

# NOTE DE STAGE 2021

## CHEZ COFLINE

### SOMMAIRE

Contexte .....	1
Glossaire:.....	1
Semaine du 21 juin au 25 juin 2021 .....	2
Missions : .....	2
Réalisations : .....	2

### CONTEXTE

Lundi, notre maître de stage a voulu renforcer notre travail en groupe qui n'était pas optimisé car nous faisons le même travail. C'était plus de l'entraide qu'un véritable travail d'équipe. Avec mon binôme on avait tout d'abord décidé de se répartir les tâches sur l'*API* en prenant pour base mon projet, je m'occupais de rédiger les tests et lui s'occupait de corriger les controllers suite à l'étude de différentes sources. Cependant suite à une nouvelle réunion dans la journée, Ugo nous a conseillé de faire autrement car les tâches se chevauchaient toujours, ainsi Julien a continué sur l'*API* seul et je me suis chargée de faire évoluer l'**Add-In Word**. J'ai décidé cette semaine de ne pas faire une mise en page par jour car les missions étaient reliées entre elles et se chevauchaient de jour en jour.

### GLOSSAIRE:

- ✿ **API:** Application Programming Interface
- ✿ **HTML:** HyperText Markup Language
- ✿ **VSCode:** Visual Studio Code
- ✿ **HTTP:** HyperText Transfer Protocol
- ✿ **HTTPS:** HyperText Transfer Protocol Secure
- ✿ **CDN:** Content Delivery Network

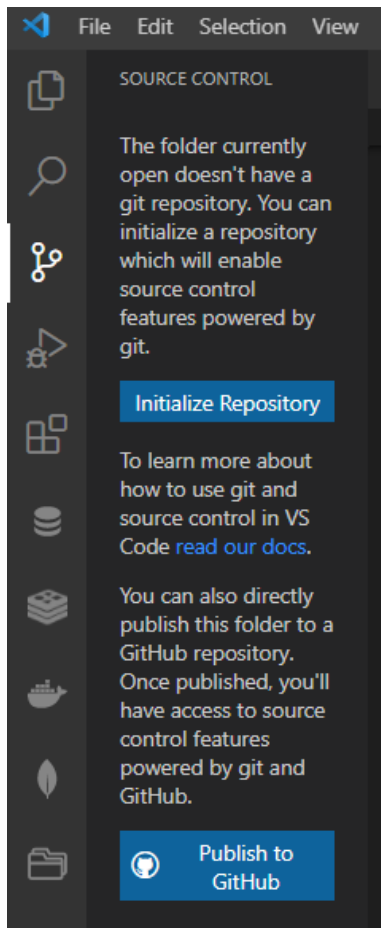
## SEMAINE DU 21 JUIN AU 25 JUIN 2021

## MISSIONS :

1. Ajout de **Git** au projet de l'*API* afin de pouvoir accentuer le travail de groupe. Passage du projet par **GitHub**.
2. Chercher comment relier l'**Add-In Word** à l'*API* en cours de développement. Mettre en pratique.
3. Chercher comment insérer les paragraphes et les différents titres avec des données provenant d'une *API* mais où la mise en forme intégrée de *Word* s'applique sans intervention manuelle.
4. Chercher s'il est possible d'insérer le texte mis en forme grâce à de l'**HTML**.
5. Chercher s'il est possible d'avoir plusieurs pages dans l'**Add-In Word**.

