

# NOTE DE STAGE 2021

## CHEZ COFLINE

### SOMMAIRE

Glossaire.....	1
Lundi 14 juin 2021 .....	2
Missions : .....	2
Réalisations : .....	2
Mardi 15 & Mercredi 16 juin 2021 .....	10
Missions : .....	10
Réalisations : .....	10
Jeudi 17 juin 2021 .....	16
Missions : .....	16
Réalisations : .....	16
Vendredi 18 juin 2021 .....	19
Missions : .....	19
Réalisations : .....	19

### GLOSSAIRE

- \* **DOM:** Document Object Model
- \* **VSC:** Visual Studio Code
- \* **NPM:** Node Package Manager
- \* **JSON:** JavaScript Object Notation
- \* **API:** Application Programming Interface
- \* **HTTP:** HyperText Transfer Protocol
- \* **XML:** Extensible Markup Language

LUNDI 14 JUIN 2021

## MISSIONS :

Pause sur les activités *ReactJS* en rapport avec le projet. L'initiation a *ReactJS* étant assez avancé, deux groupes ont été formés, un groupe pour produire un clone de Google Docs et un pour produire un clone de Google Drive.

1. Faire des recherches pour cloner Google Docs. Faire de la **conception** : **Quelle technologie ? Pourquoi ? Comment ?**

## REALISATIONS :

1.
  - a. Recherches

Notre maître de stage nous avait vaguement évoqué **Socket.IO** et **MongoDB**, je suis donc partie dans cette direction pour mes recherches. J'ai fini par trouver un tutoriel assez complet sur la chaîne YouTube **Web Dev Simplified**<sup>1</sup>.

## Résumé des recherches sur les technologies employées :

**Quill**<sup>2</sup> s'initialise avec un élément du **DOM** (un **objet Quill**) qui contiendra alors l'éditeur de texte. Il est surtout reconnu pour sa flexibilité et son extensibilité (personnalisation). On peut l'installer sur **VSC** grâce à **NPM** (dépendance) et l'ajouter à notre projet. Ce qui nous intéresse particulièrement avec cette dépendance ce sont les **Deltas**, compatible **JSON**, ils représentent les changements apportés à un document et permettent d'organiser/diviser les documents Quill (le texte, la mise en forme...) sans ambiguïté et sans complexifier la structure HTML. On enregistrera les Deltas (les changements donc) au lieu d'enregistrer tout le document à chaque fois. Quill est open source et gratuit, les alternatives telles que **Froala Editor** et **TinyMCE** sont payants. (Les Deltas sont une librairie à part entière de Quill donc possiblement utilisable avec d'autre éditeur de texte mais plus compliqué ).

**Socket.IO**<sup>3</sup> est une librairie qui permet une **communication bidirectionnelle en temps réel** et basé sur des **événements** entre le serveur et l'application. On obtient aussi un **objet socket** qui étend la classe **EventEmitter** de Node.js.

**UUID**<sup>4</sup> nous permet de générer automatiquement un identifiant unique **RFC**, qui nous permettra d'identifier chaque document et de les isoler (**Socket.IO**) afin que les données d'un utilisateur ne puisse pas être envoyées à un autre.

**MongoDB**<sup>5</sup> est une base de données « document » non relationnelle (**NoSQL**), elle conserve les données dans des documents similaire **JSON**. On peut donc utiliser du langage **JQuery** qui nous permettra de faire énormément d'opération de manière plus naturelle et plus simple qu'avec le SQL.

<sup>1</sup> <https://www.youtube.com/watch?v=iRaelG7v0OU>

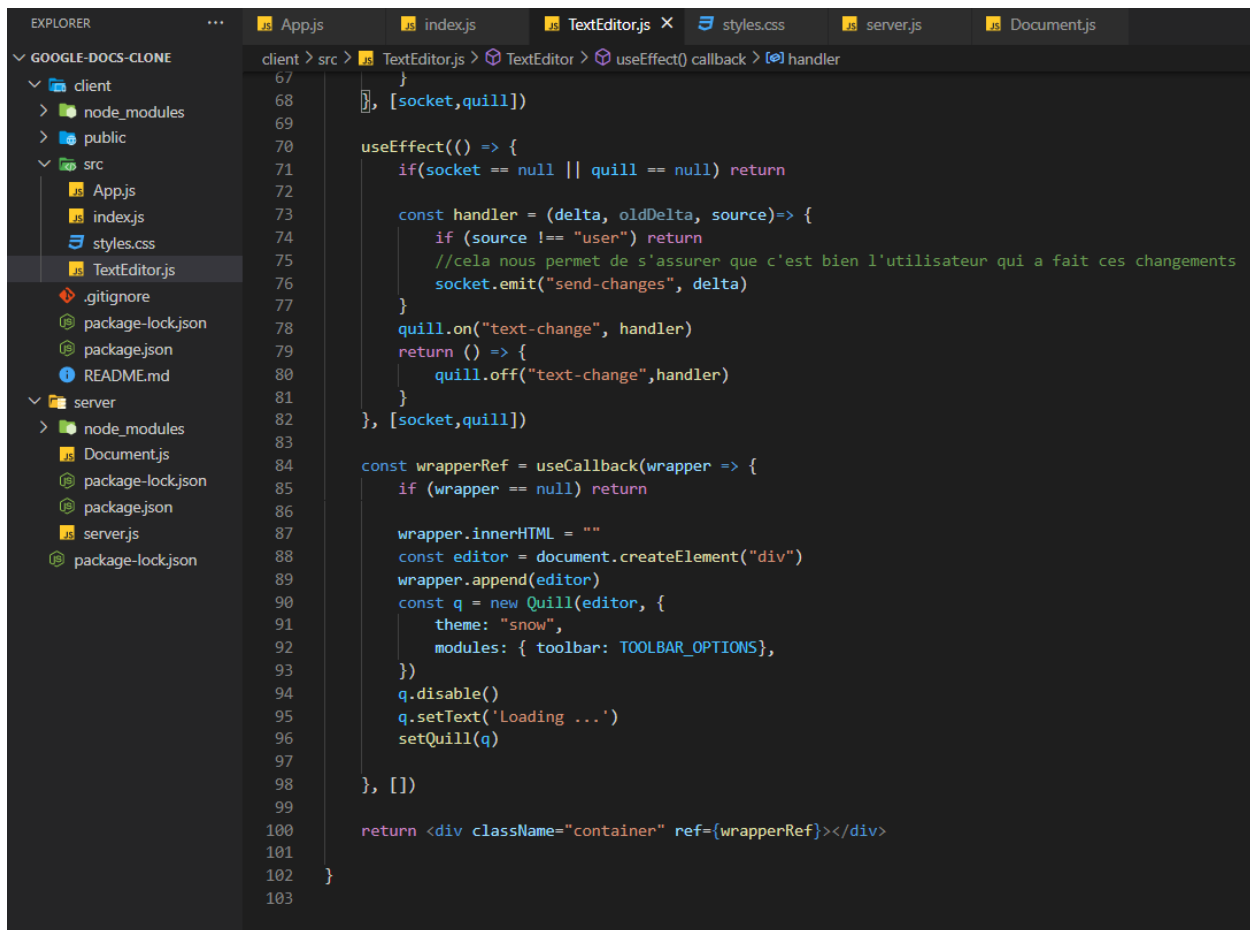
<sup>2</sup> <https://quilljs.com/docs/quickstart/>

