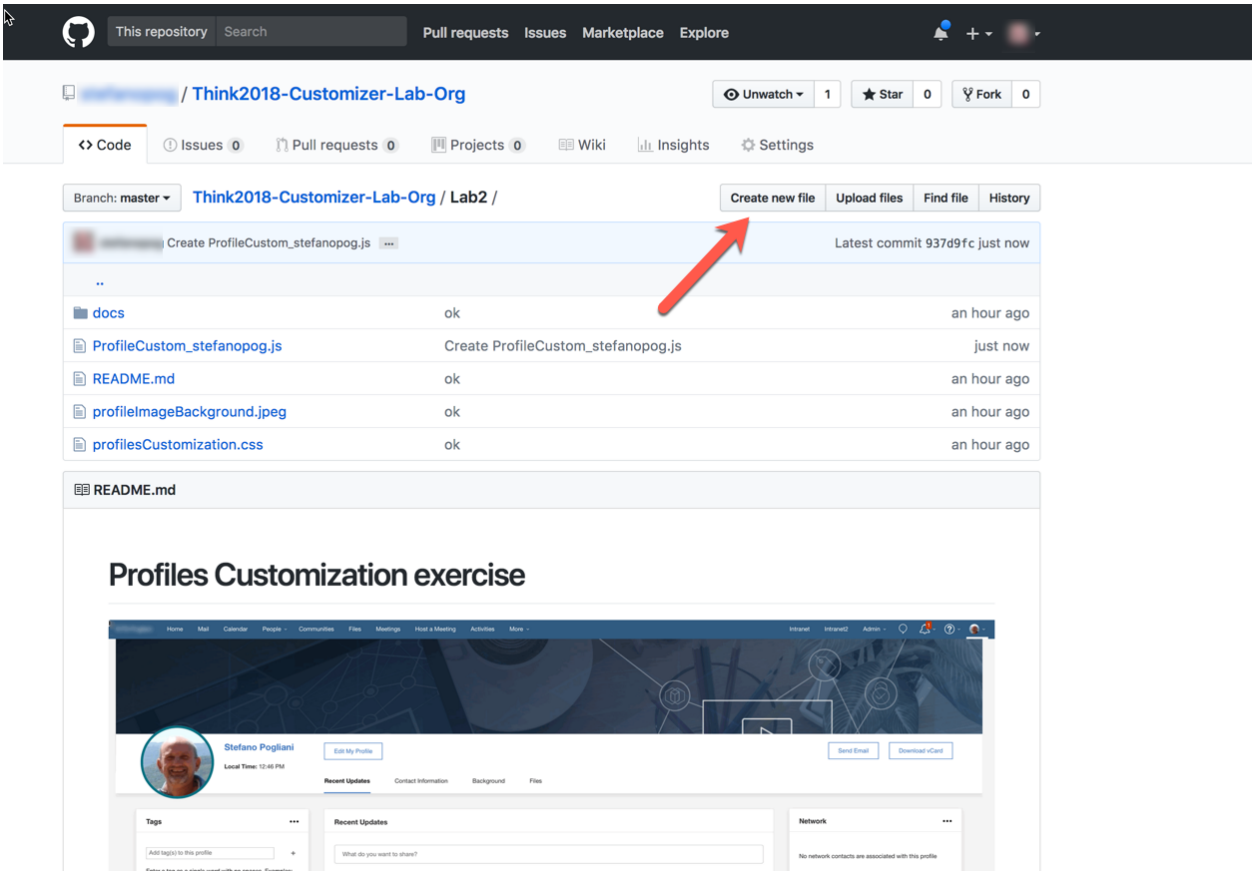
Whilst the extension can be directly created in the IBM Connections Cloud Administration panel, we suggest that you save the extension definition as a JSON file in the same Github repository.

	In this way, the extension may be used in other situations and may become part of the overall documentation of the project

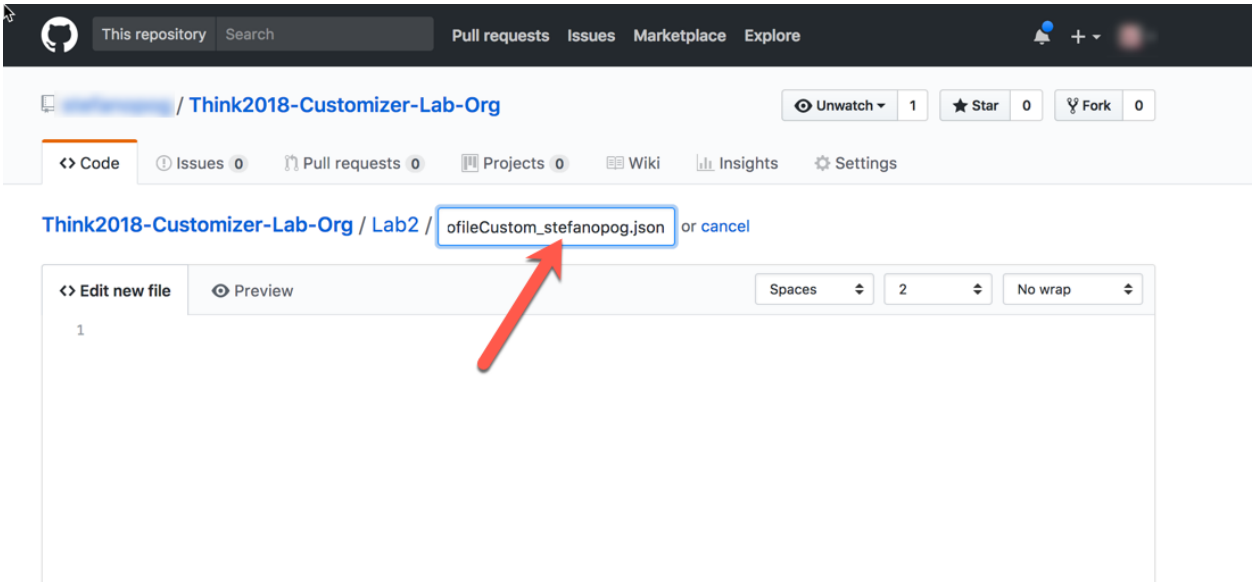
In order to create the extension, we need to create a new file inside our Github repository. The file will follow this naming convention

ProfileCustom_**<identifier>**.json

Where <identifier> is a unique label that will be provided to you by the instructor (it is going to be the first part of the email address associated with your IBM Connections Cloud account for this lab). In this way, everybody will immediately understand that this is the extension definition associated with the ProfileCustom_**<identifier>** script.



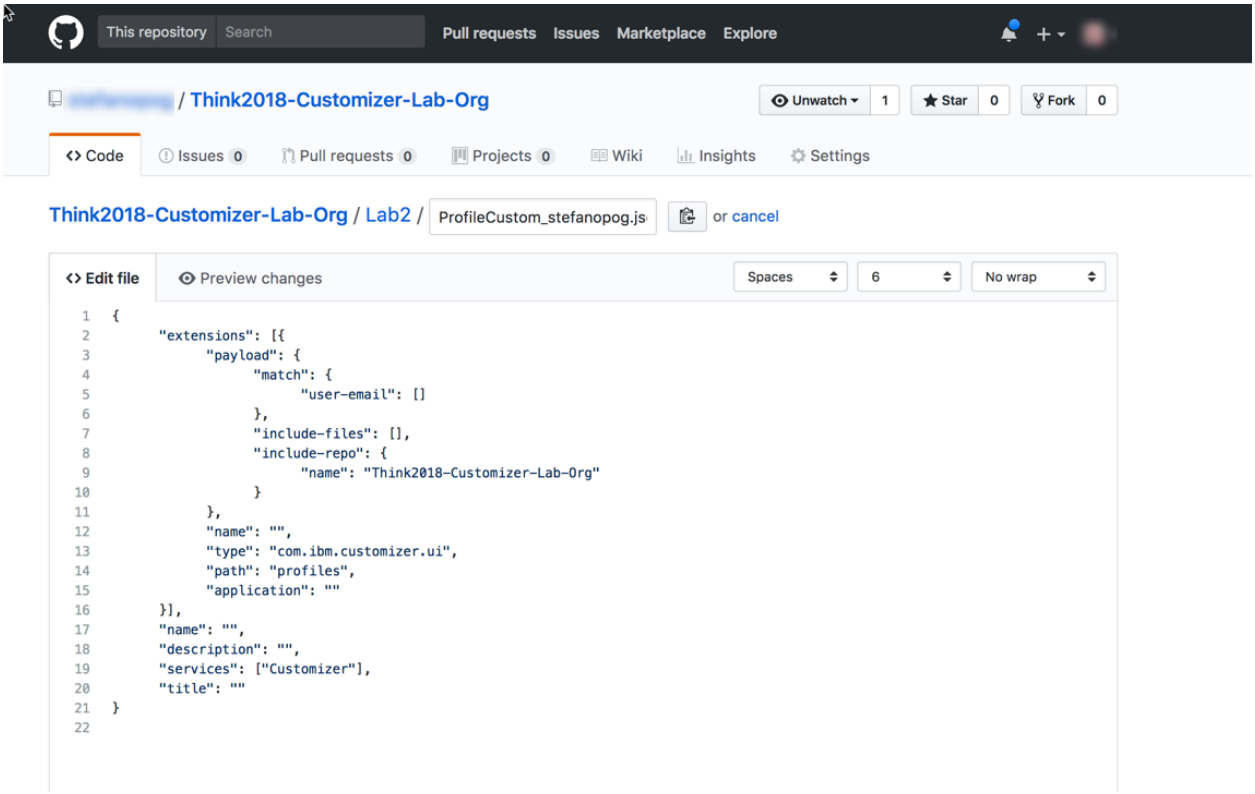
And, then,



Let's first copy this empty template inside our new json file (a copy of this template is available as the `Lab2/emptyTemplate.json` under "Reference Repository"):

```
{
    "extensions": [{
                "payload": {
                        "match": {
                                "user-email": []
                         },
                        "include-files": [],
                        "include-repo": {
                               "name": "Think2018-Customizer-Lab-Org"
                        }
                 },
                 "name": "",
                 "type": "com.ibm.customizer.ui",
                 "path":"profiles",
                 "application": ""
           }],
      "name": "",
      "description": "",
      "services": ["Customizer"],
      "title": ""
}
```

So you will get this result:



Some of the values of the JSON descriptor are already provided. A complete documentation for the attributes is available at this URL : https://github.com/ibmcnxdev/customizer/blob/master/docs/IBMConnectionsCustomizer.pdf

- “include-repo”: (line 8)
  This MUST be the name of the Github repository that we are using. It is the “root” directory that the IBM Connections Customizer engine will use in order to retrieve the scripts that the extension uses
- “type”: (line 13)
  This is a string whose value “com.ibm.customizer.ui” will tell IBM Connections Cloud which type of IBM Connections Customizer service will be used by the extension
- “path”: (line 14)
  The value of “profiles” will instruct the IBM Connections Customizer engine to add the scripts referenced by the extension only to the “profiles” service of IBM Connections
- “services”: (line 19)
  This is a string whose value “Customizer” will tell IBM Connections Cloud which type extension it needs to register in the Application Registry.

Let’s now fill the other values that are required:

- “user-email”: (line 5)
  We can restrict the injection of the scripts associated to the extension only to users

whose email address will match the string.
Since we do not want to interfere with other people working at the same time on the same IBM Connections Cloud organization, we need to provide a value for this attribute. The value needs to be the email address you have been provided by the instructor to log into IBM Connections Cloud (enclosed in double-quotes). So, something like:

```
"user-email": ["stefanopog@think2018-Customizer.com"]
```

	In real life, you may not need to use this attribute if the extension you are creating will be used by **all the people** in your organization. See the documentation for discovering all the possibilities associated to the "match" clause.

- "include-files": (line 7)
  This is where you will actually reference the script that you created earlier in the exercise.
  The path to reference the script is relative to the Gituhub repository you are using (see the "include-repo" clause described above).
  So in your situation you need to provide the following value :

```
"include-files": ["Lab2/ProfileCustom_<identifier>.js"]
```

- "name": (line 12)
  this is the name of the Extension.
  You can use the following:

```
"name": "Profile Ext Lab2 <identifier>"
```

- "name": (line 17)
  This is the name of the Application we are going to declare to IBM Connections. Typically, an "Application" can declare one or more extensions (in our example, we have one extension only associated with the application).
  This name must be unique for the IBM Connections Organization, since it is the name that will be recognized be the IBM Connections Application registry. You can use the following name:

```
"name": "Profile App Lab2 <identifier>"
```

- "application": (line 14)
  This entry (which is within the "extension") actually needs to reference the correct "Application"). So, the value you will provide for this attribute MUST BE the same as the value you provided for the "name" attribute of the Application (line 17). You can use the following name:

```
"application": "Profile App Lab2 <identifier>"
```

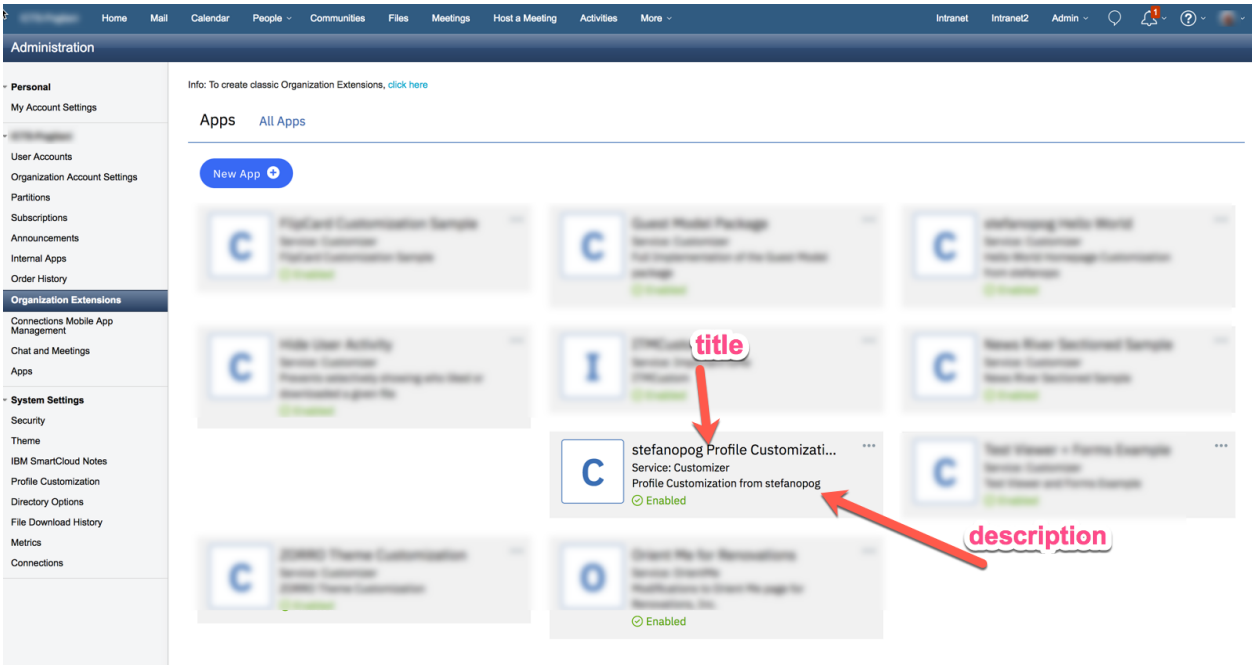
- "description": (line 18)
  This is a free text description of what the Application (and its Extensions) provide. You can use the following:

  ```
  "description": "Profile Customization from <identifier>"
  ```