## REALISATIONS :

1. Mise en place de Git et problème rencontré.

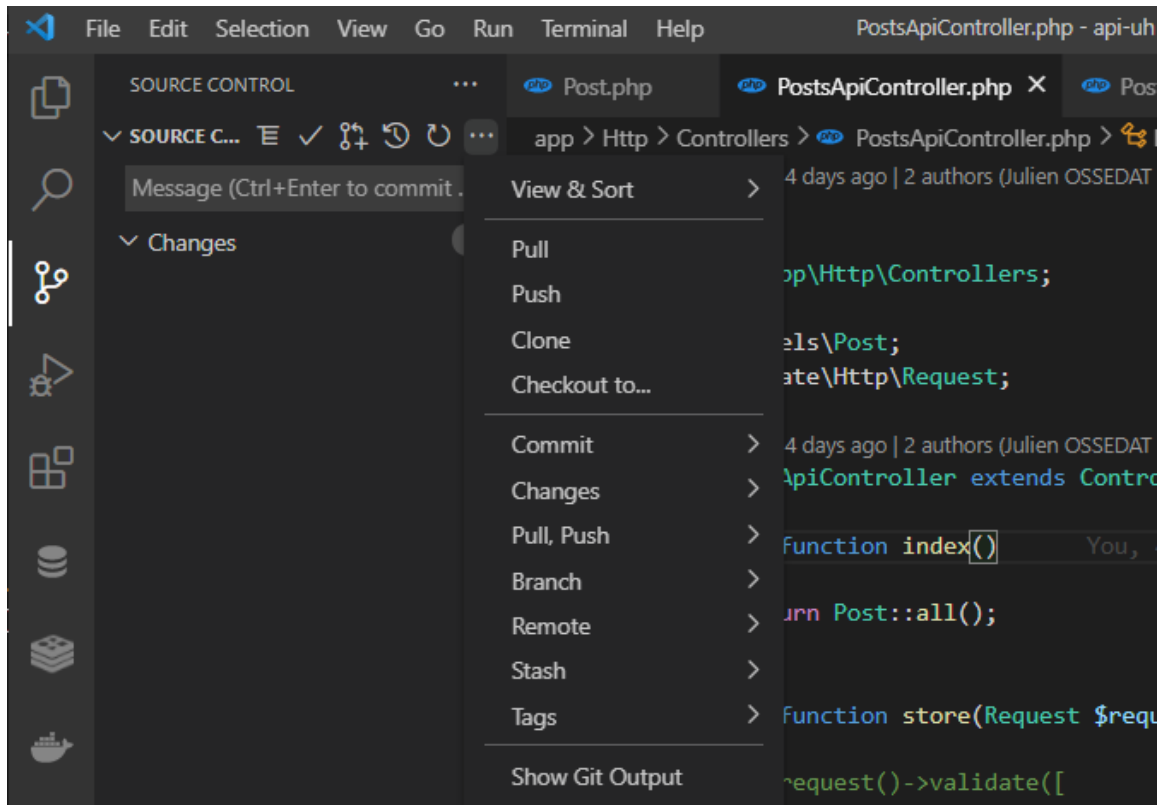


Les commandes liées à **Git** ne fonctionnaient pas dans le terminal intégré à **VSCode**, malgré le fait que je pouvais afficher la version, j'ai dû passer par la console. J'ai utilisé la commande `$ git init`, pour initialiser le repository **Git**. Préalablement, j'avais lié mon compte **GitHub** à **VSCode**. J'ai ensuite rencontré un autre problème, je ne pouvais pas faire de commit et publier sur **GitHub** tous mes fichiers. L'erreur qui s'affichait m'a amenée à effectuer les commandes suivantes :

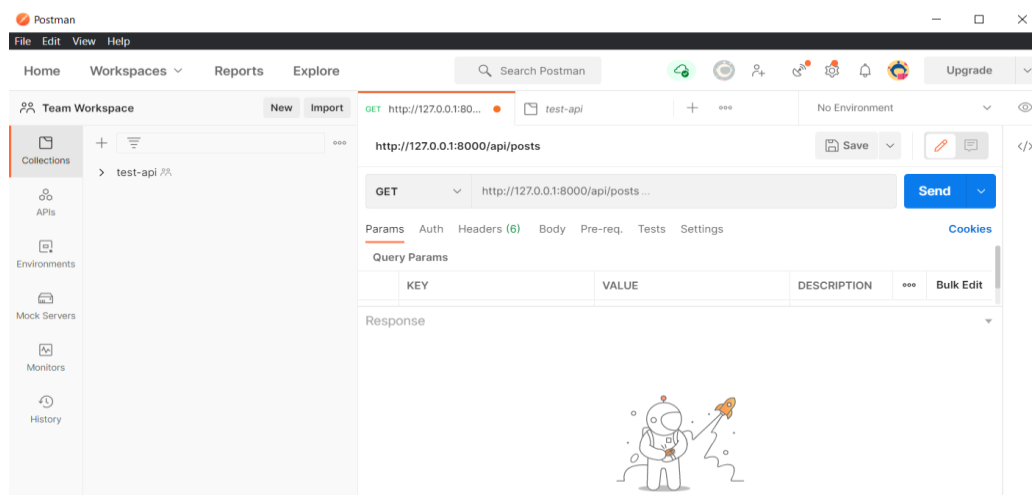
```
$ gpg - -gen-key
```

```
$ git config - -global user.signingkey 0A46826A
```