<sup>3</sup> <https://socket.io/docs/v4>

<sup>4</sup> <https://www.npmjs.com/package/uuid>

<sup>5</sup> <https://www.mongodb.com/>

## b. Application (avec tutoriel)

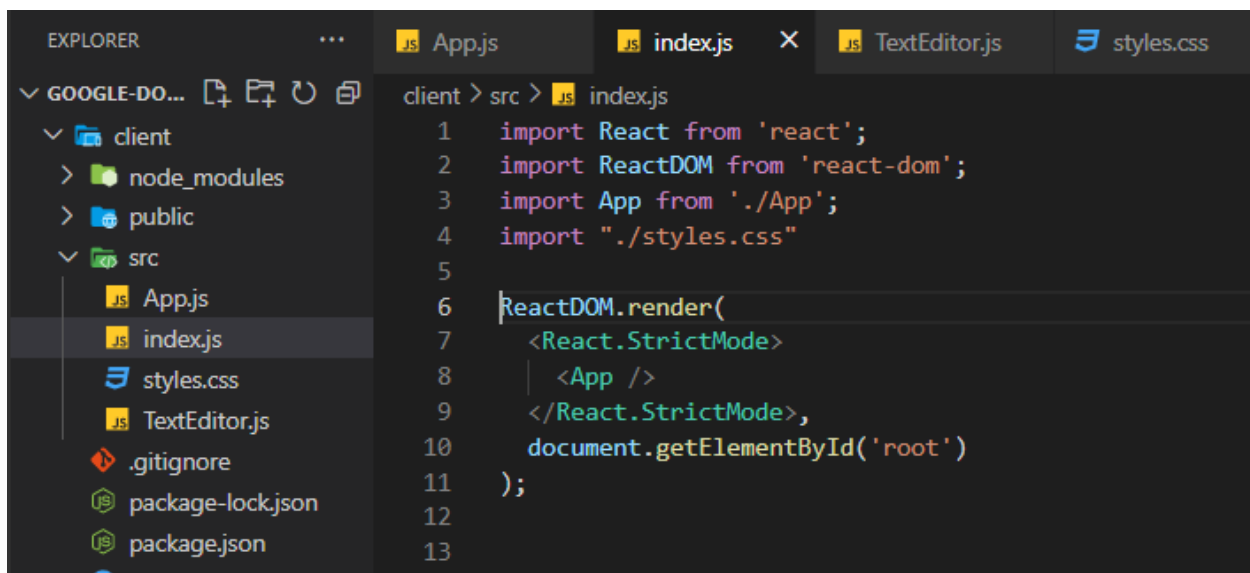


The screenshot shows the VS Code editor with the 'TextEditor.js' file open. The file is located in the 'client > src' directory. The code implements a text editor using Quill and Socket.io. It uses `useEffect` to initialize the editor and `useCallback` to create a wrapper for the editor. The code includes comments in French explaining the purpose of certain lines.

```

67  }
68  }, [socket, quill])
69
70  useEffect(() => {
71    if(socket == null || quill == null) return
72
73    const handler = (delta, oldDelta, source)=> {
74      if (source !== "user") return
75      //cela nous permet de s'assurer que c'est bien l'utilisateur qui a fait ces changements
76      socket.emit("send-changes", delta)
77    }
78    quill.on("text-change", handler)
79    return () => {
80      quill.off("text-change", handler)
81    }
82  }, [socket, quill])
83
84  const wrapperRef = useCallback(wrapper => {
85    if (wrapper == null) return
86
87    wrapper.innerHTML = ""
88    const editor = document.createElement("div")
89    wrapper.append(editor)
90    const q = new Quill(editor, {
91      theme: "snow",
92      modules: { toolbar: TOOLBAR_OPTIONS},
93    })
94    q.disable()
95    q.setText('Loading ...')
96    setQuill(q)
97
98  }, [])
99
100  return <div className="container" ref={wrapperRef}></div>
101
102  }
103

```



The screenshot shows the VS Code editor with the 'index.js' file open. The file is located in the 'client > src' directory. The code imports React, ReactDOM, and the App component, and renders the App component into the root element of the document.

```

1  import React from 'react';
2  import ReactDOM from 'react-dom';
3  import App from './App';
4  import "./styles.css"
5
6  ReactDOM.render(
7    <React.StrictMode>
8      <App />
9    </React.StrictMode>,
10    document.getElementById('root')
11  );
12
13

```

```

client > src > .js TextEditor.js > TextEditor
1  import { useCallback, useEffect, useState } from 'react'
2  import Quill from "quill"
3  import "quill/dist/quill.snow.css"
4  import {io} from "socket.io-client"
5  import { useParams } from "react-router-dom"
6
7  const SAVE_INTERVAL_MS = 2000
8  const TOOLBAR_OPTIONS = [
9    [{header: [1,2,3,4,5,6, false]}],
10   [{font: []}],
11   [{list: "ordered"},{list: "bullet"}],
12   ["bold","italic","underline"],
13   [{color:[]},{background: []}],
14   [{script:"sub"},{script: "super"}],
15   [{align: []}],
16   ["image","blockquote","code-block"],
17   ["clean"],
18 ]
19
20 export default function TextEditor() {
21   const { id: documentId } = useParams()
22   const [socket, setSocket] = useState()
23   const [quill, setQuill] = useState()
24
25   useEffect(() => {
26     const s = io("http://localhost:3001")
27     setSocket(s)
28
29     return () => {
30       s.disconnect()
31     }
32   }, [])
33
34   useEffect(() => {
35     if (socket == null || quill == null) return
36
37     socket.once("load-document", document => {
38       quill.setContents(document)
39       quill.enable()
40     })
41
42     socket.emit("get-document", documentId)
43
44   }, [socket,quill,documentId])
45

```

The image displays two screenshots of a VS Code editor interface, showing the development of a Google Docs clone. The Explorer panel on the left shows the project structure, including client and server folders. The top screenshot shows the TextEditor.js file with the following code:

```

45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83

```

The bottom screenshot shows the same file with the following code:

```

83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103

```

```

1  *, *::before, *::after {
2      box-sizing: border-box;
3  }
4
5  body {
6      background-color: #F3F3F3;
7      margin: 0;
8  }
9
10 .container .ql-editor {
11     width: 8.5in;
12     min-height: 11in;
13     padding: 1in;
14     margin: 1rem;
15     box-shadow: 0 0 5px 0 rgba(0, 0, 0, .5);
16     background-color: white;
17 }
18
19 .container .ql-container.ql-snow {
20     border: none;
21     display: flex;
22     justify-content: center;
23 }
24
25 .container .ql-toolbar.ql-snow {
26     display: flex;
27     justify-content: center;
28     position: sticky;
29     top: 0;
30     z-index: 1;
31     background-color: #F3F3F3;
32     box-shadow: 0 0 5px 0 rgba(0, 0, 0, .5);
33 }
34
35 @media print {
36     body {
37         background-color: none;
38     }
39
40     .container .ql-editor {
41         width: 6.5in;
42         height: 9in;
43         padding: 0;
44         margin: 0;
45         box-shadow: none;
46         align-self: flex-start;
47     }
48
49     .container .ql-toolbar.ql-snow {
50         display: none;
51     }