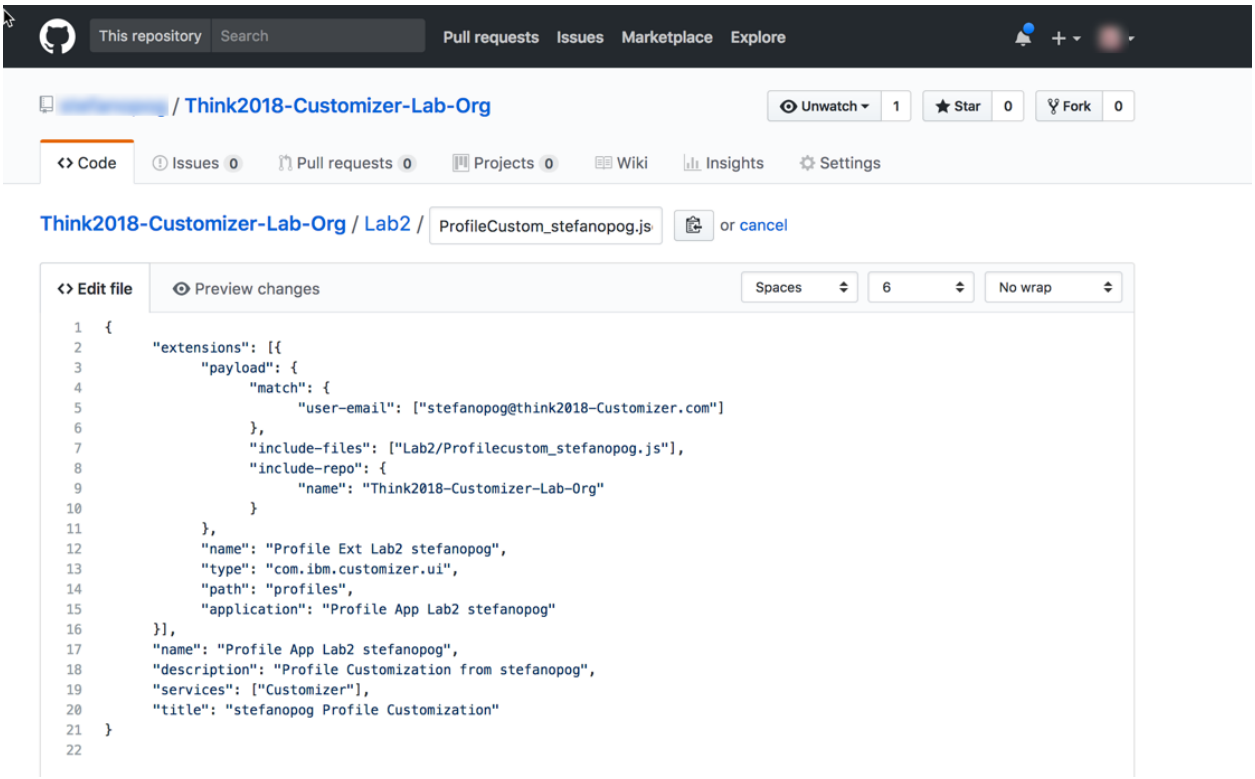
- "title": (line 20)
  This is the title that will appear in the IBM Connections Cloud application registry. You can use the following title:

  ```
  "title": "<identifier> Profile Customization"
  ```

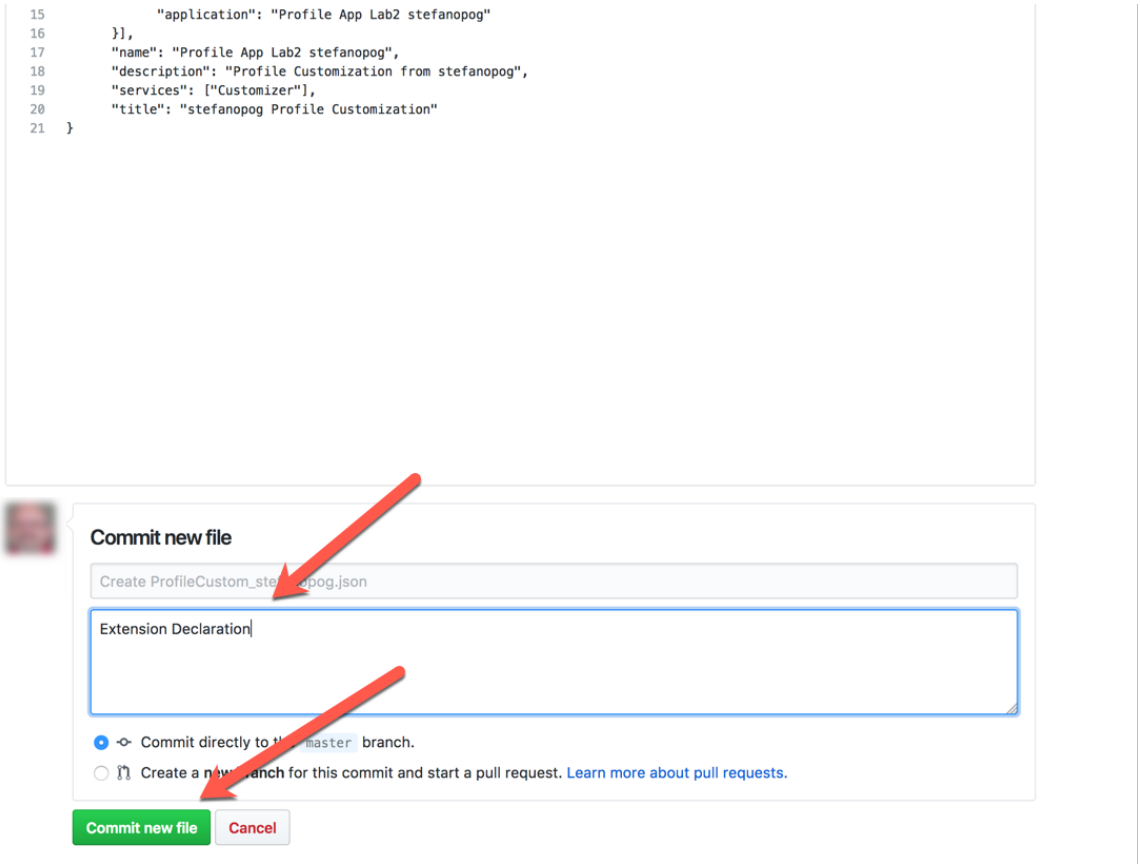
  Once the Organization admin will use your JSON file to declare the extension, this is how it will appear in the IBM Connections Cloud Application Registry:



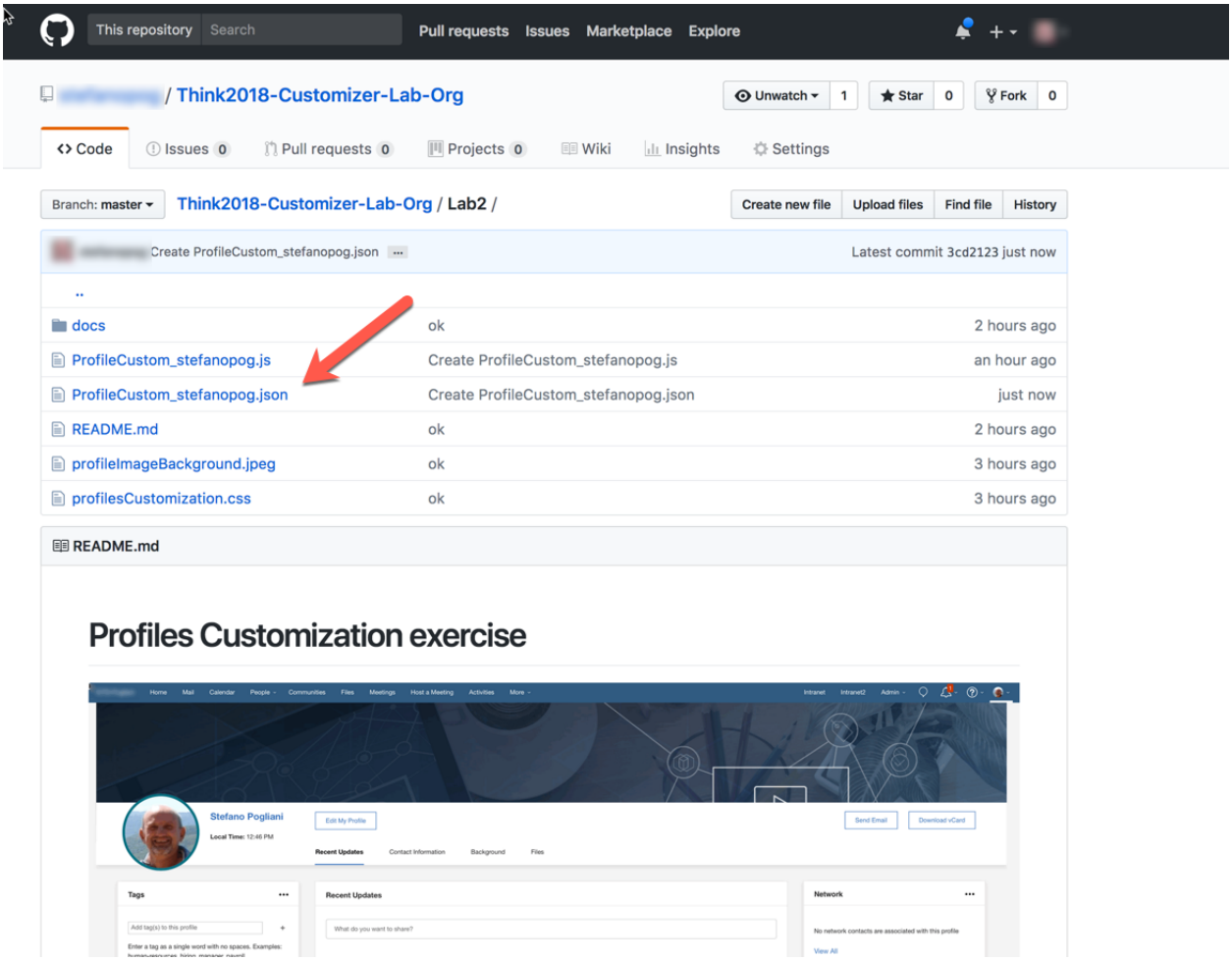
At this point, the JSON file you are editing in Github will look like this:



You can Commit the change to the Github repository:



And your Repository will look like this:



Making your extension available

At this point, everything is ready to be put in production. This is what the documentation states:

a. Share your repo with IBM – add "ibmcndev" as a collaborator

b. IBM (ibmcnxdev) then creates a fork of your repository under github.com/ibmcnxdev and grants you read access by default.

c. You can continue to work on your extension using your original repo for your source code activity, but once you are ready to deliver to IBM Cloud you must issue a pull request to IBM.

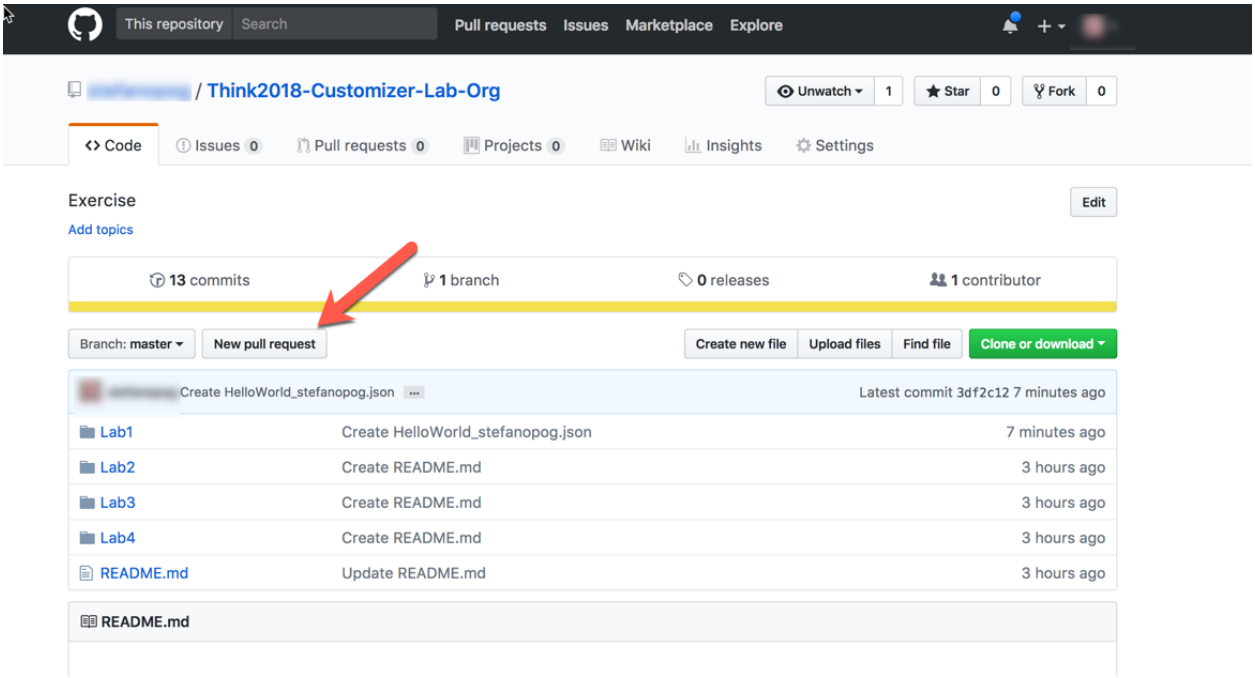
d. IBM merges your pull request once acceptance criteria are met.

e. Upon merge, the repo files are automatically pushed to IBM Customizer via a webhook.

f. Rinse & repeat starting at Step (c) for extension updates.

Steps **a.** and **b.** have already been done for you before you started the exercise.

Now, you need to execute Step **c.**
Go back to the root directory of your Github repository and click on the "Pull Request" button:



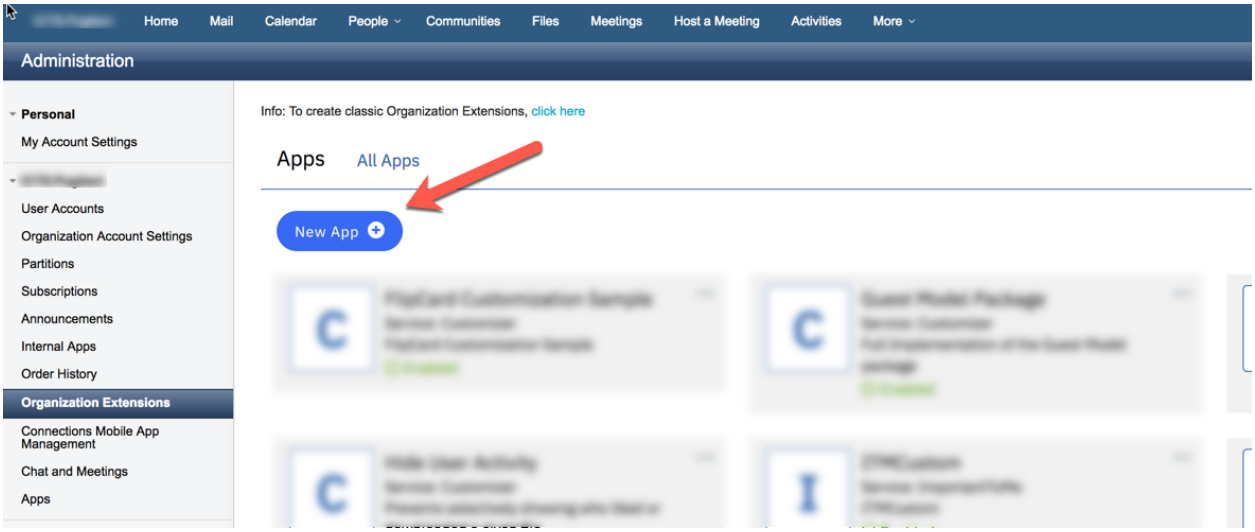
Activating the extension

Now your extension has been validated. The validation process (from **IBMCNDEV**), if successful, will make the files you created in this exercise available to IBM Connections Cloud and to IBM Connections Customizer.