Une fois cela fait, j'ai pu avoir accès à toutes les commandes **Git** par l'interface **VSCode**.



En plus de cela comme expliqué dans le contexte j'avais commencé à travailler avec Julien sur le projet de l'api. J'étais en charge des tests sur l'api et étant donné que notre maître de stage nous avait parlé de **Postman**, j'ai téléchargé le logiciel et commencé à me renseigner dessus<sup>123</sup>.



<sup>1</sup> [https://www.youtube.com/watch?v=FjgYtQK\\_zLE](https://www.youtube.com/watch?v=FjgYtQK_zLE)

<sup>2</sup> <https://www.twilio.com/blog/building-and-consuming-a-restful-api-in-laravel-php>

<sup>3</sup> <https://www.youtube.com/watch?v=ET-UrsgDwGY>

## 2.

## a. Technologies et sources

Pour réaliser cet objectif je pensais d'abord utiliser la méthode `Fetch()`. Le projet étant en *JavaScript* pour la partie dynamique je pensais que cette méthode fonctionnerait bien, elle semblait « simple » et bien documentée. Mais après discussion avec mon maître de stage et quelques recherches en plus, la méthode `Fetch()` n'étant pas la plus performante, j'ai redirigé mes recherches vers la méthode utilisant le package **Axios**<sup>4</sup>. Les deux manières sont assez proches en termes d'utilisation et de compréhension. Les deux sont basées sur le principe de **Promise**<sup>5</sup> et peuvent être utilisées de façon asynchrone avec **Async/Await**<sup>6</sup>.

Sous conseil de mon maître de stage j'ai d'abord fait mes essais sur un projet à part. J'ai donc utilisé **Axios** et fini par réussir à m'en servir, par contre j'ai dû utiliser le **CDN** pour l'importer car « `import axios from "axios"` » ne semblait pas fonctionner. J'ai pu faire mes tests grâce à **Reqres.in**<sup>7</sup> une **Rest-API** libre d'accès. Mais au début j'avais beaucoup de difficulté car c'était nouveau et aussi parce que j'utilisais **Mockbin.com**<sup>8</sup> pour les tests, le lien étant en **HTTP** et non **HTTPS** cela ne fonctionnait pas car non supporté par **WordAPI**.

Voici les principales sources utilisées :

<https://openclassrooms.com/fr/courses>

<https://www.youtube.com/watch?v=tc8DU14qX6I>

<https://www.youtube.com/watch?v=cuEtnrL9-H0&t=274s>

[https://developer.mozilla.org/en-US/docs/Web/API/Fetch\\_API/Using\\_Fetch](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch)