```

The screenshot shows the VS Code editor with the 'server.js' file open. The Explorer sidebar on the left shows the project structure: 'GOOGLE-DOCS-CLONE' with folders 'client', 'node\_modules', 'public', and 'src'. The 'src' folder contains 'App.js', 'index.js', 'styles.css', and 'TextEditor.js'. The 'server' folder contains 'node\_modules', 'Document.js', 'package-lock.json', 'package.json', and 'server.js'. The 'server.js' file is selected and its content is displayed in the editor. The code defines a MongoDB connection, sets up a Socket.io server on port 3001 with CORS, and implements a simple document management system with 'get-document', 'send-changes', and 'save-document' events. A 'findOrCreateDocument' function is also defined.

```

server > server.js > io > cors
1  const mongoose = require("mongoose")
2  const Document = require("./Document")
3
4  mongoose.connect("mongodb://localhost/google-docs-clone", {
5    useNewUrlParser: true,
6    useUnifiedTopology: true,
7    useFindAndModify: false,
8    useCreateIndex: true,
9  })
10
11  const io = require("socket.io")(3001, {
12    cors: {
13      origin: "http://localhost:3000",
14      methods: ["GET", "POST"],
15    },
16  })
17
18  const defaultValue = ""
19
20  io.on("connection", socket => {
21    socket.on("get-document", async documentId => {
22      const document = await findOrCreateDocument(documentId)
23      socket.join(documentId)
24      socket.emit("load-document", document.data)
25    })
26
27    socket.on("send-changes", delta => {
28      socket.broadcast.to(documentId).emit("receive-changes", delta)
29    })
30
31    socket.on("save-document", async data => {
32      await Document.findByIdAndUpdate(documentId, { data })
33    })
34  })
35
36  async function findOrCreateDocument(id) {
37    if (id == null) return
38
39    const document = await Document.findById(id)
40    if (document) return document
41    return await Document.create({ _id: id, data: defaultValue })
42  }
43
44

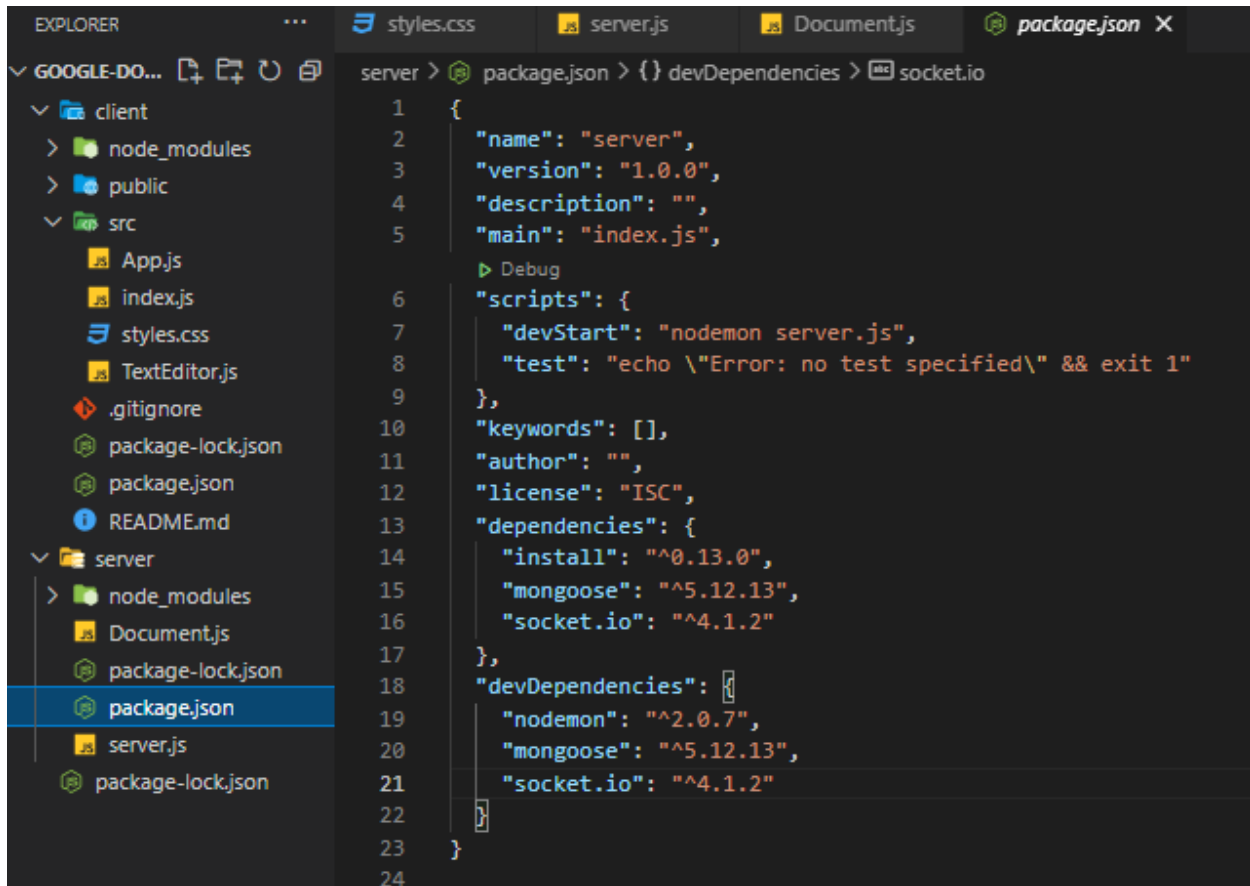
```

The screenshot shows the VS Code editor with the 'Document.js' file open. The Explorer sidebar on the left shows the project structure, with 'node\_modules' selected. The 'Document.js' file is selected and its content is displayed in the editor. The code defines a Mongoose schema for a 'Document' with fields '\_id' (String) and 'data' (Object), and registers it with the Mongoose model.

```

server > Document.js > <unknown>
1  const { Schema, model } = require("mongoose")
2
3  const Document = new Schema({
4    _id: String,
5    data: Object,
6  })
7
8  module.exports = model("Document", Document)

```

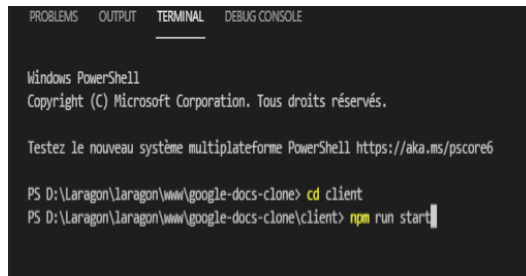


The screenshot shows the VS Code interface. On the left, the Explorer pane displays the project structure for 'GOOGLE-DO...'. The 'server' folder is selected, showing files like 'node\_modules', 'Document.js', 'package-lock.json', 'package.json', and 'server.js'. The main editor area shows the 'package.json' file for the 'server' project. The file contains the following JSON structure:

```

1 {
2   "name": "server",
3   "version": "1.0.0",
4   "description": "",
5   "main": "index.js",
6   "scripts": {
7     "devStart": "nodemon server.js",
8     "test": "echo \"Error: no test specified\" && exit 1"
9   },
10  "keywords": [],
11  "author": "",
12  "license": "ISC",
13  "dependencies": {
14    "install": "^0.13.0",
15    "mongoose": "^5.12.13",
16    "socket.io": "^4.1.2"
17  },
18  "devDependencies": {
19    "nodemon": "^2.0.7",
20    "mongoose": "^5.12.13",
21    "socket.io": "^4.1.2"
22  }
23 }

```



The terminal shows a Windows PowerShell session. The user navigates to the 'client' directory and runs the command to start the application:

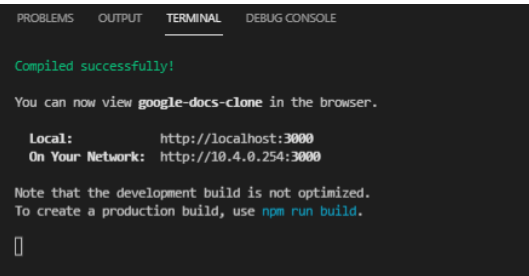
```

Windows PowerShell
Copyright (C) Microsoft Corporation. Tous droits réservés.

Testez le nouveau système multiplateforme PowerShell https://aka.ms/pscore6

PS D:\Laragon\Laragon\www\google-docs-clone> cd client
PS D:\Laragon\Laragon\www\google-docs-clone\client> npm run start

```



The terminal shows the output of the 'npm run start' command. It indicates that the application was compiled successfully and provides the local and network URLs to view the application in a browser:

```

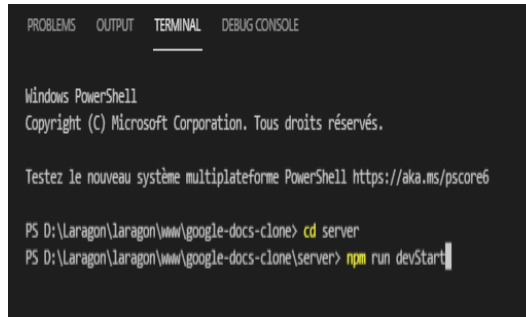
Compiled successfully!

You can now view google-docs-clone in the browser.

Local:      http://localhost:3000
On Your Network: http://10.4.0.254:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

```



The terminal shows a Windows PowerShell session. The user navigates to the 'server' directory and runs the command to start the development server:

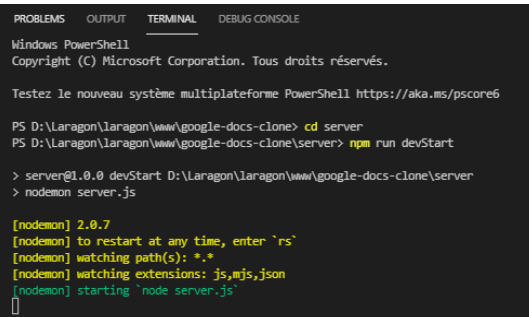
```

Windows PowerShell
Copyright (C) Microsoft Corporation. Tous droits réservés.

Testez le nouveau système multiplateforme PowerShell https://aka.ms/pscore6

PS D:\Laragon\Laragon\www\google-docs-clone> cd server
PS D:\Laragon\Laragon\www\google-docs-clone\server> npm run devStart

```



The terminal shows the output of the 'npm run devStart' command. It indicates that the development server is running using nodemon, which is watching for file changes and restarting the server automatically:

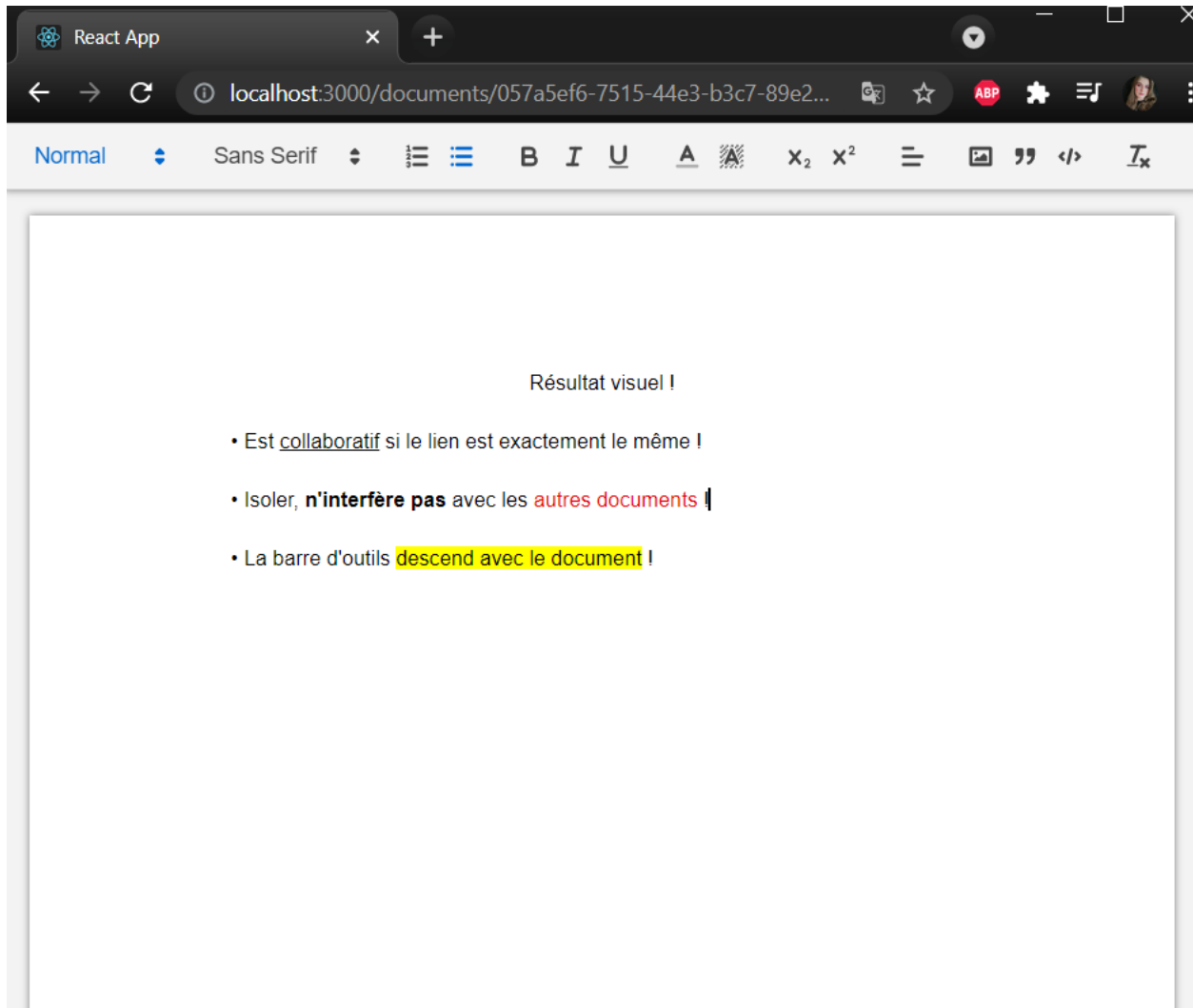
```

> server@1.0.0 devStart D:\Laragon\Laragon\www\google-docs-clone\server
> nodemon server.js

[nodemon] 2.0.7
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node server.js`

```





MARDI 15 &amp; MERCREDI 16 JUIN 2021

## MISSIONS :

1. Chercher comment faire un **Add-In Word** pour pouvoir le lier par la suite à une api Laravel.  
Aboutir à la création de cette Add-In.

## REALISATIONS :

1.
  - a. Recherche par rapport à la création d'un Add-In Word

Au début il y'a eu une confusion avec notre maître de stage car il avait employé le terme d'Add-On. Mais, après plusieurs recherches, on a pu trouver la documentation Microsoft et son tutoriel pour créer un Add-In. J'ai aussi trouvé le même tutoriel sur GitHub qui personnellement était plus simple à suivre pour moi.

## Liens parcourus :

<https://docs.microsoft.com/fr-fr/javascript/api/overview?view=word-js-preview>  
<https://docs.microsoft.com/fr-fr/javascript/api/word?view=word-js-preview>  
<https://docs.microsoft.com/fr-fr/javascript/api/word/word.application?view=word-js-preview>  
<https://docs.microsoft.com/en-us/javascript/api/office?view=excel-js-preview> (common)  
<https://docs.microsoft.com/en-us/office/dev/add-ins/develop/understanding-the-javascript-api-for-office>  
<https://www.youtube.com/watch?v=D8zIHntMSmE>  
[https://www.youtube.com/watch?v=T3Px88x\\_PsA](https://www.youtube.com/watch?v=T3Px88x_PsA)

## Liens retenus :

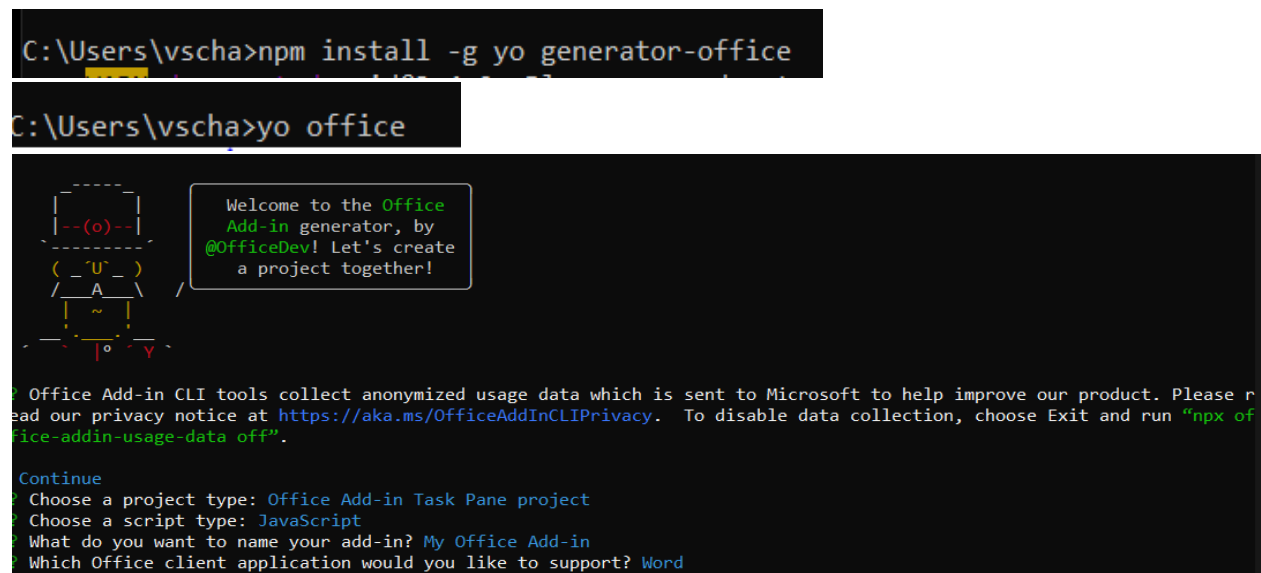
<https://github.com/OfficeDev/office-js-docs-pr/blob/master/docs/tutorials/word-tutorial.md>  
<https://docs.microsoft.com/en-us/office/dev/add-ins/tutorials/word-tutorial>

- b. Tutoriel et application des recherches :

J'ai commencé par installer **Yeoman Generator** grâce à **NPM** afin de pouvoir générer le projet de l'Add-In Word selon les critères de Microsoft Office.

```
C:\Users\vscha>npm install -g yo generator-office

C:\Users\vscha>yo office
```



```

Welcome to the Office
Add-in generator, by
@OfficeDev! Let's create
a project together!

? Office Add-in CLI tools collect anonymized usage data which is sent to Microsoft to help improve our product. Please read our privacy notice at https://aka.ms/OfficeAddInCLIPrivacy. To disable data collection, choose Exit and run "npm run office-addin-usage-data off".
Continue
? Choose a project type: Office Add-in Task Pane project
? Choose a script type: JavaScript
? What do you want to name your add-in? My Office Add-in
? Which Office client application would you like to support? Word