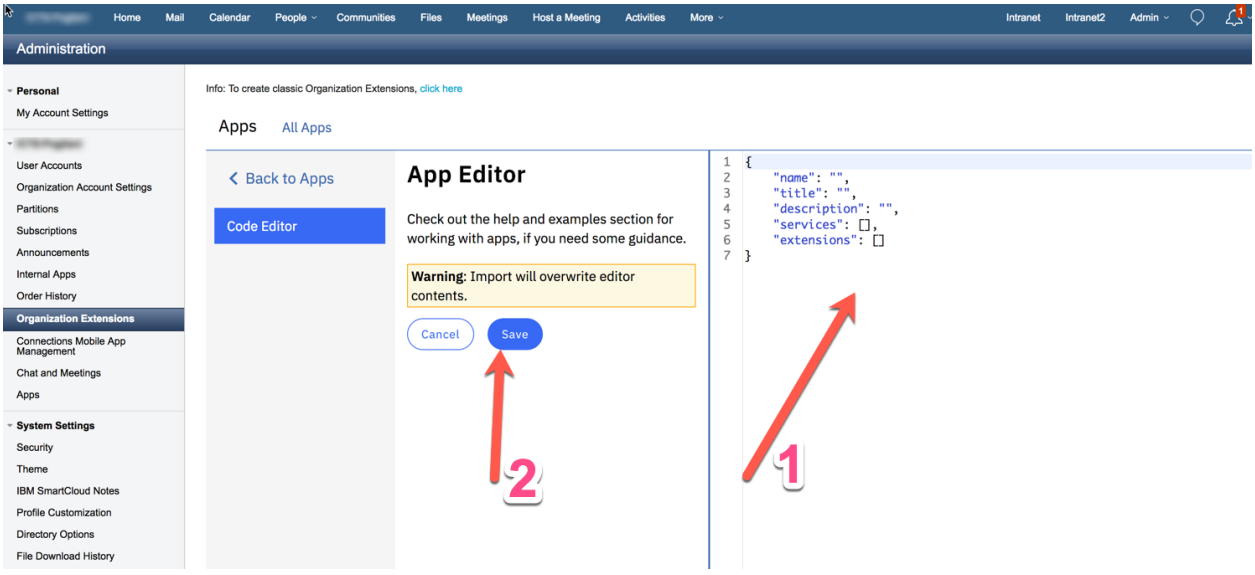
The only thing that is missing is to actually define the Extension in the IBM Connections Cloud Application Registry. This is the step that informs IBM Connections Customizer of what to do with the scripts that you just created.

The IBM Connections Cloud organization administrator needs to be given the JSON file which contains the extension (the one you created here).

He will go to the IBM Connections Application Registry and will declare a new extension:

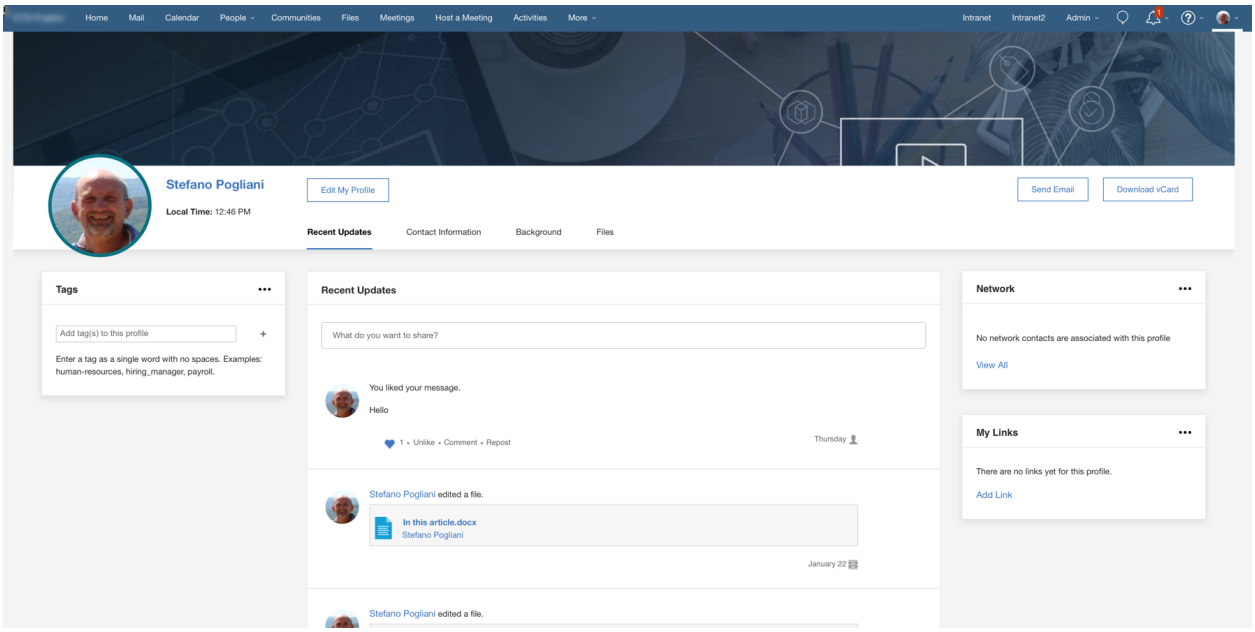


A new screen is shown:



The IBM Connections Cloud organization admin will replace the content pointed by the right arrow (1) with the content of the JSON file you created and, then, saves (2) the extension.

You can now go to the homepage and see the final result:


Home
Mail
Calendar
People
Communities
Files
Meetings
Host a Meeting
Activities
More
Intranet
Intranet2
Admin
Stefano Pogliani
Local Time: 12:46 PM
Edit My Profile
Send Email
Download vCard
Recent Updates
Contact Information
Background
Files
Tags
Add tag(s) to this profile
Enter a tag as a single word with no spaces. Examples: human-resources, hiring_manager, payroll.
Recent Updates
What do you want to share?
You liked your message.
Hello
1 • Unlike • Comment • Repost
Thursday
Stefano Pogliani edited a file.
In this article.docx
Stefano Pogliani
January 22
Stefano Pogliani edited a file.
Network
No network contacts are associated with this profile
View All
My Links
There are no links yet for this profile.
Add Link

Using CSS Declaration

What we have been describing in the previous section (Using Javascript injection) is the approach which has Javascript to load the CSS file.

Recently, the IBM Connections Customizer engine has been improved in order to directly accept the declaration of CSS files within the Extension declaration. Roughly, you will be able to use a direct reference to the CSS files you may want to apply to a given IBM Connections page from the “include-files” clause of the Extension declaration.

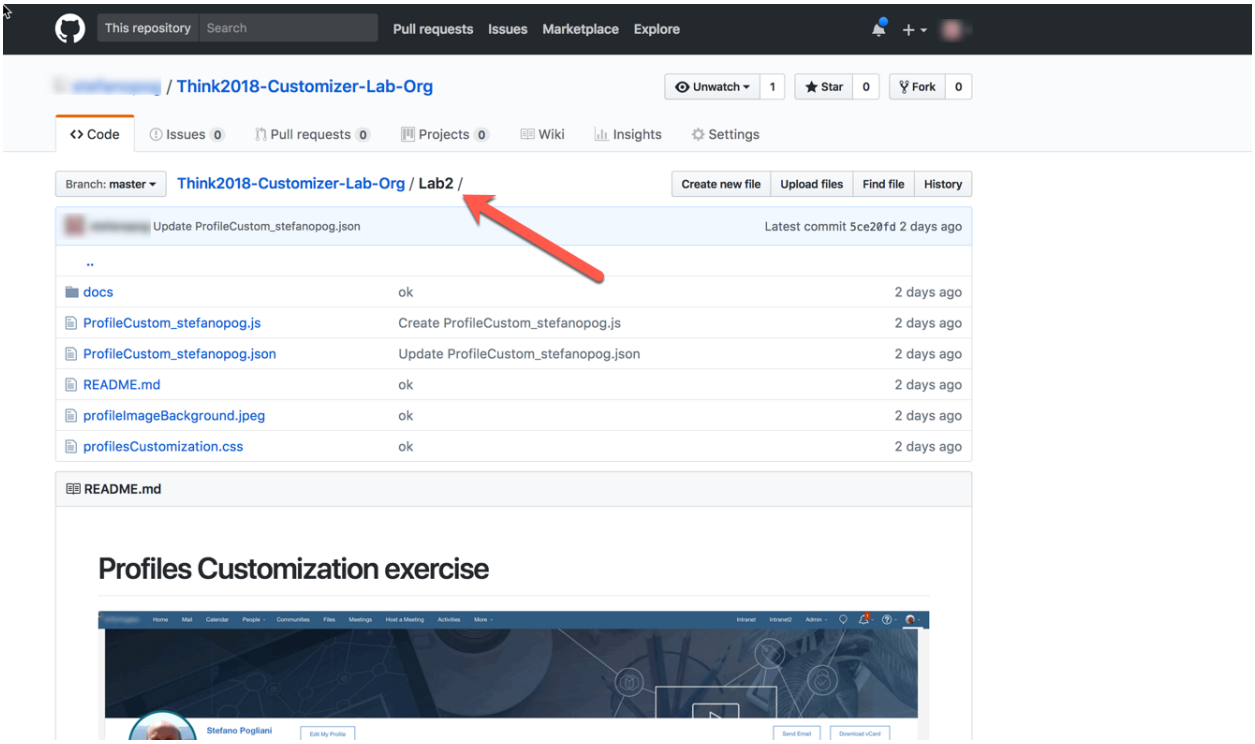
	Obviously, the rendering of the “customized” IBM Connections page will be much quicker and effective

Let’s see together how we can modify what we have previously done in order to make a proper use of this new functionality.

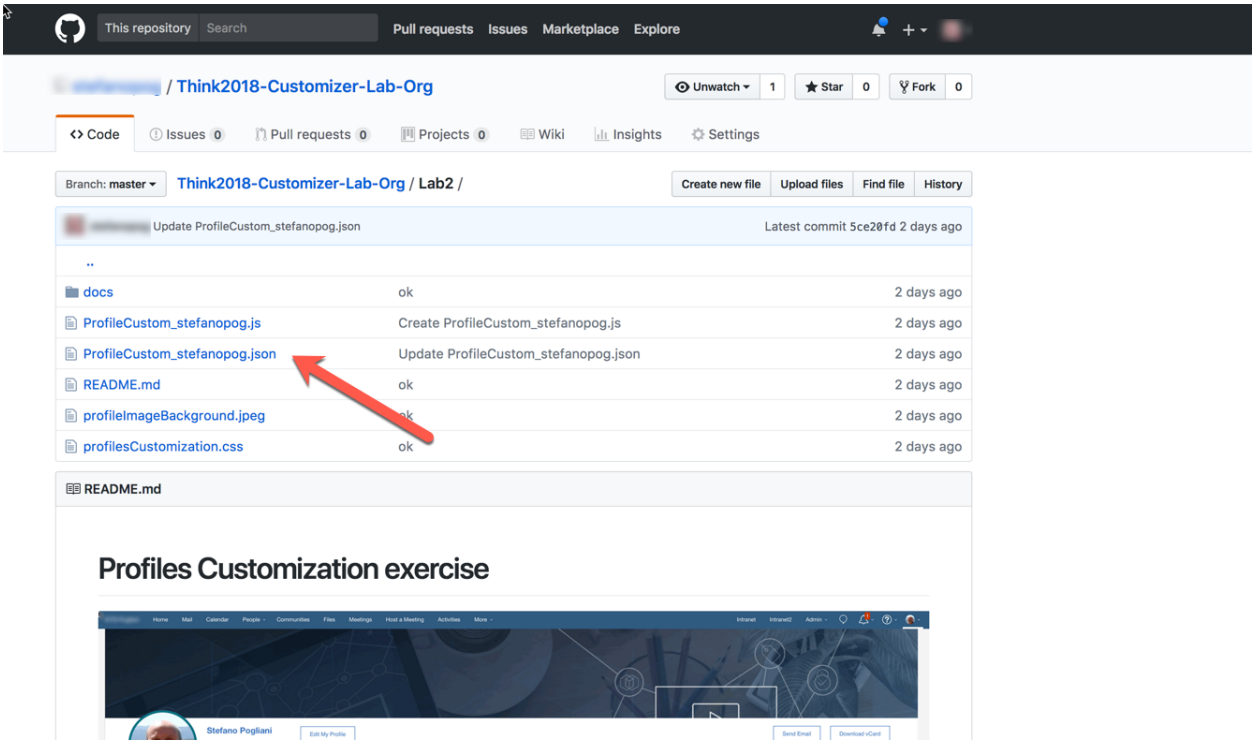
Preparation Activities

What we need to do is, simply, to modify the Extension declaration in order to reference the CSS file instead of the Javascript file.

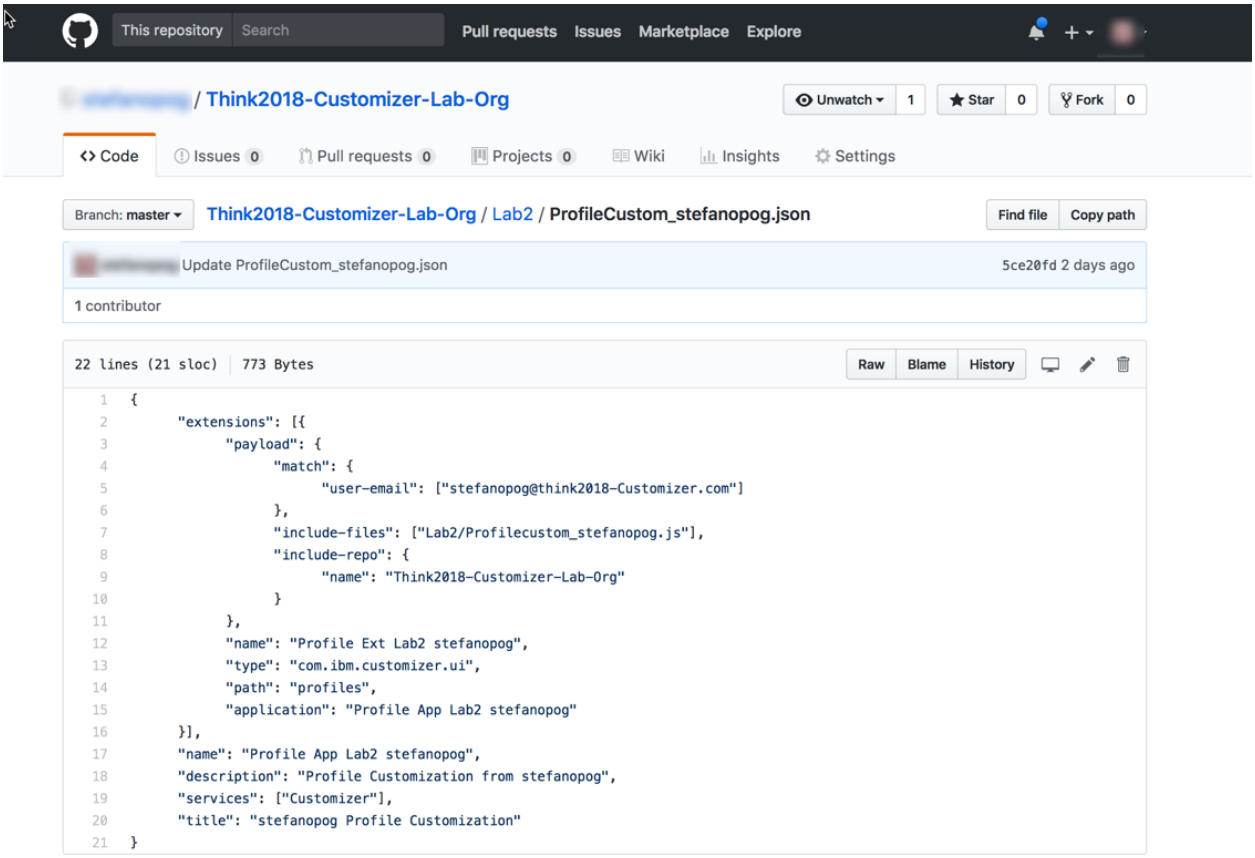
Go to the Github directory associated with the second Lab (the one we just worked on in the first part of this lab).