<https://blog.logrocket.com/how-to-make-http-requests-like-a-pro-with-axios/>

<https://www.youtube.com/watch?v=6LyagkoRWYA>

<https://github.com/bradtraversy/axios-crash>

<https://codesandbox.io/s/zwlve?file=/src/index.js>

<https://zetcode.com/javascript/axios/>

<sup>4</sup> <https://github.com/axios/axios>

<sup>5</sup> <https://www.youtube.com/watch?v=DHvZLI7Db8E>

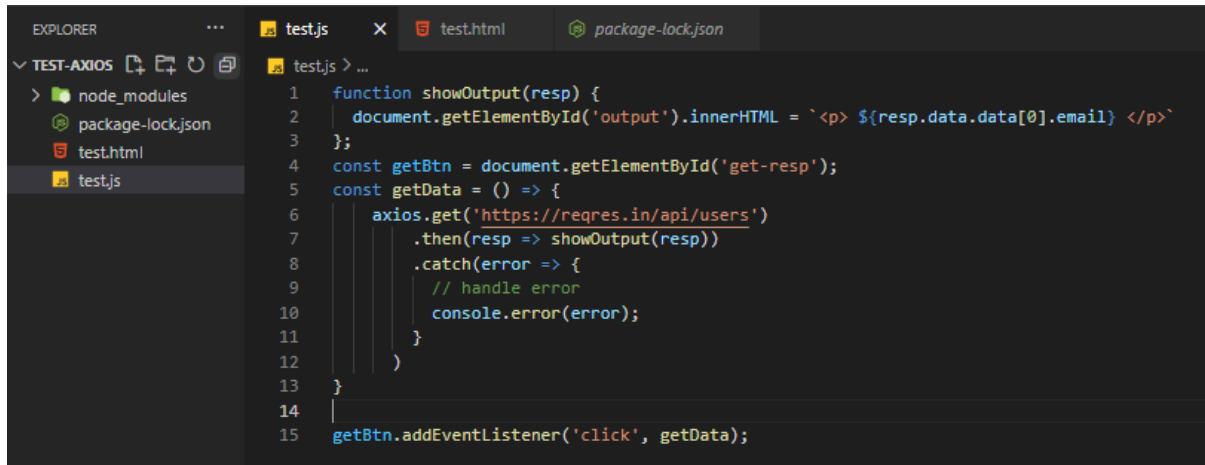
<sup>6</sup> [https://www.youtube.com/watch?v=V\\_Kr9OSfDeU](https://www.youtube.com/watch?v=V_Kr9OSfDeU)