```

Puis j'ai suivi étape par étape le tutoriel GitHub et j'ai abouti à ceci :

```

1 <!-- Copyright (c) Microsoft Corporation. All rights reserved. Licensed under the MIT license. -->
2 <!-- This file shows how to design a first-run page that provides a welcome screen to the user about the features of the add-in. -->
3
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
16  <!-- For more information on Fluent UI, visit https://developer.microsoft.com/fluentui#/. -->
17  <link rel="stylesheet" href="https://static2.sharepointonline.com/files/fabric/office-ui-fabric-core/9.6.1/css/fabric.min.css"/>
18
19  <!-- Template styles -->
20  <link href="taskpane.css" rel="stylesheet" type="text/css" />
21 </head>
22
23 <body class="ms-font-m ms-welcome ms-Fabric">
24   <header class="ms-welcome_header ms-bgColor-neutrallighter">
25     
26     <h1 class="ms-font-su">Welcome</h1>
27   </header>
28   <section id="sideload-msg" class="ms-welcome_main">
29     <h2 class="ms-font-xl">Please sideload your add-in to see app body.</h2>
30   </section>
31   <main id="app-body" class="ms-welcome_main" style="display: none;">
32     <button class="ms-Button" id="insert-paragraph">Insert Paragraph</button><br/><br/>
33     <button class="ms-Button" id="apply-style">Apply Style</button><br/><br/>
34     <button class="ms-Button" id="apply-custom-style">Apply Custom Style</button><br/><br/>
35     <button class="ms-Button" id="change-font">Change Font</button><br/><br/>
36     <button class="ms-Button" id="insert-text-into-range">Insert Abbreviation</button><br/><br/>
37     <button class="ms-Button" id="insert-text-outside-range">Add Version Info</button><br/><br/>
38     <button class="ms-Button" id="replace-text">Change Quantity Term</button><br/><br/>
39     <button class="ms-Button" id="insert-image">Insert Image</button><br/><br/>
40     <button class="ms-Button" id="insert-html">Insert HTML</button><br/><br/>
41     <button class="ms-Button" id="insert-table">Insert Table</button><br/><br/>
42     <button class="ms-Button" id="create-content-control">Create Content Control</button><br/><br/>
43     <button class="ms-Button" id="replace-content-in-control">Rename Service</button><br/><br/>
44   </main>
45 </body>
46
47 </html>

```

```

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

```

The image displays two screenshots of a Visual Studio Code editor interface, showing the development of an Office extension. The left sidebar shows the Explorer view with the project structure for 'MY OFFICE ADD-IN'.

**Top Screenshot:** The editor shows the `taskpane.js` file. The code defines two functions: `insertParagraph` and `applyStyle`. Both functions use `Word.run` to interact with the Office document. `insertParagraph` inserts a new paragraph with the text "Office has many versions, including Office 2016, Office 2019, Office 365 Click-to-Run (C2R), Office on the web.", followed by "Start". `applyStyle` applies the `intenseReference` style to the first paragraph. Both functions include error handling using `.catch` and `console.log`.

```

38
39 function insertParagraph() {
40   Word.run(function (context) {
41
42     var docBody = context.document.body;
43     docBody.insertParagraph(
44       "Office has many versions, including Office 2016, Office 2019, Office 365 Click-to-Run (C2R), Office on the web.",
45       "Start"
46     );
47
48     return context.sync();
49   })
50   .catch(function (error) {
51     console.log("Error: " + error);
52     if (error instanceof OfficeExtension.Error) {
53       console.log("Debug info: " + JSON.stringify(error.debugInfo));
54     }
55   });
56 }
57
58 function applyStyle() {
59   Word.run(function (context) {
60
61     var firstParagraph = context.document.body.paragraphs.getFirst();
62     firstParagraph.styleBuiltIn = Word.Style.intenseReference;
63
64     return context.sync();
65   })
66   .catch(function (error) {
67     console.log("Error: " + error);
68     if (error instanceof OfficeExtension.Error) {
69       console.log("Debug info: " + JSON.stringify(error.debugInfo));
70     }
71   });
72 }
73

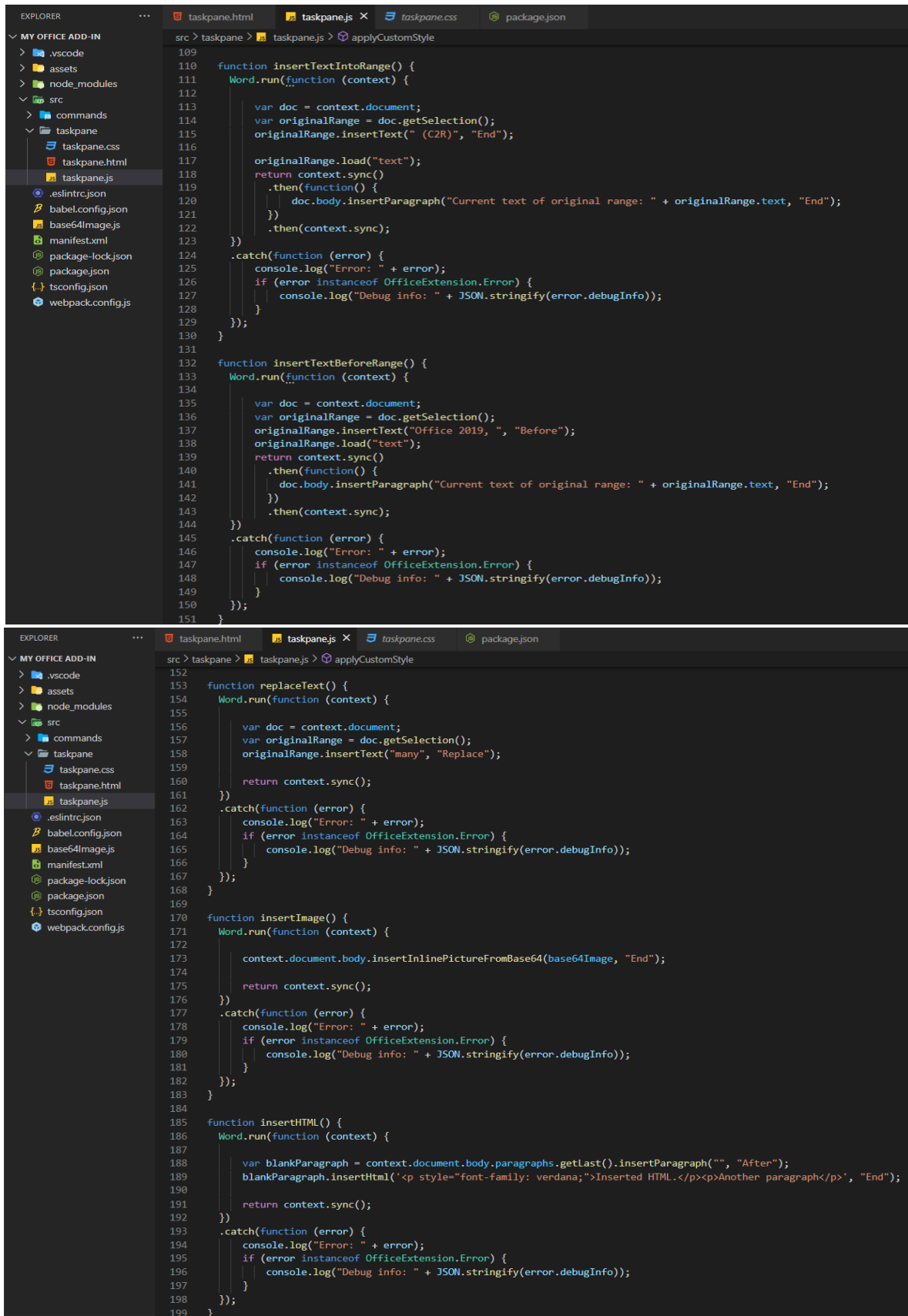
```

**Bottom Screenshot:** The editor shows the `taskpane.js` file. The code defines two functions: `applyCustomStyle` and `changeFont`. `applyCustomStyle` applies the `MyCustomStyle` style to the last paragraph. `changeFont` changes the font of the second paragraph to "Courier New", bold, and size 18. Both functions include error handling using `.catch` and `console.log`.

```

74 function applyCustomStyle() {
75   Word.run(function (context) {
76
77     var lastParagraph = context.document.body.paragraphs.getLast();
78     lastParagraph.style = "MyCustomStyle";
79
80     return context.sync();
81   })
82   .catch(function (error) {
83     console.log("Error: " + error);
84     if (error instanceof OfficeExtension.Error) {
85       console.log("Debug info: " + JSON.stringify(error.debugInfo));
86     }
87   });
88 }
89
90 function changeFont() {
91   Word.run(function (context) {
92
93     var secondParagraph = context.document.body.paragraphs.getFirst().getNext();
94     secondParagraph.font.set({
95       name: "Courier New",
96       bold: true,
97       size: 18
98     });
99
100    return context.sync();
101  })
102  .catch(function (error) {
103    console.log("Error: " + error);
104    if (error instanceof OfficeExtension.Error) {
105      console.log("Debug info: " + JSON.stringify(error.debugInfo));
106    }
107  });
108 }
109