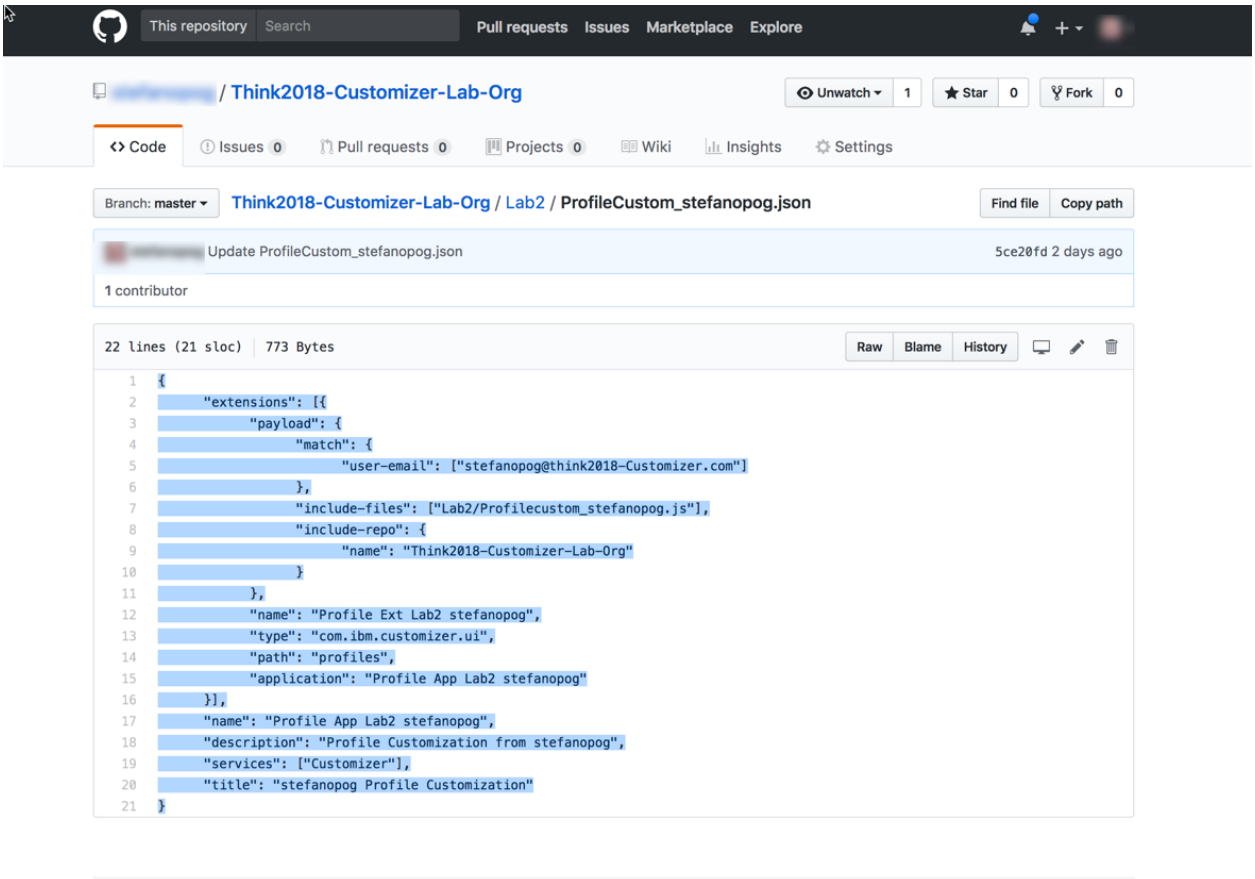
Click on the JSON Extension file your previously created:



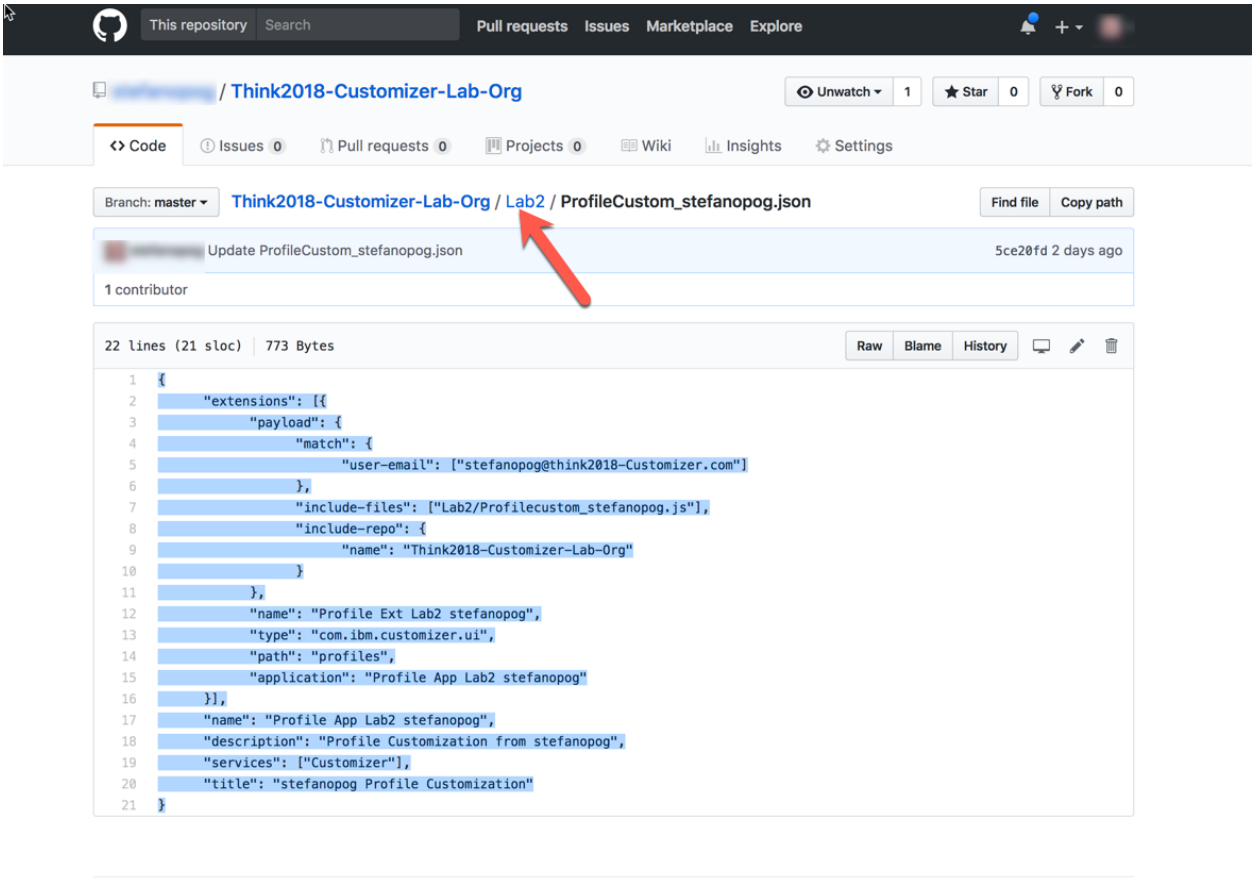
When the “view” on this file opens,



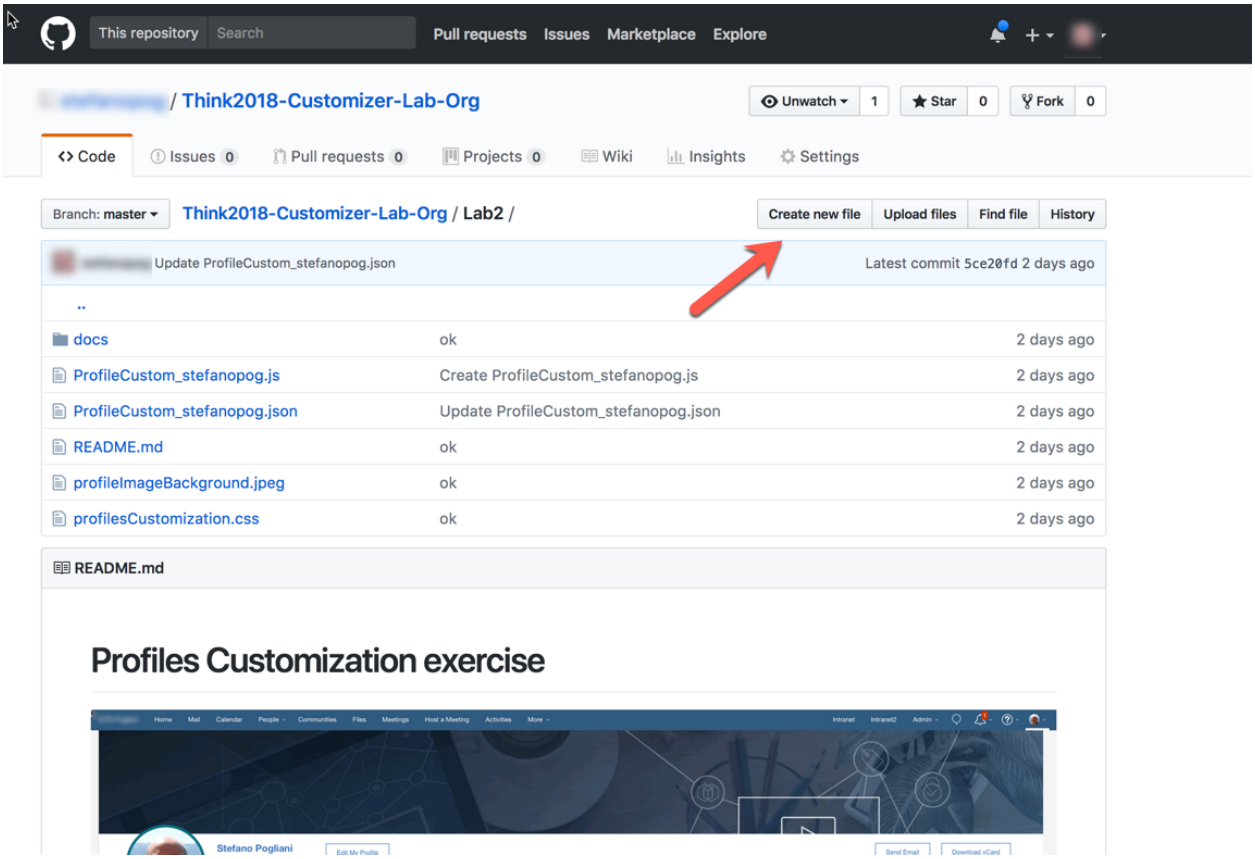
by means of the mouse, select the whole content of the file and, then, issue CTRL-C to copy the selection into the computer clipboard:



Now go back to the Lab2 Directory clicking on the “Lab2” link shown in the picture below:



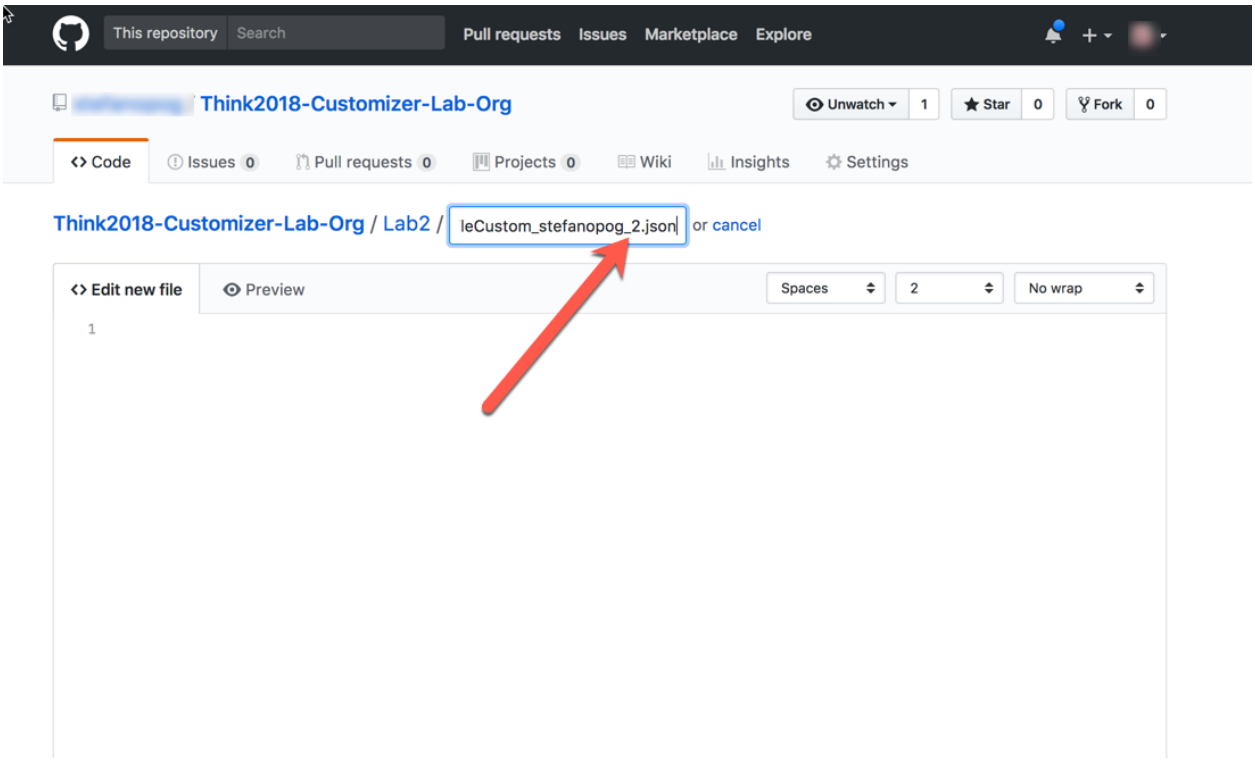
Now create a new file in the Lab2 Directory as shown here:



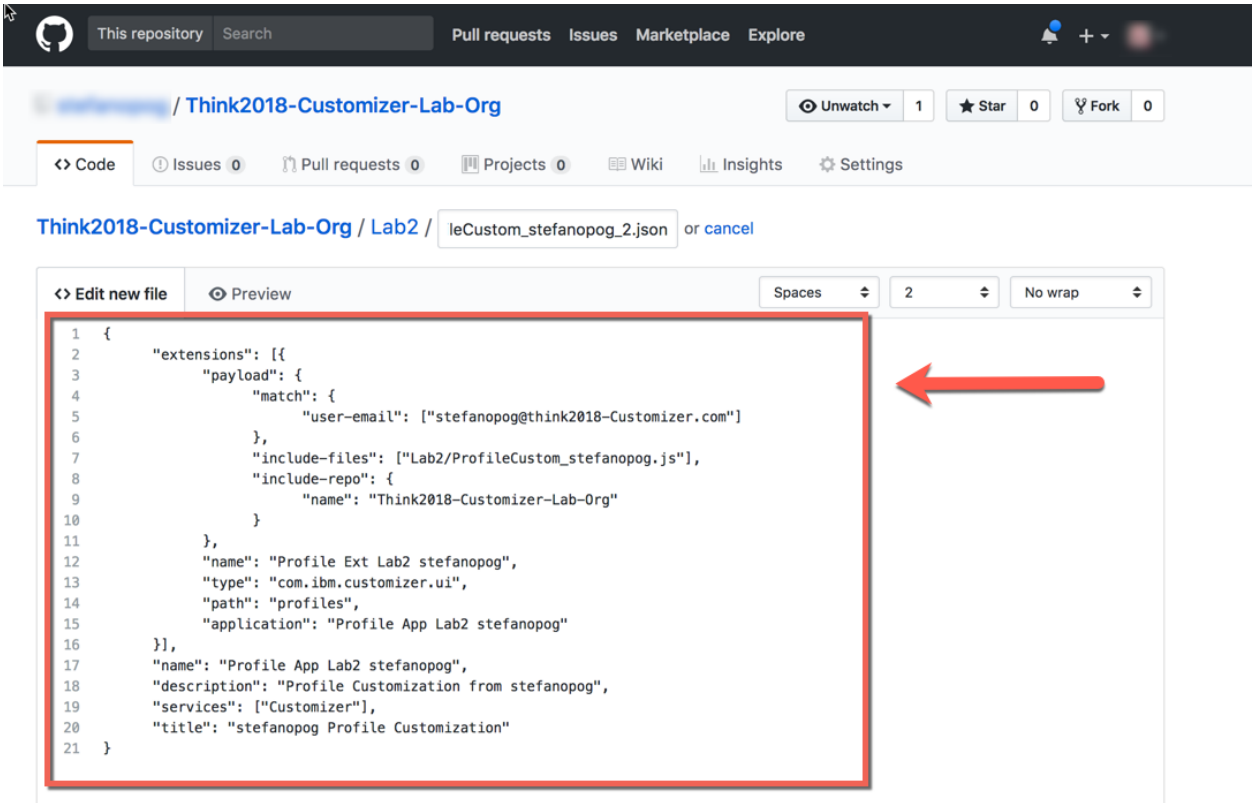
Give the new file the following name:

ProfilCustom_**<identifier>**_2.json

Where <identifier> is a unique label that will be provided to you by the instructor (it is going to be the first part of the email address associated with your IBM Connections Cloud account for this lab). Note that we append a **"_2"** suffix to the file name in order to distinguish it from the other one.



Now, using CTRL-V paste in the canvas what you previously copied to the clipboard:



The only modifications you need to perform are:

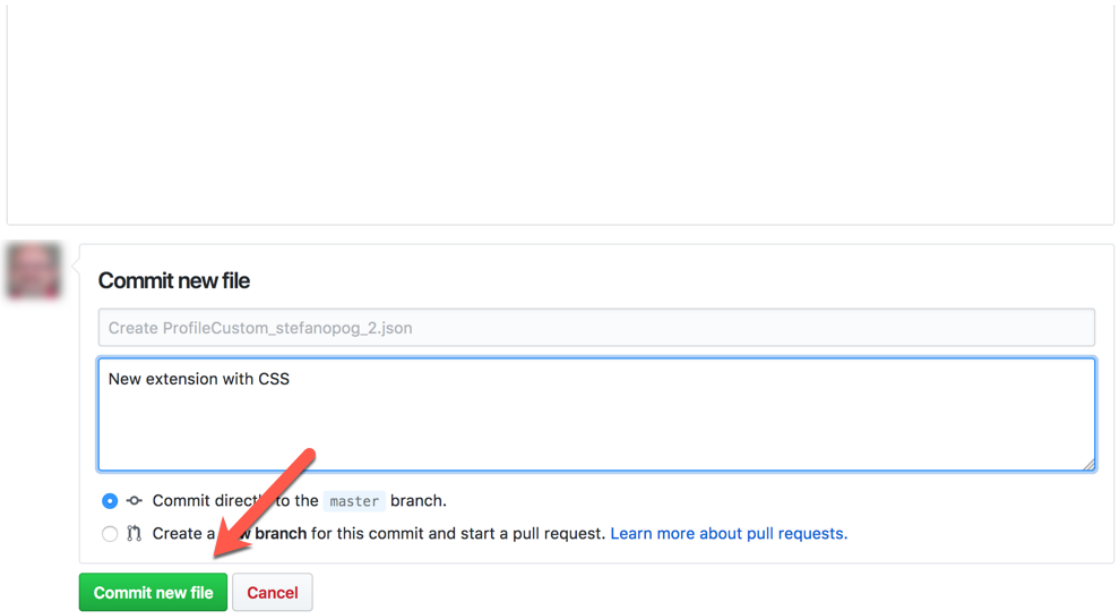
- **at line 7**. You will change:
  `"include-files": ["Lab2/ProfileCustom_stefanopog.js"]`
  with
  `"include-files": ["Lab2/profilesCustomization.css"]`

- **at line 17**. You will change:
  `"name": "Profile App Lab2 <identifier>"`
  with
  `"name": "Profile App 2 Lab2 <identifier>"`

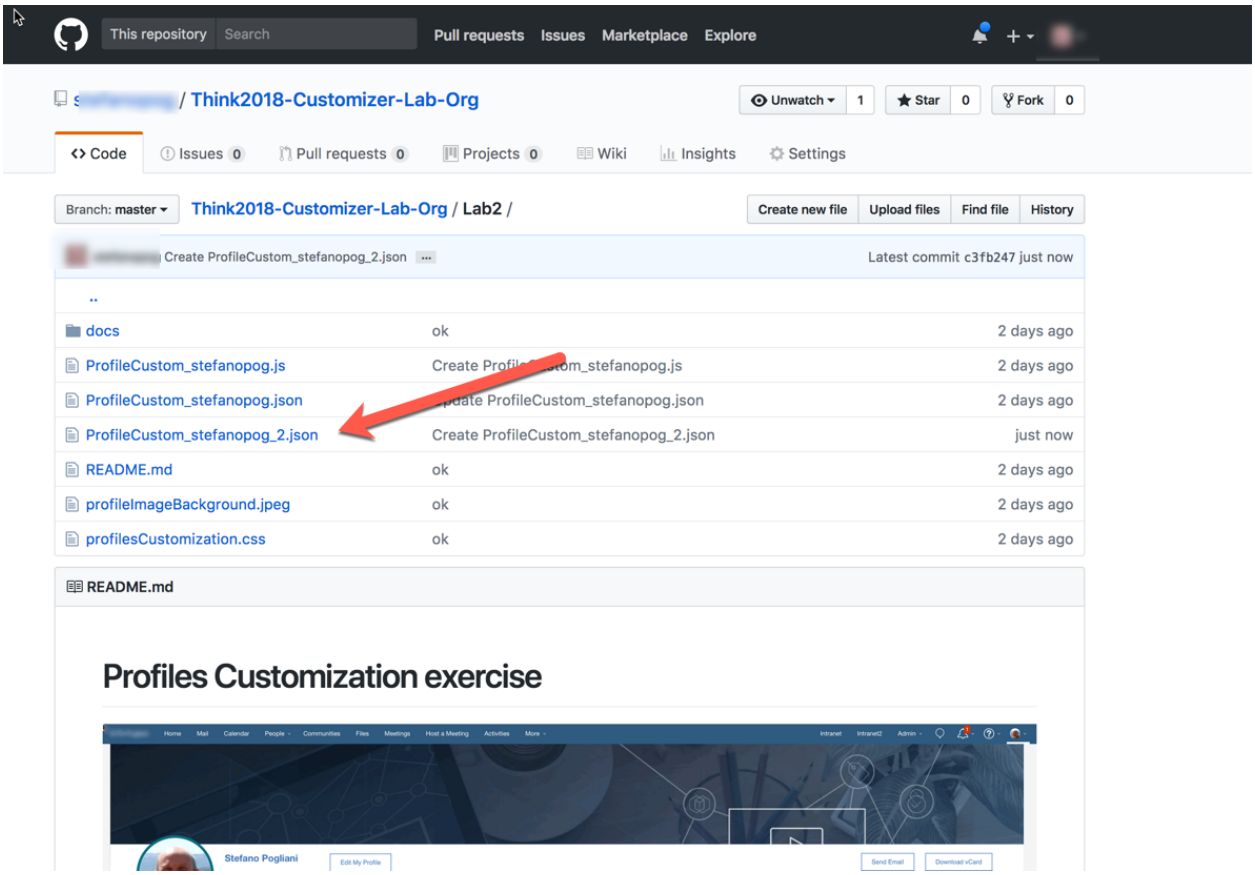
- **at line 15**. You will change:
  `"application": "Profile App Lab2 <identifier>"`
  with
  `"application": "Profile App 2 Lab2 <identifier>"`

Those changes created a new extension (we added a **2** to the name in order to distinguish it from the one we previously created) which loads the CSS file directly.

Then you can commit your change:



And your Repository will look like this:



Making your extension available

At this point, everything is ready to be put in production. This is what the documentation states:

a. Share your repo with IBM – add "ibmcndev" as a collaborator

b. IBM (ibmcnxdev) then creates a fork of your repository under github.com/ibmcnxdev and grants you read access by default.

c. You can continue to work on your extension using your original repo for your source code activity, but once you are ready to deliver to IBM Cloud you must issue a pull request to IBM.

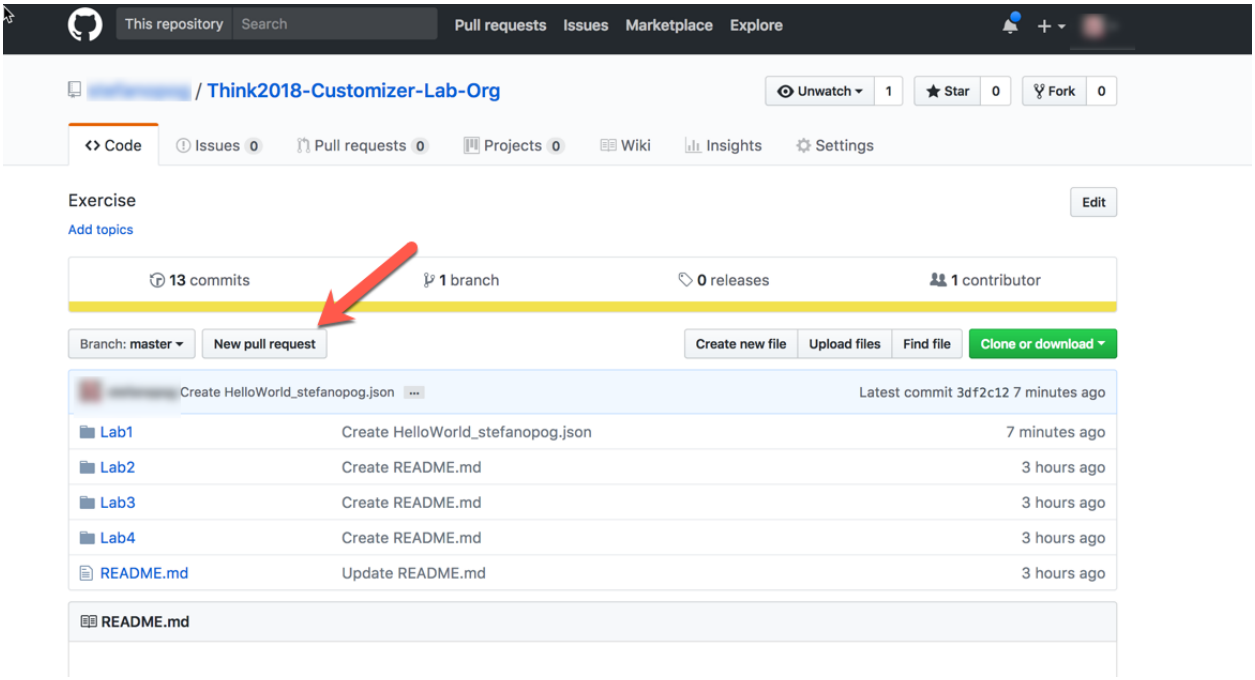
d. IBM merges your pull request once acceptance criteria are met.

e. Upon merge, the repo files are automatically pushed to IBM Customizer via a webhook.

f. Rinse & repeat starting at Step (c) for extension updates.

Steps **a.** and **b.** have already been done for you before you started the exercise.

Now, you need to execute **Step c**.
Go back to the root directory of your Github repository and click on the “Pull Request” button:



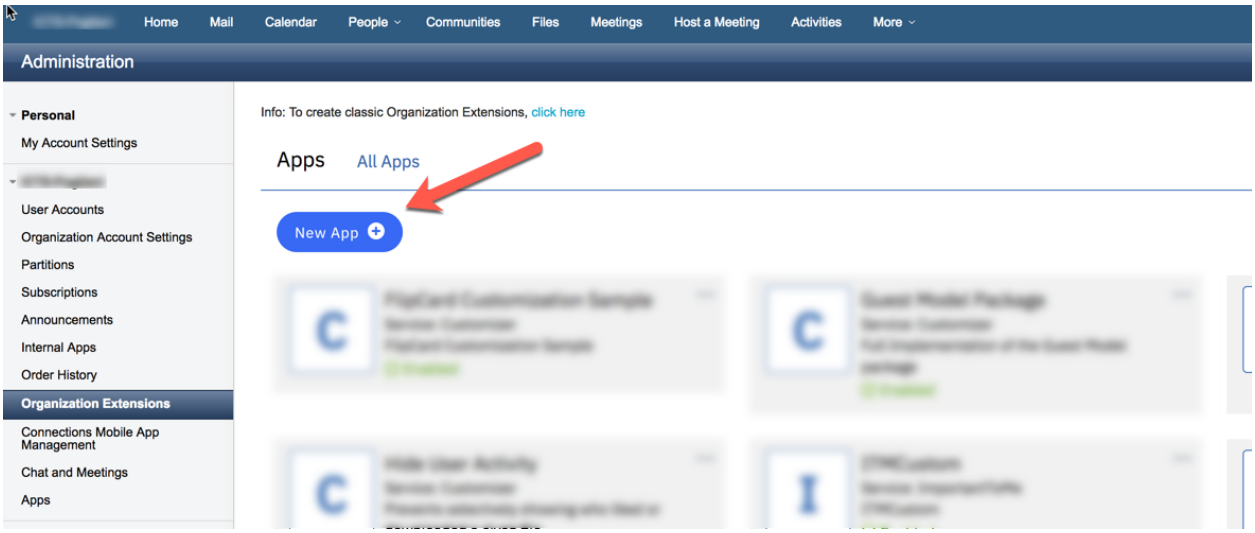
Activating the extension

Now your extension has been validated. The validation process (from **IBMCNDEV**), if successful, will make the files you created in this exercise available to IBM Connections Cloud and to IBM Connections Customizer.