<sup>7</sup> <https://reqres.in/>

<sup>8</sup> <https://mockbin.com/>

## b. Mise en pratique

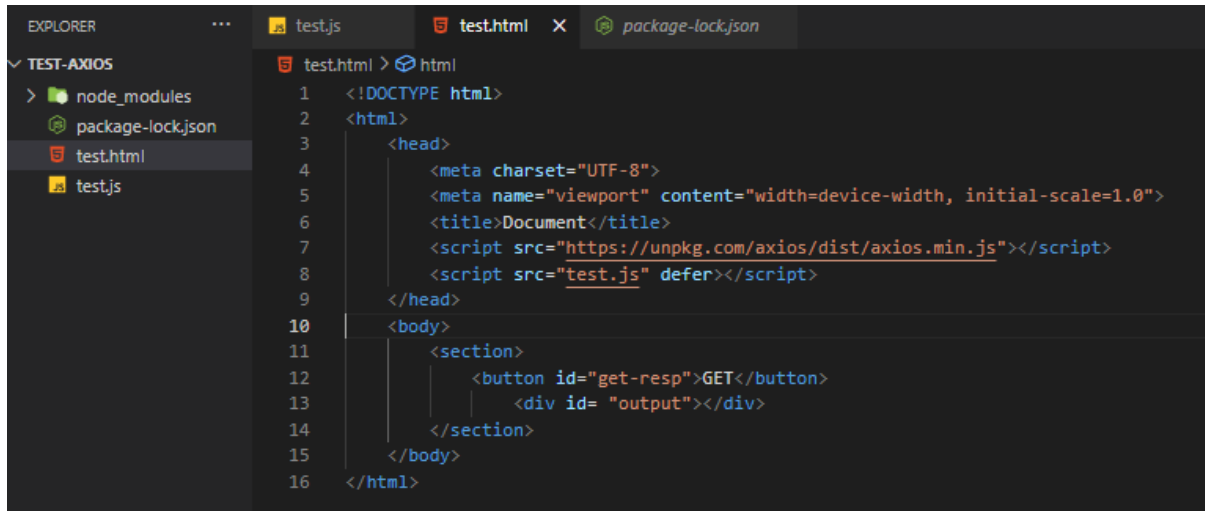
- Test sur un projet à part :



```

1  function showOutput(resp) {
2    document.getElementById('output').innerHTML = `<p> ${resp.data.data[0].email} </p>`
3  };
4  const getBtn = document.getElementById('get-resp');
5  const getData = () => {
6    axios.get('https://reqres.in/api/users')
7      .then(resp => showOutput(resp))
8      .catch(error => {
9        // handle error
10       console.error(error);
11     })
12  }
13
14  getBtn.addEventListener('click', getData);

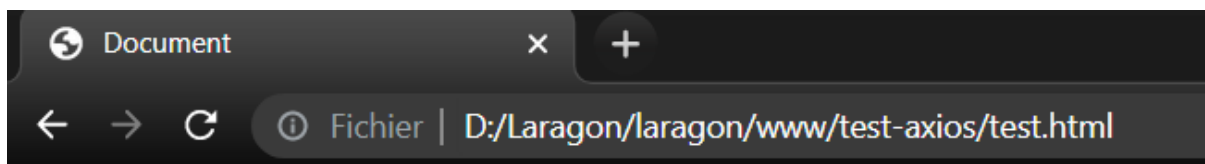
```



```

1  <!DOCTYPE html>
2  <html>
3    <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>Document</title>
7      <script src="https://unpkg.com/axios/dist/axios.min.js"></script>
8      <script src="test.js" defer></script>
9    </head>
10   <body>
11     <section>
12       <button id="get-resp">GET</button>
13       <div id= "output"></div>
14     </section>
15   </body>
16 </html>

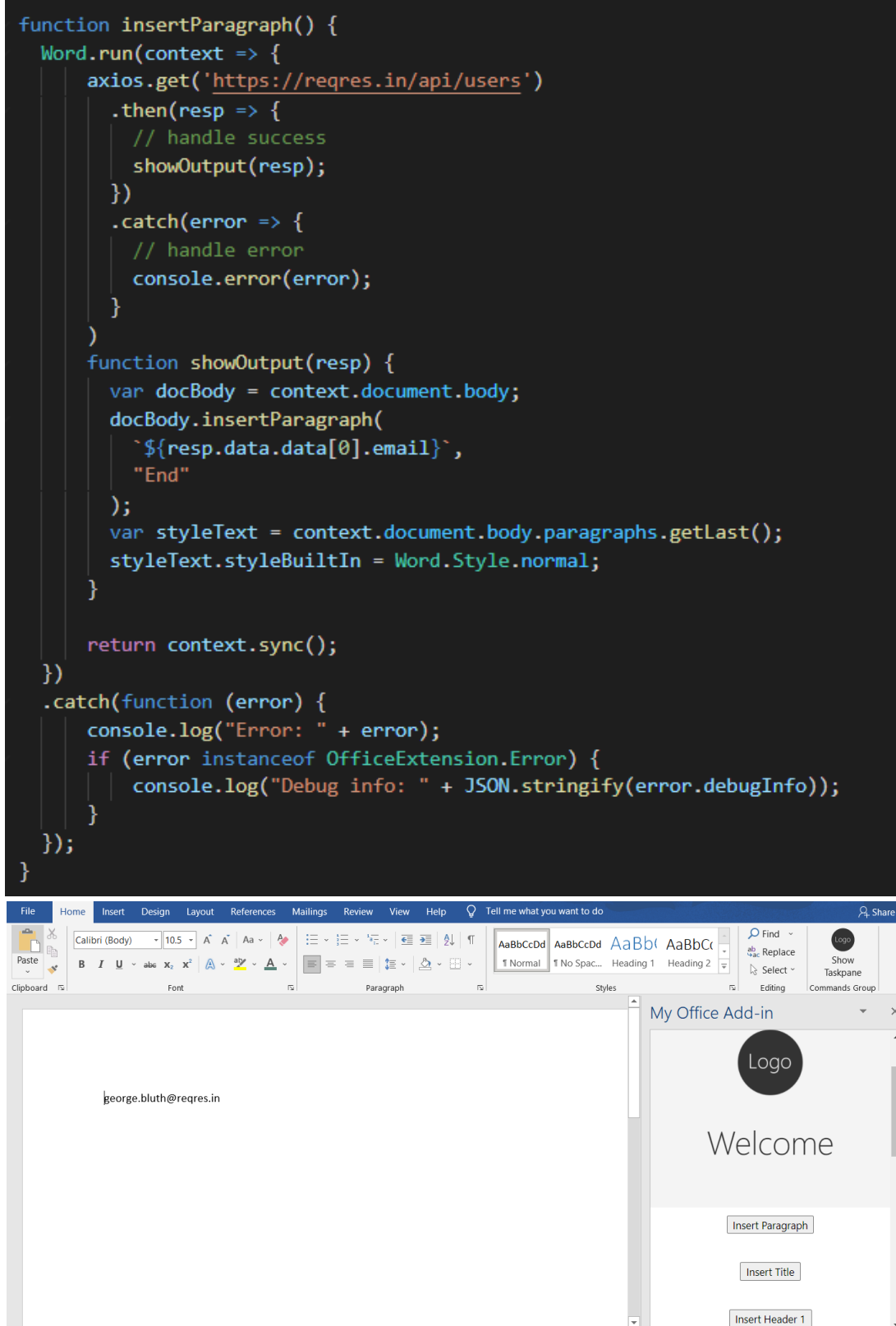
```