```



```

109
110 function insertTextIntoRange() {
111     Word.run(function (context) {
112
113         var doc = context.document;
114         var originalRange = doc.getSelection();
115         originalRange.insertText(" (C2R)", "End");
116
117         originalRange.load("text");
118         return context.sync()
119             .then(function () {
120                 doc.body.insertParagraph("Current text of original range: " + originalRange.text, "End");
121             })
122             .then(context.sync);
123     });
124     .catch(function (error) {
125         console.log("Error: " + error);
126         if (error instanceof OfficeExtension.Error) {
127             console.log("Debug info: " + JSON.stringify(error.debugInfo));
128         }
129     });
130 }
131
132 function insertTextBeforeRange() {
133     Word.run(function (context) {
134
135         var doc = context.document;
136         var originalRange = doc.getSelection();
137         originalRange.insertText("Office 2019, ", "Before");
138         originalRange.load("text");
139         return context.sync()
140             .then(function () {
141                 doc.body.insertParagraph("Current text of original range: " + originalRange.text, "End");
142             })
143             .then(context.sync);
144     });
145     .catch(function (error) {
146         console.log("Error: " + error);
147         if (error instanceof OfficeExtension.Error) {
148             console.log("Debug info: " + JSON.stringify(error.debugInfo));
149         }
150     });
151 }

```

```

152
153 function replaceText() {
154     Word.run(function (context) {
155
156         var doc = context.document;
157         var originalRange = doc.getSelection();
158         originalRange.insertText("many", "Replace");
159
160         return context.sync();
161     });
162     .catch(function (error) {
163         console.log("Error: " + error);
164         if (error instanceof OfficeExtension.Error) {
165             console.log("Debug info: " + JSON.stringify(error.debugInfo));
166         }
167     });
168 }
169
170 function insertImage() {
171     Word.run(function (context) {
172
173         context.document.body.insertInlinePictureFromBase64(base64Image, "End");
174
175         return context.sync();
176     });
177     .catch(function (error) {
178         console.log("Error: " + error);
179         if (error instanceof OfficeExtension.Error) {
180             console.log("Debug info: " + JSON.stringify(error.debugInfo));
181         }
182     });
183 }
184
185 function insertHTML() {
186     Word.run(function (context) {
187
188         var blankParagraph = context.document.body.paragraphs.getLast().insertParagraph("", "After");
189         blankParagraph.insertHtml('<p style="font-family: verdana;">Inserted HTML.</p><p>Another paragraph</p>', "End");
190
191         return context.sync();
192     });
193     .catch(function (error) {
194         console.log("Error: " + error);
195         if (error instanceof OfficeExtension.Error) {
196             console.log("Debug info: " + JSON.stringify(error.debugInfo));
197         }
198     });
199 }

```

The image displays two screenshots of a Visual Studio Code editor interface, showing the development of a Word add-in. The Explorer pane on the left shows the project structure, including files like .vscode, assets, node\_modules, src, commands, taskpane, taskpane.css, taskpane.html, taskpane.js, .eslintrc.json, babel.config.json, base64Image.js, manifest.xml, package-lock.json, package.json, tsconfig.json, and webpack.config.js.

The top screenshot shows the `taskpane.js` file with the `insertTable` function. The function uses `Word.run` to insert a table into the document. The table data is defined as an array of objects, and the function uses `context.document.body.paragraphs.getFirst().getNext()` to find the insertion point. The function also includes error handling and a `context.sync()` call.

```

201 function insertTable() {
202   Word.run(function (context) {
203
204     var secondParagraph = context.document.body.paragraphs.getFirst().getNext();
205
206     var tableData = [
207       ["Name", "ID", "Birth City"],
208       ["Bob", "434", "Chicago"],
209       ["Sue", "719", "Havana"],
210     ];
211     secondParagraph.insertTable(3, 3, "After", tableData);
212
213     return context.sync();
214   })
215   .catch(function (error) {
216     console.log("Error: " + error);
217     if (error instanceof OfficeExtension.Error) {
218       console.log("Debug info: " + JSON.stringify(error.debugInfo));
219     }
220   });
221 }
222
223
224 function createContentControl() {
225   Word.run(function (context) {
226
227     var serviceNameRange = context.document.getSelection();
228     var serviceNameContentControl = serviceNameRange.insertContentControl();
229     serviceNameContentControl.title = "Service Name";
230     serviceNameContentControl.tag = "serviceName";
231     serviceNameContentControl.appearance = "Tags";
232     serviceNameContentControl.color = "blue";
233
234     return context.sync();
235   })
236   .catch(function (error) {
237     console.log("Error: " + error);
238     if (error instanceof OfficeExtension.Error) {
239       console.log("Debug info: " + JSON.stringify(error.debugInfo));
240     }
241   });
242 }
243

```

The bottom screenshot shows the `taskpane.js` file with the `replaceContentInControl` function. The function uses `Word.run` to replace the content of a content control. The function uses `context.document.contentControls.getByTag("serviceName").getFirst()` to find the content control. The function also includes error handling and a `context.sync()` call.

```

243
244 function replaceContentInControl() {
245   Word.run(function (context) {
246
247     var serviceNameContentControl = context.document.contentControls.getByTag("serviceName").getFirst();
248     serviceNameContentControl.insertText("Fabrikam Online Productivity Suite", "Replace");
249
250     return context.sync();
251   })
252   .catch(function (error) {
253     console.log("Error: " + error);
254     if (error instanceof OfficeExtension.Error) {
255       console.log("Debug info: " + JSON.stringify(error.debugInfo));
256     }
257   });
258 }
259