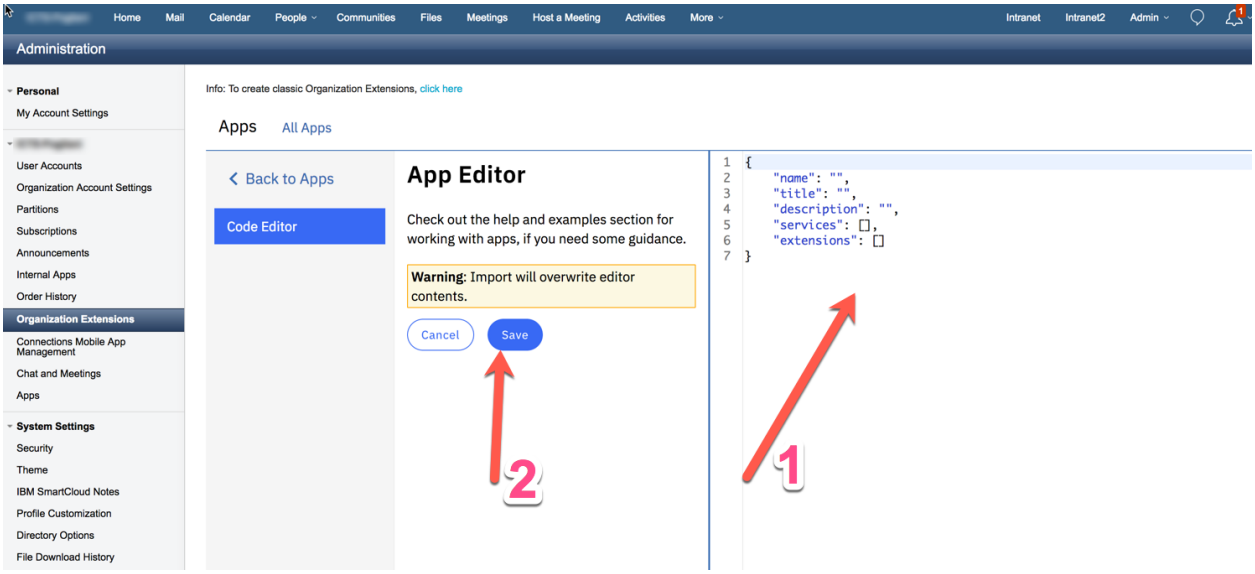
The only thing that is missing is to actually define the Extension in the IBM Connections Cloud Application Registry. This is the step that informs IBM Connections Customizer of what to do with the scripts that you just created.

The IBM Connections Cloud organization administrator needs to be given the JSON file which contains the extension (the one you created here).

He will go to the IBM Connections Application Registry and will declare a new extension:

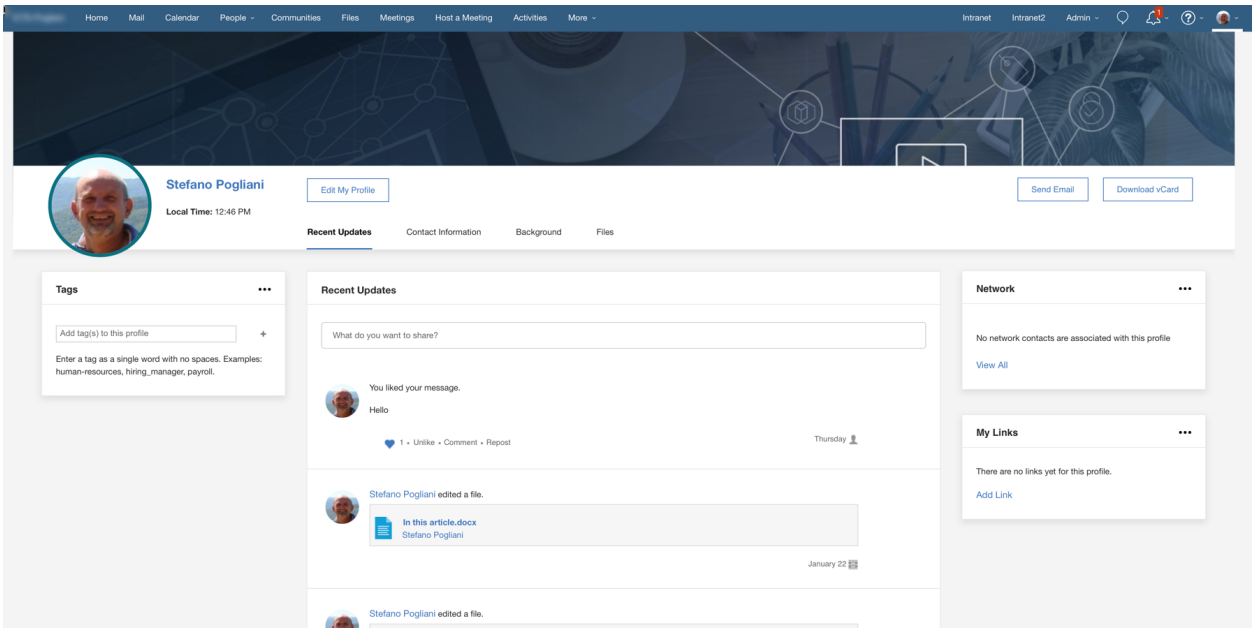


A new screen is shown:



The IBM Connections Cloud organization admin will replace the content pointed by the right arrow (1) with the content of the JSON file you created and, then, **saves** (2) the extension.

You can now go to the homepage and see the final result:



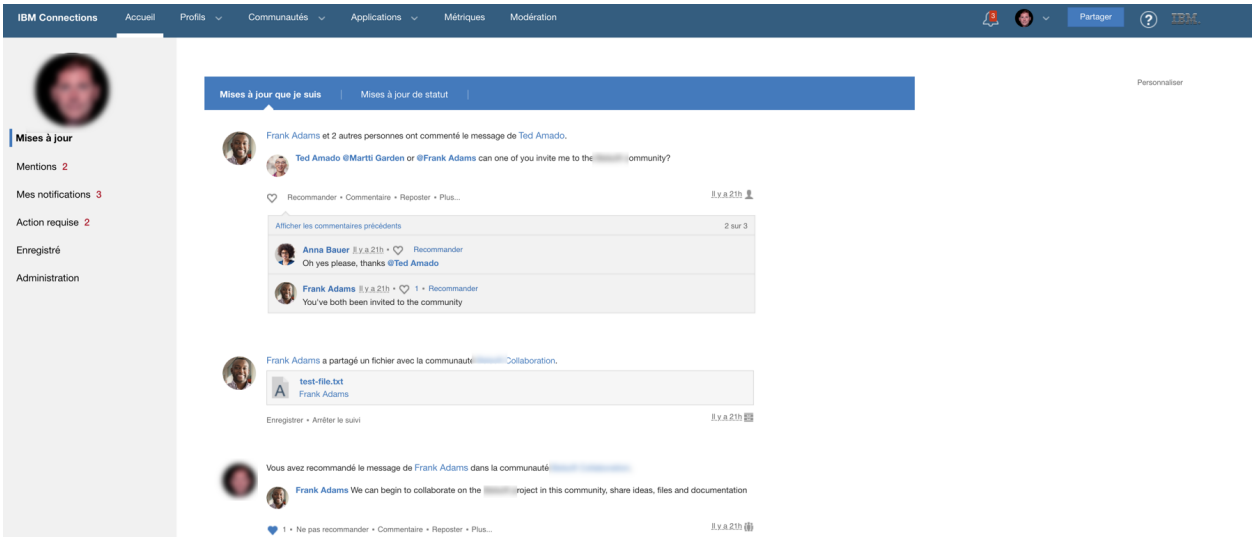
This ends Lab 2.
Congratulations!

Lab 3 Reducing the complexity of Connections UI

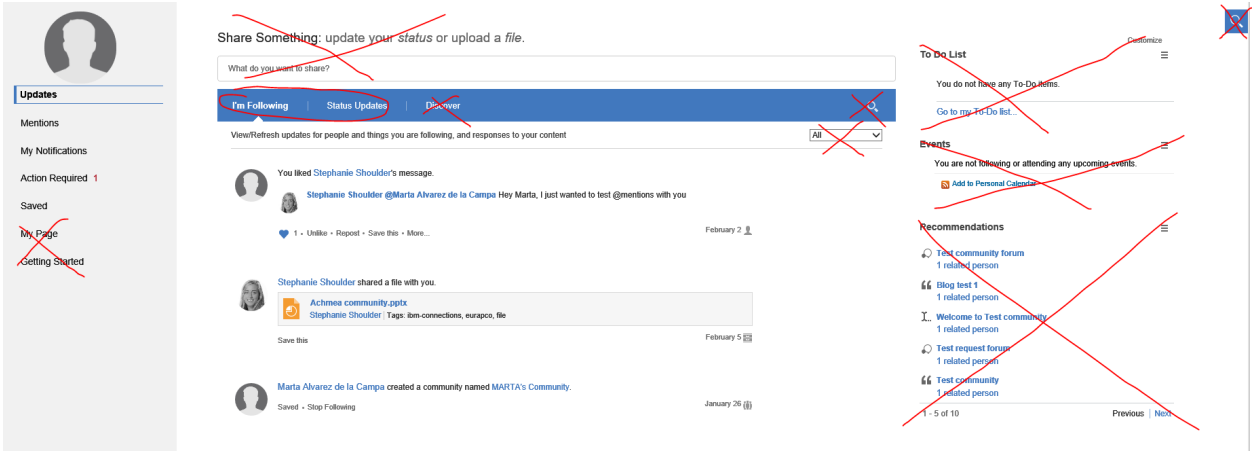
This is a very simple exercise, which continues what we learned in the second part of the Lab 2 exercise.

The main goal here is to reduce the clutter on the Classic IBM Connections Homepage. And we will do using CSS only.

What we want to achieve

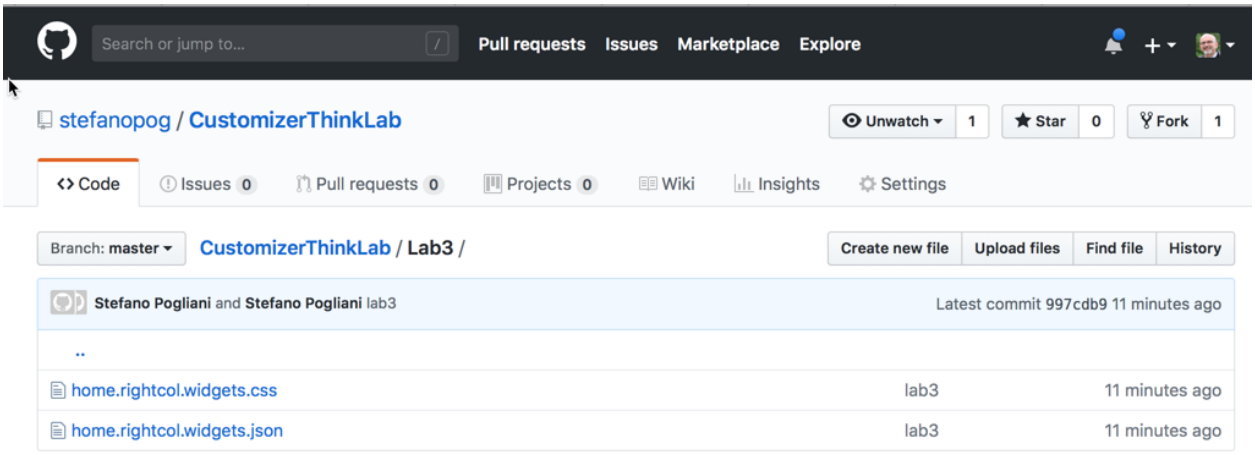


Compare the previous image with the default :

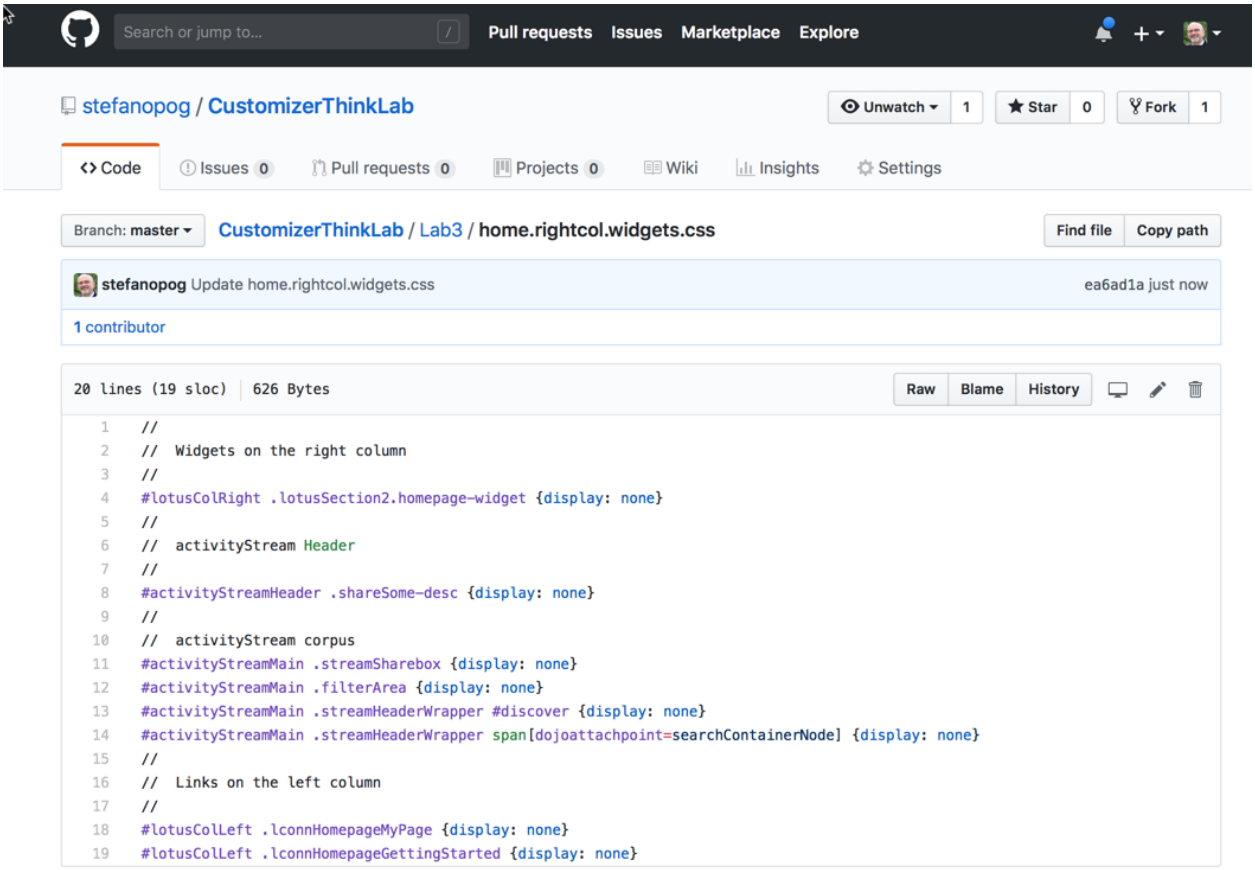


How we did ?

So, you will find at this location Lab3 the repository containing the code.



Let's look at the CSS file first:



Very few lines of CSS made the magic.