GET

george.bluth@reqres.in

- Test sur le projet de l'Add-In Word :

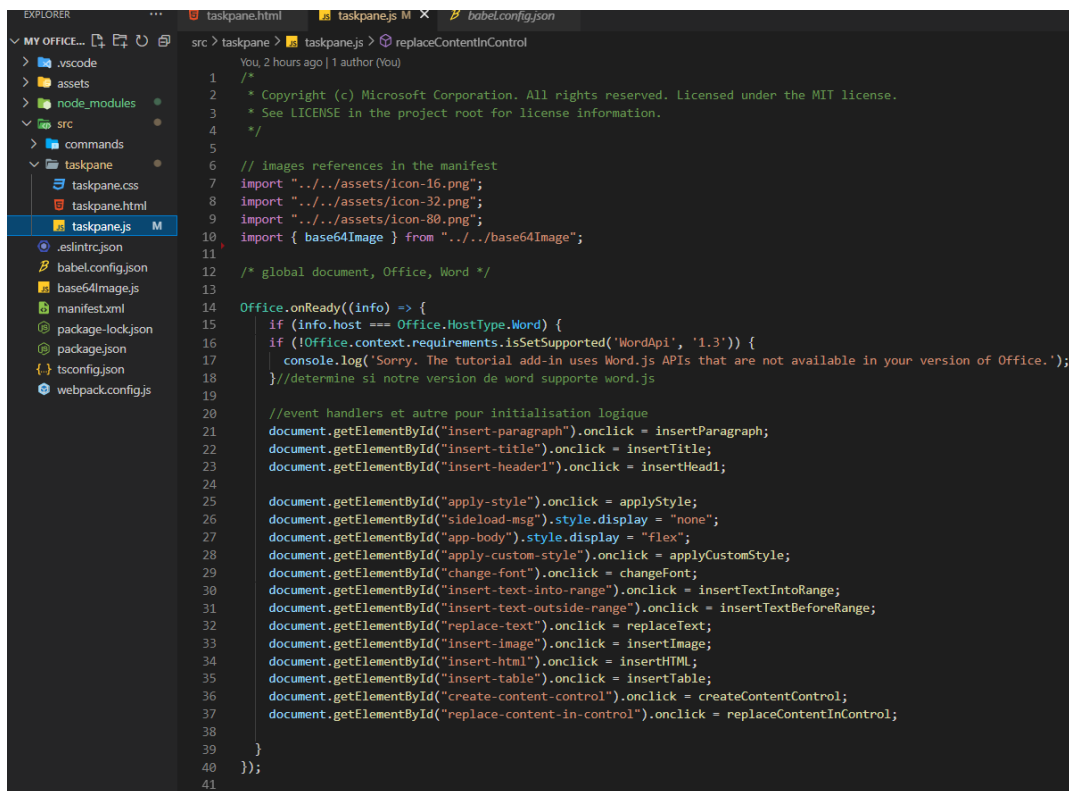


## 3.

## a. Processus

En m'inspirant du code déjà écrit en suivant le tutoriel de **WordAPI**<sup>9</sup>, celui écrit cette semaine et de la documentation **Microsoft**<sup>10</sup>, j'ai pu faire en sorte que le texte inséré se mette au format prédéfini par **Word**. Cependant ce n'est pas encore parfait car les fonctions de **WordAPI** sont assez restrictives. De plus j'ai dû passer par la méthode `insertParagraph()`<sup>11</sup> même pour les titres car il n'existe pas de méthode directe, on retrouve le titre seulement en tant que *property*<sup>12</sup>.