```

## SCHALCKENS Valentine – BTS SIO – SLAM

```

PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

Testez le nouveau système multiplateforme PowerShell https://aka.ms/pscore6

PS D:\Laragon\Laragon\www\My Office Add-in> npm start

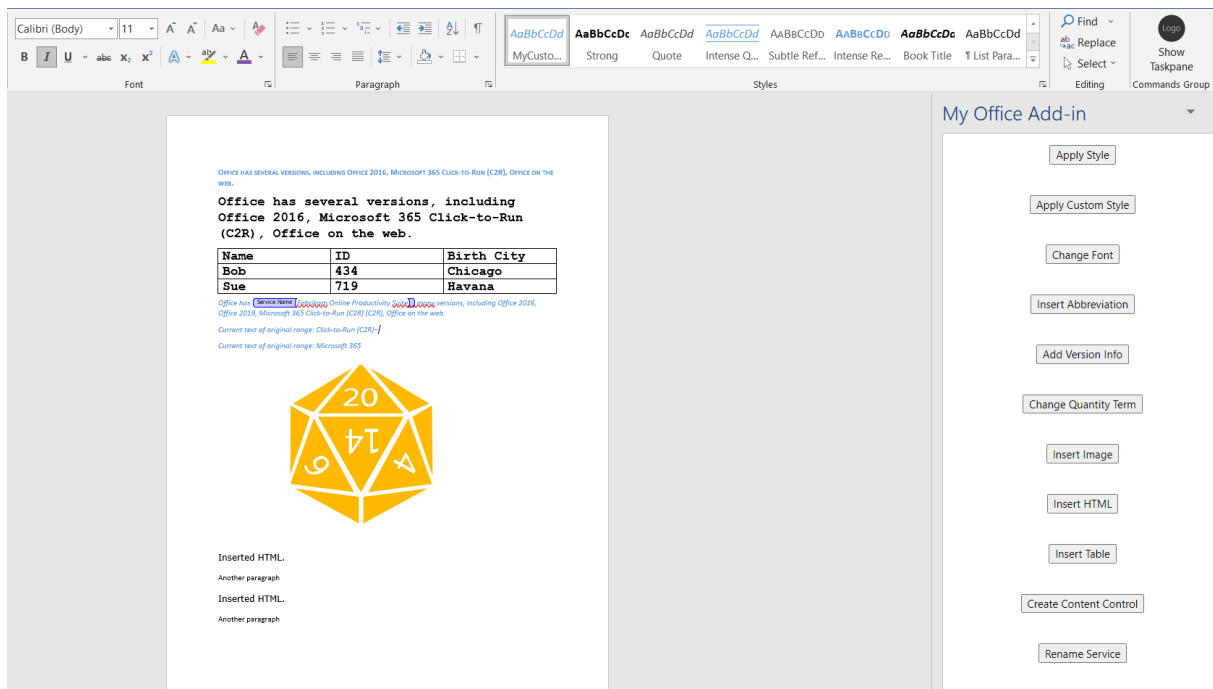
> office-addin-taskpane-js@0.0.1 start D:\Laragon\Laragon\www\My Office Add-in
> office-addin-debugging start manifest.xml

Debugging is being started...
App type: desktop
? Allow localhost loopback for Microsoft Edge WebView? No
Enabled debugging for add-in 44e85735-bb3d-465e-b563-28d19c5756e9.
Starting the dev server... (webpack-dev-server --mode development)
The dev server is running on port 3000. Process id: 16436
Sideloading the Office Add-in...
Debugging started.
[]

(node) "D:\Laragon\Laragon\www\My Office Add-in\node_modules\bin\webpack-dev-server\bin\webpack-dev-server.js" --mod...
You already have trusted access to https://localhost.
Certificate: C:\Users\valsch\office-addin-dev-certs\localhost.crt
Key: C:\Users\valsch\office-addin-dev-certs\localhost.key
Status: Project is running at https://localhost:3000/
Status: webpack output is served from /
Status: Content not from webpack is served from D:\Laragon\Laragon\www\My Office Add-in
Status: Hash: 5ab8dfe79b1141edcb
Version: webpack 4.46.0
Time: 1831ms
Built at: 06/12/2021 18:36:49 AM
    Asset      Size      Chunks             Chunk Names
assets/icon-16.png  1.56 KiB          0             [emitted]
assets/icon-32.png  2.33 KiB          0             [emitted]
assets/icon-80.png  4.72 KiB          0             [emitted]
assets/logo-filled.png  11.5 KiB          0             [emitted]
  commands.html  473 bytes          0             [emitted]
  commands.js    356 KiB          0             [emitted]
  commands.js.map  488 KiB          0             [emitted] [dev]
manifest.dev.xml  4.89 KiB          0             [emitted]
polyfill.js     991 KiB          0             [emitted]
polyfill.js.map  944 KiB          0             [emitted] [dev]
taskpane.css    1.48 KiB          0             [emitted]
taskpane.html   2.39 KiB          0             [emitted]
taskpane.js     376 KiB          0             [emitted]
taskpane.js.map  424 KiB          0             [emitted] [dev]
Entrypoint polyfill = polyfill.js polyfill.js.map
Entrypoint taskpane = taskpane.js taskpane.js.map
Entrypoint commands = commands.js commands.js.map
[0] multi (webpack)-dev-server/client?https://localhost:3000 core-js/stable regenerator-runtime/runtime 52 bytes [polyfill] [built]

[3] multi (webpack)-dev-server/client?https://localhost:3000 ./src/commands/commands.js 40 bytes {commands} [built]
[./assets/icon-16.png] 64 bytes {taskpane} [built]
[./assets/icon-32.png] 64 bytes {taskpane} [built]
[./assets/icon-80.png] 64 bytes {taskpane} [built]
[./base64Image.js] 11.5 KiB {taskpane} [built]
[./node_modules/core-js/es/index.js] 8.89 KiB {polyfill} [built]
[./node_modules/core-js/internals/path.js] 71 bytes {polyfill} [built]
[./node_modules/core-js/stable/index.js] 102 bytes {polyfill} [built]
[./node_modules/core-js/web/index.js] 398 bytes {polyfill} [built]
[./node_modules/regenerator-runtime/runtime.js] 24 KiB {polyfill} [built]
[./node_modules/webpack-dev-server/client/index.js?https://localhost:3000] (webpack)-dev-server/client?https://localhost:3000 4.29 KiB {polyfill} {taskpane} {commands} [built]
[./src/commands/commands.js] 1.06 KiB {commands} [built]
[./src/taskpane/taskpane.js] 7.55 KiB {taskpane} [built]
+ 410 hidden modules
Child HtmlWebpackCompiler:
    Asset      Size      Chunks             Chunk Names
assets/logo-filled.png  11.6 KiB          0             [emitted]
+ 2 hidden assets
Entrypoint HtmlWebpackPlugin_0 = __child-HtmlWebpackPlugin_0
Entrypoint HtmlWebpackPlugin_1 = __child-HtmlWebpackPlugin_1
[./assets/logo-filled.png] 68 bytes {HtmlWebpackPlugin_0} [built]
[./node_modules/html-webpack-plugin/lib/loader.js!./src/commands/commands.html] 450 bytes {HtmlWebpackPlugin_1} [built]
[./node_modules/html-webpack-plugin/lib/loader.js!./src/taskpane/taskpane.html] 2.71 KiB {HtmlWebpackPlugin_0} [built]
[0] wdm: Compiled successfully.
  
```

## Résultat visuel dans Word :



JEUDI 17 JUIN 2021

## MISSIONS :

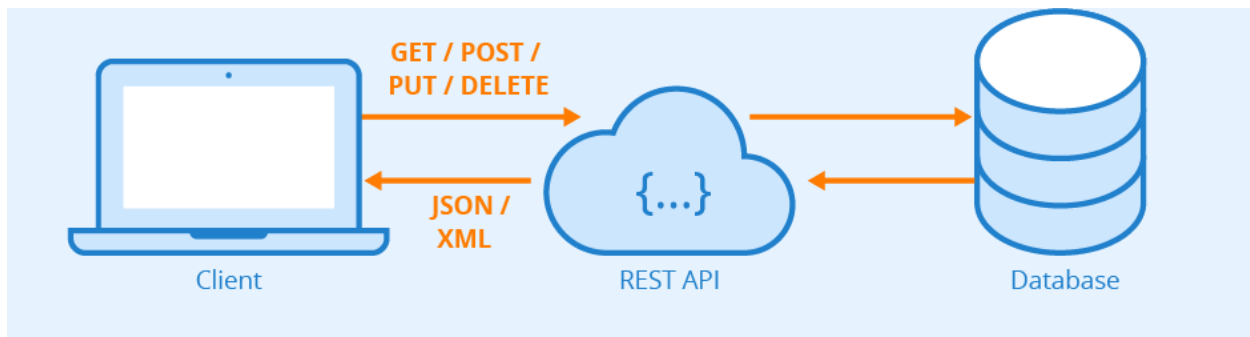
1. Chercher comment relier cet Add-In à notre projet.
2. Faire une **API** qui permet de mettre dans un document Word des paragraphes enregistrés en base de données.

## REALISATIONS :

1. Recherches :

D'après mes recherches, la solution pour relier cet Add-In à notre projet serait de passer par une **API**. Une **API** permet de faire le lien entre une partie **Frontend (Client)** et la partie **Backend (Server)**. La partie client peut choisir différents « Endpoint » proposés par l'**API** qui engendreront des actions et un retour de la partie serveur vers l'**API** qui redistribue l'information à la partie client.

Plus précisément on utilisera une **RESTful API (Representational State Transfer)**, cela représente en fait une forme d'architecture utilisée pour communiquer entre le client et le serveur. Ce type d'**API** utilise l'**HTTP** et les données qu'elle transmet sont en **JSON** (ou **XML** mais cela ne nous intéresse pas dans notre cas).