- On Line 4 we addressed the Widgets that appear on the right column:

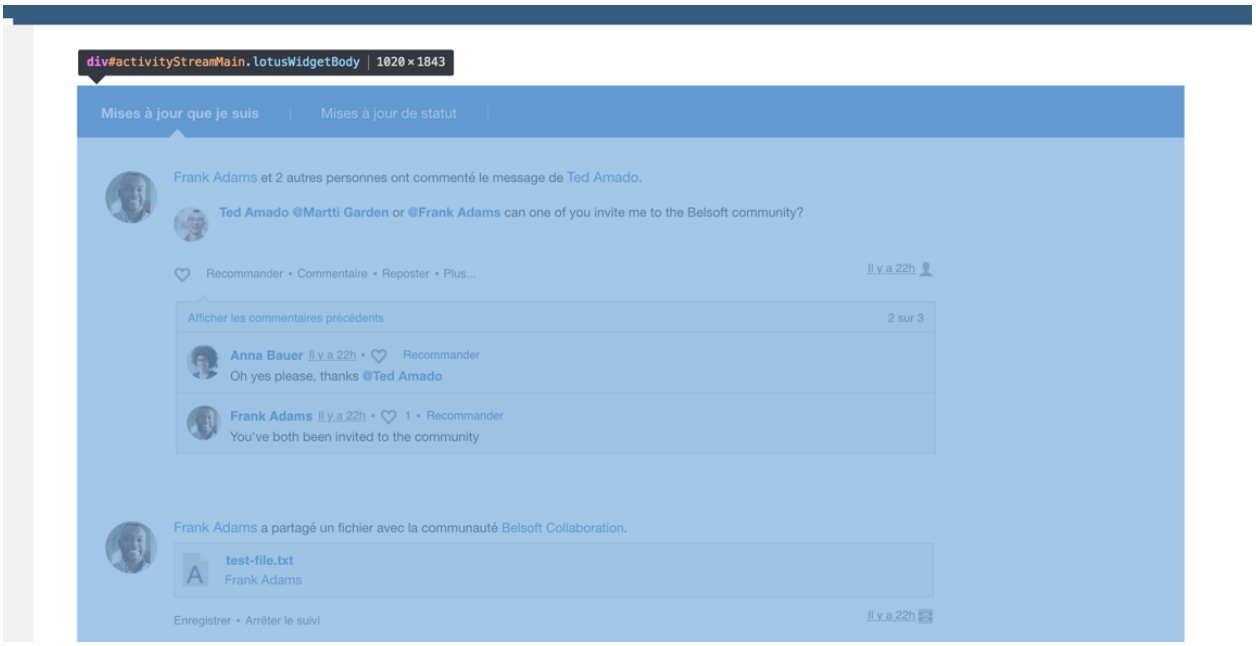
Using the browser Developers Tools, we find that this area corresponds to :

```
<div id="leftColBgPlaceholder" class="lotusHidden leftColBackgroundPlaceholder"> </div>
▼<div id="lotusColRight" class="lotusColRight">
  <div id="hpplaceholder_right_top"></div>
  ▼<div id="1" class="dojoDndContainer dojoDndSource dojoDndTarget">
    ▶<div class="lotusSection2 homepage-widget" role="complementary" aria-labelledby="a7cffd2b-6
    7c1fd3226f52">…</div>
    ▶<div class="lotusSection2 homepage-widget" role="complementary" aria-labelledby="d56915da-7
    d97e58f00b0a">…</div>
    ▶<div class="lotusSection2 homepage-widget" role="complementary" aria-labelledby="73fb9044-d
    13790cf94bd6">…</div>
    </div>
    <div id="hpplaceholder_right_center"></div>
    <div id="hpplaceholder_right_bottom"></div>
```

So with the CSS we just said that we want to hide any instance of class "`lotusSection2 homepage-widget`" which inherits from the widget whose id is "`lotusColRight`"

- The widget whose id is "`activityStreamHeader`" controls the header of the ActivityStream
The widget whose class is "`shareSome-Desc`" controls this area:

- The Widget whose id is "`activityStreamMain`" actually encompasses all the area shown here below:



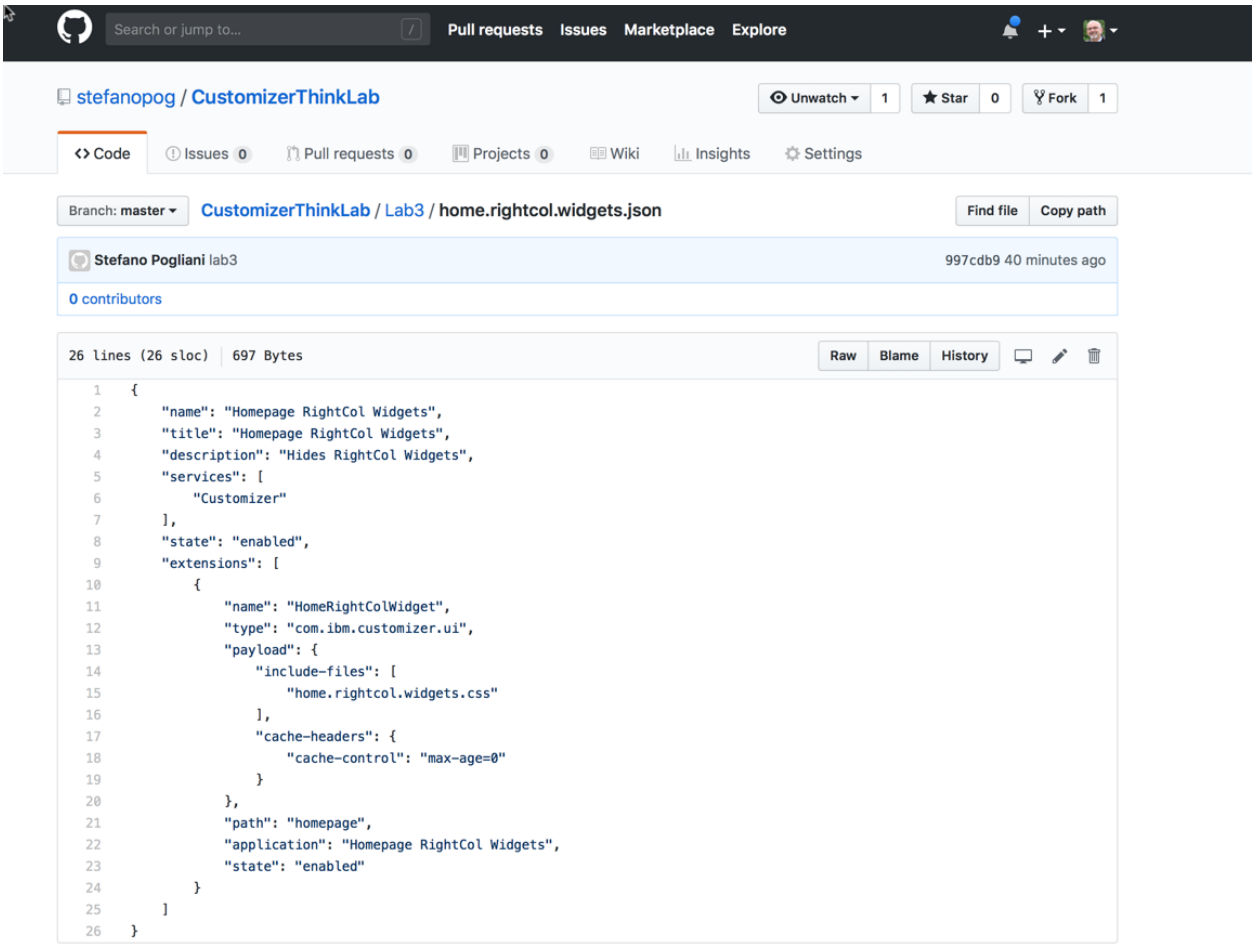
  - Line 13 and Line 14 respectively control this :

  - Line 11 and Line 12 respectively control this:

- Lines 18 and 19 hide the two items in the left column

The Extension ?

At this point, the JSON extension is very simple : it is available here:



Congratulations. This ends Lab 3.

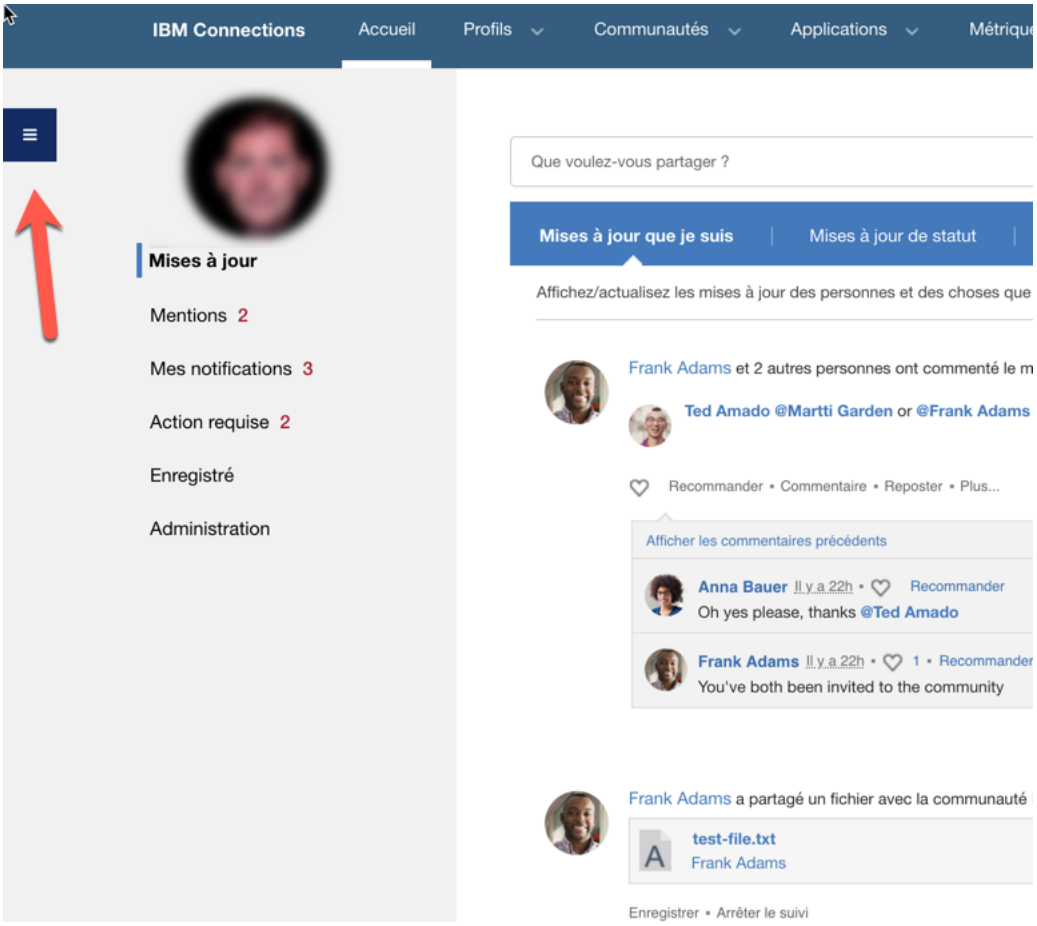
Lab 4 Using JQuery

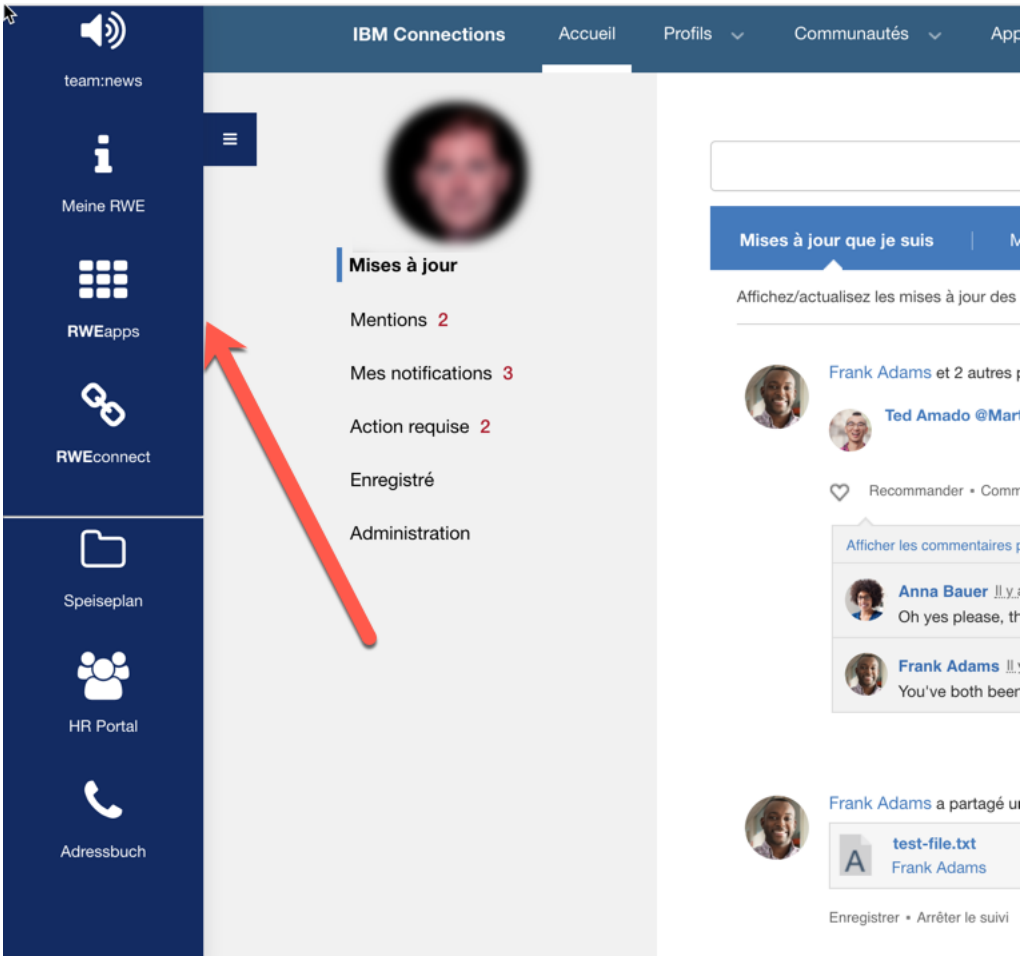
During the initial Lab1 and Lab2, we have started to see the use of the **Dojo Toolkit** to develop dynamic scripts.

Some developers are more familiar with the **JQuery toolkit**. This exercise guides you through the use of a simple **JQuery** script used within IBM Connections Customizer.