## b. Mise en pratique



```

1  You, 2 hours ago | 1 author (You)
2  /*
3   * Copyright (c) Microsoft Corporation. All rights reserved. Licensed under the MIT license.
4   * See LICENSE in the project root for license information.
5   */
6
7  // images references in the manifest
8  import "../assets/icon-16.png";
9  import "../assets/icon-32.png";
10 import "../assets/icon-80.png";
11 import { base64Image } from "../base64Image";
12
13 /* global document, Office, Word */
14
15 Office.onReady((info) => {
16   if (info.host === Office.HostType.Word) {
17     if (!Office.context.requirements.isSetSupported('WordApi', '1.3')) {
18       console.log('Sorry. The tutorial add-in uses Word.js APIs that are not available in your version of Office.');

```

<sup>9</sup> <https://docs.microsoft.com/en-us/office/dev/add-ins/tutorials/word-tutorial>

<sup>10</sup> [https://docs.microsoft.com/en-us/javascript/api/word/word.list?view=word-js-preview#insertParagraph\\_paragraphText\\_insertLocation](https://docs.microsoft.com/en-us/javascript/api/word/word.list?view=word-js-preview#insertParagraph_paragraphText_insertLocation)

<sup>11</sup> [https://docs.microsoft.com/en-us/javascript/api/word/word.list?view=word-js-preview#insertParagraph\\_paragraphText\\_insertLocation](https://docs.microsoft.com/en-us/javascript/api/word/word.list?view=word-js-preview#insertParagraph_paragraphText_insertLocation)

<sup>12</sup> <https://docs.microsoft.com/en-us/javascript/api/word/word.documentproperties?view=word-js-preview#title>

The image displays two screenshots of a Visual Studio Code editor window, showing the development of a Word add-in. The Explorer pane on the left shows the project structure, including files like `taskpane.html`, `taskpane.js`, `taskpane.css`, and various configuration files. The main editor area shows the `taskpane.js` file.

**Top Screenshot:** The code defines a function `insertParagraph()` that uses `Word.run(context => {` to execute a function within the Word context. Inside, it uses `axios.get('https://reqres.in/api/users')` to fetch data. The `.then` callback handles success by calling `showOutput(resp)`. The `.catch` callback handles errors by logging them. The `showOutput` function inserts a paragraph with the email of the first user and sets its style to `Word.Style.normal`.

```

41
42
43 function insertParagraph() {
44     Word.run(context => {
45         axios.get('https://reqres.in/api/users')
46             .then(resp => {
47                 // handle success
48                 showOutput(resp);
49             })
50             .catch(error => {
51                 // handle error
52                 console.error(error);
53             })
54     })
55     function showOutput(resp) {
56         var docBody = context.document.body;
57         docBody.insertParagraph(
58             `${resp.data.data[0].email}`,
59             "End"
60         );
61         var styleText = context.document.body.paragraphs.getLast();
62         styleText.styleBuiltIn = Word.Style.normal;
63     }
64
65     return context.sync();
66 })
67 .catch(function (error) {
68     console.log("Error: " + error);
69     if (error instanceof OfficeExtension.Error) {
70         console.log("Debug info: " + JSON.stringify(error.debugInfo));
71     }
72 });
73 }
74