Source : seobility.net

## Liens retenus lors des recherches :

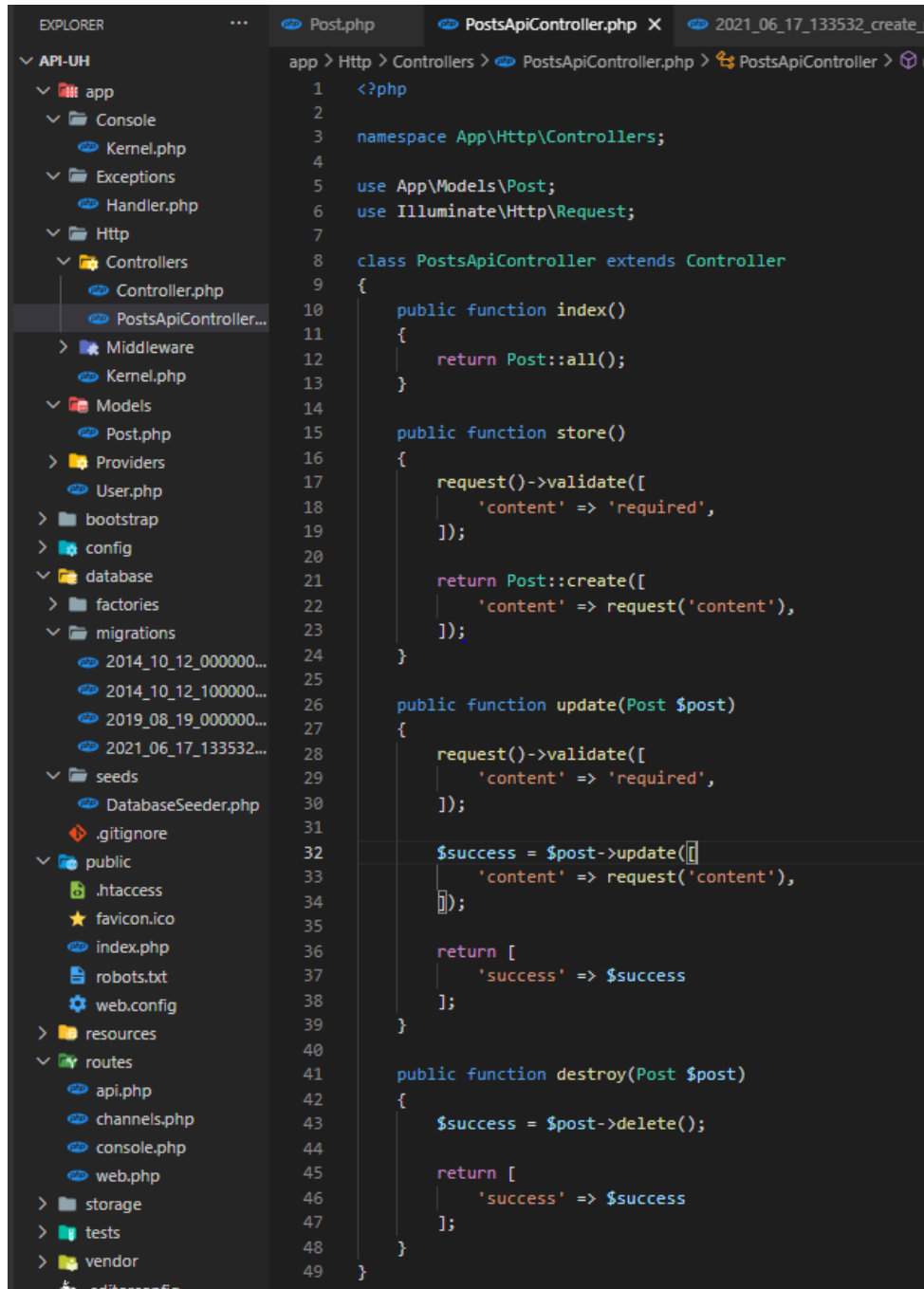
<https://www.youtube.com/watch?v=xvqPEEpRBj4>  
<https://www.youtube.com/watch?v=WDha52dbLWM>  
<https://laravel.com/docs/8.x/routing#view-routes>  
<https://laravel.com/docs/8.x/container#container-events>  
<https://laravel.sillo.org/cours-laravel-8-les-donnees-les-ressources-api/>  
[https://www.seobility.net/en/wiki/REST\\_API](https://www.seobility.net/en/wiki/REST_API)



## 2. Application :

Après avoir effectué la première mission, j'ai demandé à mon maitre de stage la prochaine étape : « *Pour le moment j'aimerais que vous fassiez une API qui permet de mettre dans un document Word des paragraphes enregistrés en base de données.* »

J'ai donc commencé par créer un nouveau projet avec la structure qui me semblait adéquate (similaire à la première semaine de stage) :

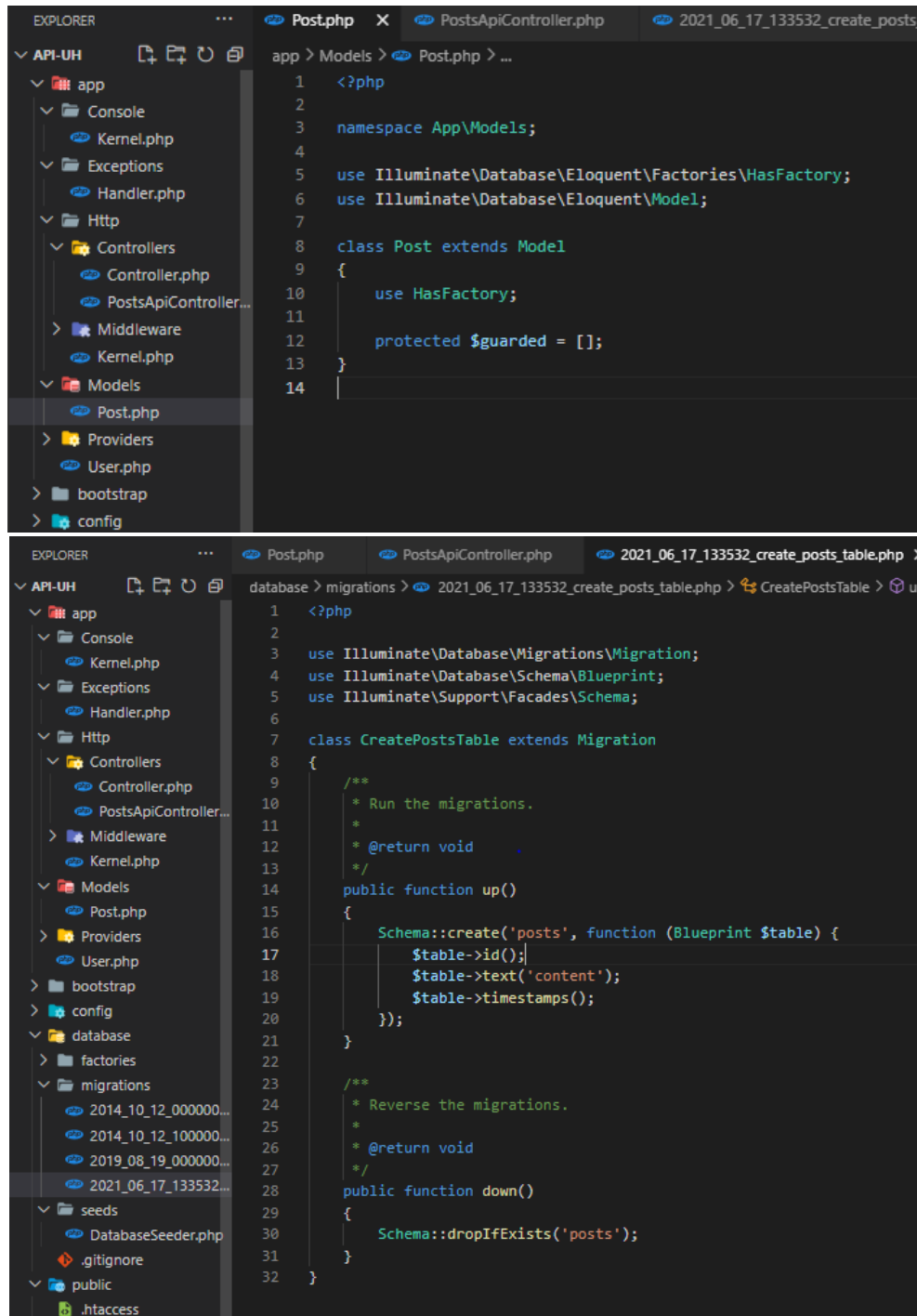


The screenshot shows a code editor with a file explorer on the left and a PHP file open in the center. The file explorer shows a project structure for 'API-UH' with folders like 'app', 'Middleware', 'Models', 'Providers', 'bootstrap', 'config', 'database', 'factories', 'migrations', 'seeds', 'public', 'resources', 'routes', 'storage', 'tests', and 'vendor'. The 'app' folder is expanded, showing 'Controllers' and 'PostsApiController.php'. The 'PostsApiController.php' file is open, showing the following code:

```

1  <?php
2
3  namespace App\Http\Controllers;
4
5  use App\Models\Post;
6  use Illuminate\Http\Request;
7
8  class PostsApiController extends Controller
9  {
10     public function index()
11     {
12         return Post::all();
13     }
14
15     public function store()
16     {
17         request()->validate([
18             'content' => 'required',
19         ]);
20
21         return Post::create([
22             'content' => request('content'),
23         ]);
24     }
25
26     public function update(Post $post)
27     {
28         request()->validate([
29             'content' => 'required',
30         ]);
31
32         $success = $post->update([
33             'content' => request('content'),
34         ]);
35
36         return [
37             'success' => $success
38         ];
39     }
40
41     public function destroy(Post $post)
42     {
43         $success = $post->delete();
44
45         return [
46             'success' => $success
47         ];
48     }
49 }

```



Après avoir fait la structure et la base de données avec des paragraphes pré-enregistrés ('content'), j'ai cherché comment faire le lien entre cette API et l'Add-In que nous avons créé. Je sais maintenant qu'il faut passer par le fichier '**routes/api.php**', mais je n'ai pas encore saisi comment réaliser ces routes correctement. Je continue donc mes recherches.

VENDREDI 18 JUIN 2021

MISSIONS :

1. Continuer sur les activités en cours.

REALISATIONS :

- 1.

**En Cours ...**