What we want to achieve

We want to add a simple flyout menu to all IBM Connections page. The Flyout menu has an animation as shown in the following pictures:



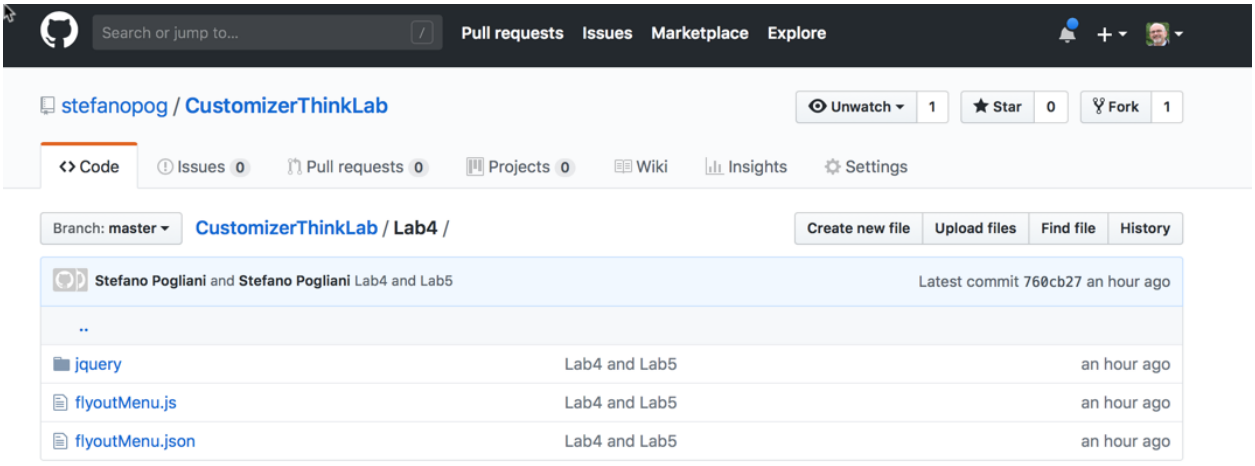


People familiar with the **JQuery** package are, probably, familiar with this kind of development.

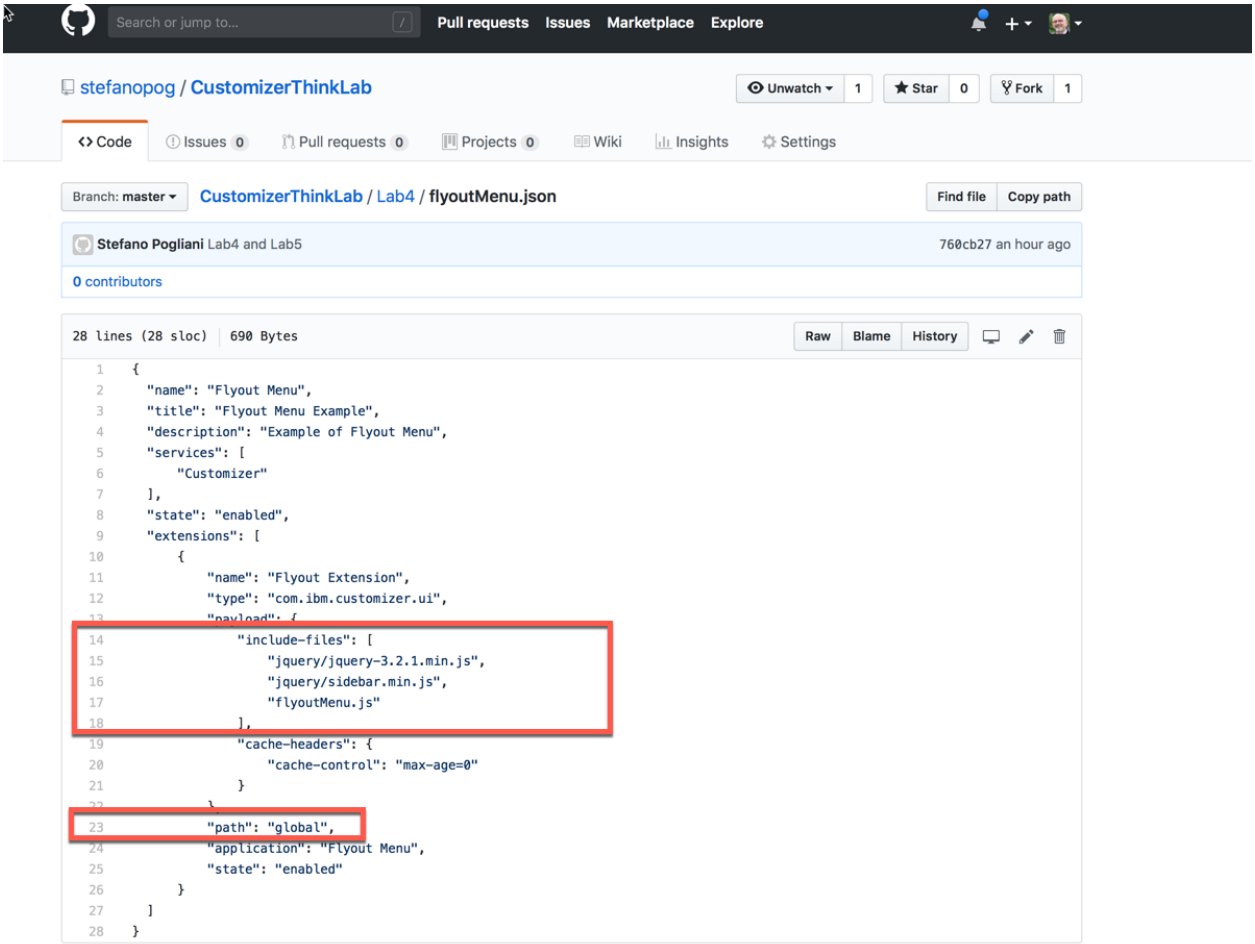
The intent of this lab is not to describe in detail the process of developing interfaces using JQuery, but to show how a JQuery-based development can be easily re-used within IBM Connections customizer.

How is it done ?

All the code is available in the Lab4 exercise.



Let's look at the JSON file first, flyoutMenu.json, this time:



The important sections are highlighted in the screenshot above and explained here:

- The "`include-files`" section.
  It references 3 files. Two of them come from the "`jquery subdirectory`" and the third one is the "`flyoutMenu.js`" that we will analyze later and that contains the code for our extension.
  The two scripts contain the runtime of the "**JQuery package**" and of the additional "**Semantic Sidebar**" package.
  We upload them in this way as it is very simple.

  We will discuss during the class how to eventually load them differently.

- The "`path: 'global'`" section.
  We want here to make the script available on **all Connections pages**.

Now, let's have a look at the flyoutMenu.js file, which actually contains the code that makes the magic for us.

- The first thing to notice is that we introduce the script with a function that is declared and executed on the fly.
  This function immediately inherits from JQuery (and we see the familiar "`$ notation`" used immediately). This means that JQuery is immediately active.

- The lines from 4 to 7 accomplish a basic function: they load the CSS scripts associated to the two packages we previously uploaded.
  This could be done statically using the same principle used for the javascript packages

- From that point on, this is mere JQuery programming.

Congratulations. You have finished exercise 4.

Lab 5 Using a Framework

In the previous exercises, we have seen the use of some controls related to the packages that can be used to create your IBM Connections Customizer scripts.

There are also situations in which an IBM Connections page is not rendered statically; in those cases, you may not be able to find the elements you require right away.

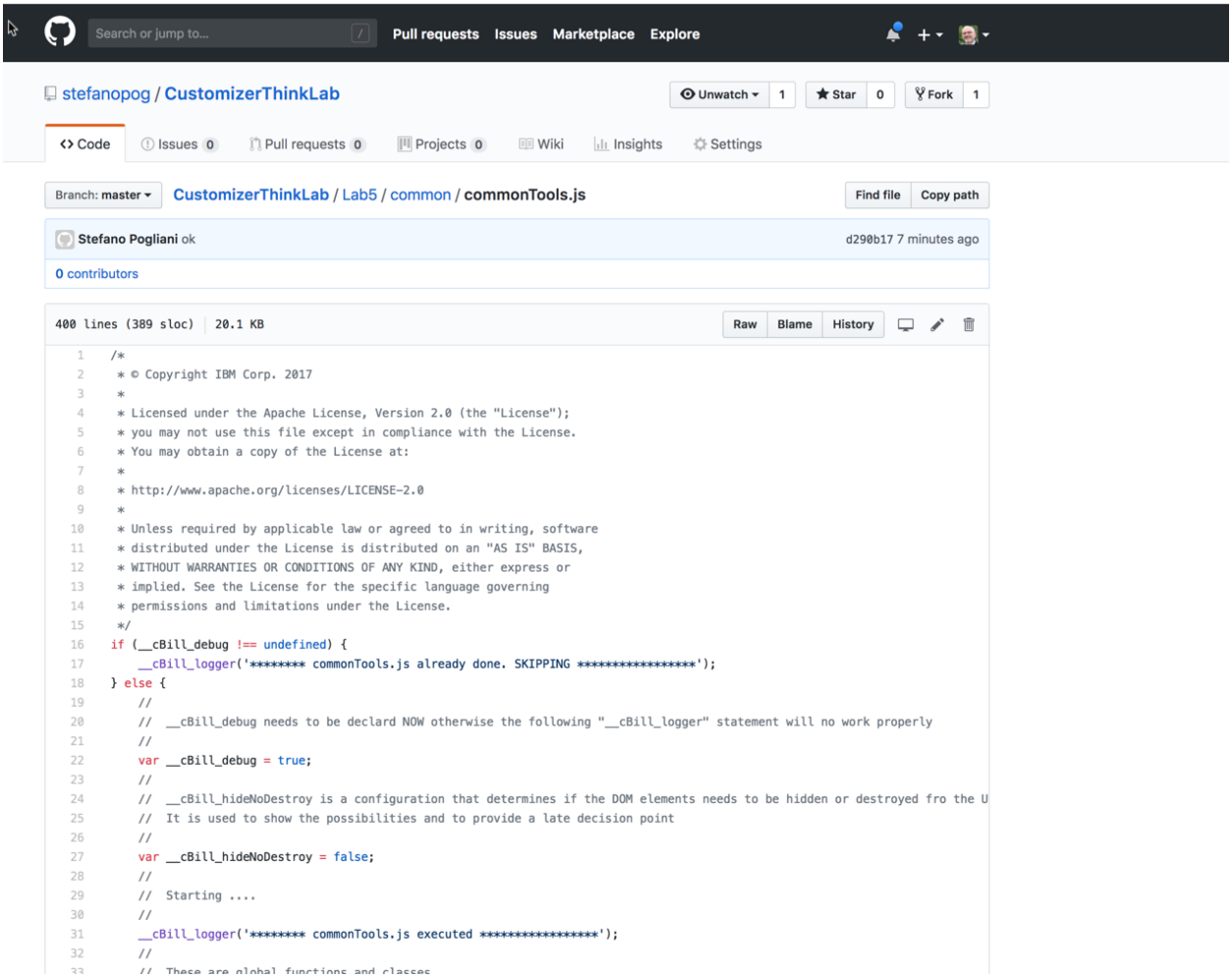
There are other situations in which you have to simulate a mouse action in order to, for instance, change the default value/status of a given element; in most situations, changing the visual representation is not enough as you may miss some “side-effects” that properly executing the “mouse action” would apply.

These are some of the reasons why you may benefit from a “framework”. By “framework” here we refer to a collection of tools that would allow you to simplify development and to keep it consistent.

What we want to achieve

This module will introduce you a minimalist set of tools that you can use in writing your IBM Customizer extensions.

Please refer to this link (Lab5) and to the commonTools.js file.



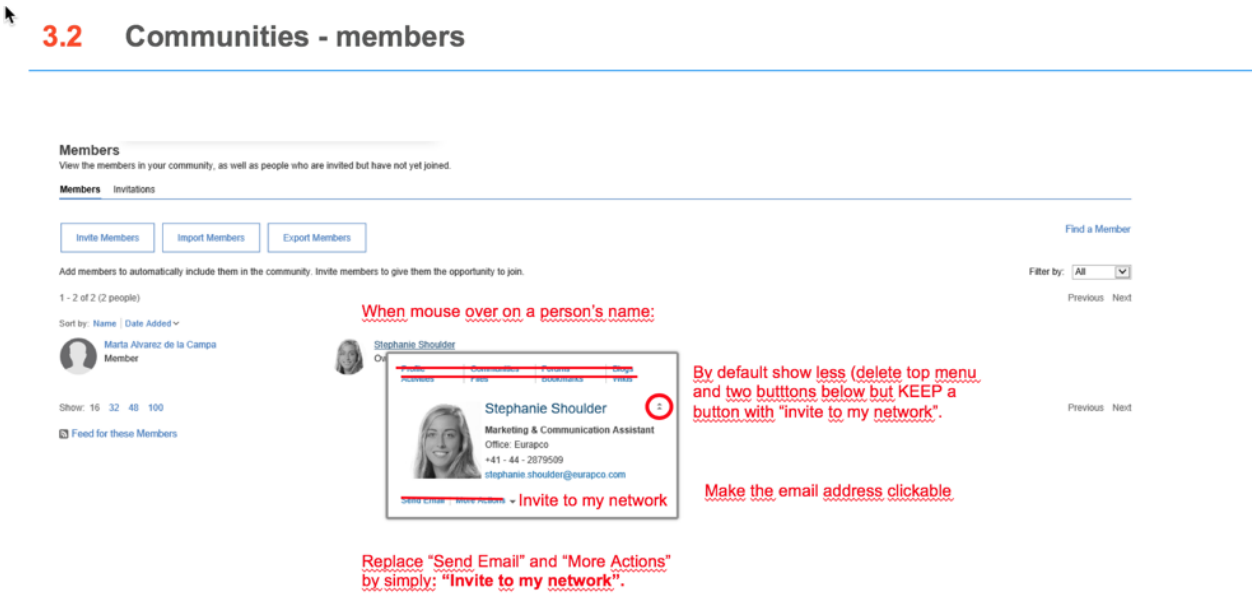
The instructor will guide you through the tools included in the framework.

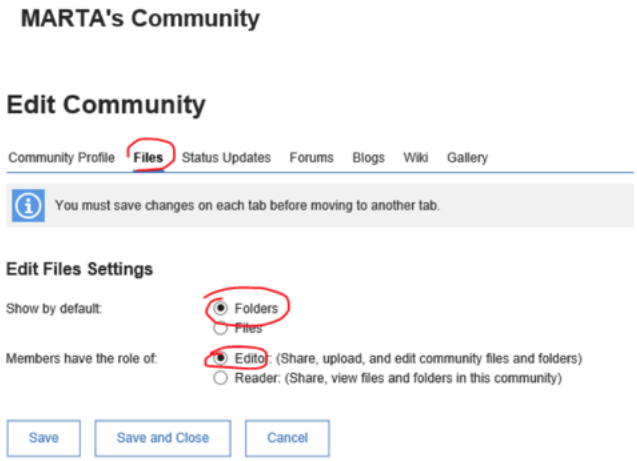
Lab 6 Changing the Behavior of IBM Connections

So far we have been doing very simple labs that, mainly, focused on changing the look and feel of some elements of the IBM Connections user interface.

In this lab, we will be dealing with something more complex: we want to change some default behavior of the interface and we want to provide a different user experience.

What we want to achieve





The challenges are interesting:

1. We need to change the behavior of the Business Card “in the context of a Connections application, in our case Communities”
   The challenge here are the following:
   a. to understand that Business Cards are dynamically generated: so, we need to apply our changes once a given Business Card is generated
   b. to understand that the default behavior of Connections says that once a user changes some behavior about a Business Card she is viewing, that behavior gets changed for all other Business Cards
   c. to understand that, sometimes, IBM Connections does not refresh the page (i.e. your Customizer script is not invoked) even if the page visually changes.

2. We need to change the default associated to the Business Card
   The challenge here is that the only way to ensure that an “opaque behavior” is honored is, actually, to follow the way an interactive user executes it.

3. We need to change the default applied to a given community.
   The challenge here is catch which panel the user is editing.

We Value Your Feedback!

- Don’t forget to submit your Think 2018 session and speaker feedback! Your feedback is very important to us – we use it to continually improve the conference.
- Access the Think 2018 agenda tool to quickly submit your surveys from your smartphone, laptop or conference kiosk.