```

**Bottom Screenshot:** The code defines a function `insertTitle()` that uses `Word.run(context => {` to execute a function within the Word context. Inside, it uses `axios.get('https://reqres.in/api/users')` to fetch data. The `.then` callback handles success by calling `setTitle(resp)`. The `.catch` callback handles errors by logging them. The `setTitle` function inserts a paragraph with the email of the first user and sets its style to `Word.Style.title`.

```

74
75
76 function insertTitle() {
77     Word.run(context => {
78         axios.get('https://reqres.in/api/users')
79             .then(resp => {
80                 // handle success
81                 setTitle(resp);
82             })
83             .catch(error => {
84                 // handle error
85                 console.error(error);
86             })
87     })
88
89     function setTitle(resp) {
90         var docBody = context.document.body;
91         docBody.insertParagraph(
92             `${resp.data.data[0].email}`,
93             "Start"
94         );
95         var firstParagraph = context.document.body.paragraphs.getFirst();
96         firstParagraph.styleBuiltIn = Word.Style.title;
97     }
98
99     return context.sync();
100 })
101 .catch(function (error) {
102     console.log("Error: " + error);
103     if (error instanceof OfficeExtension.Error) {
104         console.log("Debug info: " + JSON.stringify(error.debugInfo));
105     }
106 });
107 }
108
109
110
111

```



```

111
112 function insertHead1() {
113     Word.run(context => {
114         axios.get('https://regres.in/api/users')
115             .then(resp => {
116                 // handle success
117                 setHead1(resp);
118             })
119             .catch(error => {
120                 // handle error
121                 console.error(error);
122             })
123     })
124
125     function setHead1(resp) {
126         var docBody = context.document.body;
127         docBody.insertParagraph(
128             `${resp.data.data[0].email}`,
129             "End"
130         );
131         var styleText = context.document.body.paragraphs.getLast();
132         styleText.styleBuiltIn = Word.Style.heading1;
133     }
134
135     return context.sync();
136 }
137
138 .catch(function (error)
139 {
140     console.log("Error: " + error);
141     if (error instanceof OfficeExtension.Error)
142     {
143         console.log("Debug info: " + JSON.stringify(error.debugInfo));
144     }
145 });
146 }

```

```

4 <DOCTYPE html>
5 <html>
6
7 <head>
8     <meta charset="UTF-8" />
9     <meta http-equiv="X-UA-Compatible" content="IE=Edge" />
10    <meta name="viewport" content="width=device-width, initial-scale=1">
11    <title>Contoso Task Pane Add-in</title>
12
13    <!-- Office JavaScript API -->
14    <script type="text/javascript" src="https://appsforoffice.microsoft.com/lib/1.1/hosted/office.js"></script>
15    <script src="https://unpkg.com/axios/dist/axios.min.js"></script>
16
17    <!-- For more information on Fluent UI, visit https://developer.microsoft.com/fluentui/#. -->
18    <link rel="stylesheet" href="https://static2.sharepointonline.com/files/fabric/office-ui-fabric-core/9.6.1/css/fabric.min.css"/>
19
20    <!-- Template styles -->
21    <link href="taskpane.css" rel="stylesheet" type="text/css" />
22 </head>
23
24 <body class="ms-font-m ms-welcome ms-Fabric">
25     <header class="ms-welcome__header ms-bgColor-neutrallighter">
26         
27         <h1 class="ms-font-su">Welcome</h1>
28     </header>
29     <section id="sideload-msg" class="ms-welcome__main">
30         <h2 class="ms-font-xl">Please sideload your add-in to see app body.</h2>
31     </section>
32     <main id="app-body" class="ms-welcome__main" style="display: none;">
33         <button class="ms-Button" id="insert-paragraph">Insert Paragraph</button><br/><br/>
34         <button class="ms-Button" id="insert-title">Insert Title</button><br/><br/>
35         <button class="ms-Button" id="insert-header1">Insert Header 1</button><br/><br/>
36         <button class="ms-Button" id="apply-style">Apply Style</button><br/><br/>
37         <button class="ms-Button" id="apply-custom-style">Apply Custom Style</button><br/><br/>
38     </main>
39

```

4&5.

Je travaille sur ces missions en ce moment. Je dois continuer mes recherches et mes tests mais la création de plusieurs pages semble faisable. Par contre pour l'injection d'éléments **HTML** cela semble plus compliqué.

**En Cours ...**