[Date]

Zenidoc

[Nom de la société]

[Adresse de la société]

[Titre du document]

[Sous-titre du document]

Les informations contenues dans ce document pourront faire l’objet de modifications.

Aucune partie de ce manuel ne peut être reproduite ou transmise à quelque fin ou par quelque moyen que ce soit, électronique ou mécanique, sans la permission expresse de **Zenidoc** S.A.

*© 2022* ***Zenidoc*** *S.A. Tous droits réservés.*

*Zenidoc® est une marque déposée de Zenidoc S.A.*

Table des matières

I - Description du travail proposé3

A – Le besoin3

B – La solution proposée3

II - Description du travail réalisé4

A – Vuetify 4

B – Axios 4

C – Adaptations 5

1 – Json-server 5

2 – Versions utilisées 5

3 – Format de données à respecter5

III – Commandes pour le lancement du projet6

IV - Bibliographie6

**I – Description du travail proposé**

**A – Le besoin**

Ce projet est la nouvelle version, la 8ème, de la solution web de Zenidoc. Elle fonctionne en lien avec le portail backend au travers d'une API dédiée, la PortalAPI. Ce frontend repose sur le framework Vue CLI qui permet d'aboutir après compilation à un ensemble de fichiers HTML/CSS/JS compatibles avec n'importe quel navigateur. Ce projet a été réalisé avec la version 2.6.11 de Vue.js. La structure des composants du projet existant est disponible en annexe 3 et 4, ce diagramme est à titre purement illustratif et ne respecte pas les normes UML.

Le but de ce projet est ainsi de reprendre un projet préexistant, en permettant l’affichage de documents sélectionnés grâce à un arborescence de fichiers intégrés, un TreeView. Les données contenues dans cette arborescence seront récupérées depuis le serveur et seront au format JSON.

**B – La solution proposée**

Réalisation d’un ensemble de composants pouvant être intégrés au projet principal. Ces composants devront pouvoir afficher une arborescence de fichiers et afficher également ces fichiers grâce à des données reçues au format JSON.

Puisqu’il est impossible de tester les composants en conditions réelles, c’est-à-dire en les intégrant directement dans le projet principal, il faudra développer une solution intégrant des outils permettant de recréer des conditions réelles (en générant le contenu du TreeView avec des données dynamiques au format JSON, et non pas statiques avec des données contenues dans le data par exemple).

L’actuel composant TreeView ainsi que le projet principal dans sa globalité possèdent une présentation syntaxique particulière. Si la présentation du code peut différer de celle que l’on peut croiser habituellement, c’est parce que l’on utilise la librairie **vue-property-decorator** (Vue Class Component) de Vue.js, permettant de créer des components ayant ainsi une syntaxe similaire à des classes. Elle permet de rendre le code plus compréhensible. La solution proposée utilise également cette librairie pour faciliter son intégration au projet principal.

**II – Description du travail réalisé**

**A – Vuetify**

La nouvelle version du composant TreeView a été construite grâce à la librairie **Vuetify**. Cette librairie, permet de créer rapidement et avec facilité des interfaces utilisateurs grâce à une documentation très complète. Cette librairie est construite au-dessus de Vue.js, ce qui permet de rendre l’ensemble des composants construits avec **Vuetify** compatibles avec des composants Vue.js.

La structure et les diverses fonctionnalités de notre composant sont ainsi directement gérées et créées par **Vuetify**. Le résultat obtenu est visualisable en Annexe 1.

**B – Axios**

**Axios** est une bibliothèque JavaScript fonctionnant comme un client HTTP. Elle permet de communiquer avec des API en utilisant des requêtes. Elle permet ainsi la récupération des données depuis le serveur via une requête effectuée dans le composant en lui-même. Le projet initial n’utilisant pas Vuex mais bien Axios, il était inutile de concevoir une solution utilisant cette technologie.

On envoie une requête pour pouvoir construire l’arborescence de notre TreeView grâce aux données récupérées.

Dans le cycle de vie du composant, cet appel est effectué dès l’initialisation des injections et de la réactivité. Un schéma récapitulatif de ce cycle de vie est disponible en Annexe 2.

created() {

        axios

          .get('http://localhost:3000/items')

          .then(response => {

            console.log(response.data)

            this.items = response.data

          })

          .catch(error => {

            console.log('Il y a eu une erreur', error.response)

          })

      }

L’utilisation d’Axios étant faite dans le composant parent de TreeView (SideBar), elle permet la transmission des données au composant par son élément parent via des props, comme c’est le cas dans le projet principal.

**C – Vue Router**

**Vue Router** est le router officiel pour Vue.js. Il s'intègre aisément avec Vue.js pour faire des applications mono page avec Vue.js.

Utilisé dans le projet, il permet de définir des routes ainsi que le ou les composants à afficher en fonction de la route définie. La route /ged qui utilise comme composant de sidepanel le TreeView (qui est compris dans le SideBar.

path: '/ged',

      component: AppTemplateDefault,

      children: [

        {

          path: '',

          components: {

            header: () => import('./views/app.ged/Header.vue'),

            sidepanel: () => import('./views/app.ged/SideBar.vue'),

            content: () => import('./views/app.ged/Content.vue'),

          },

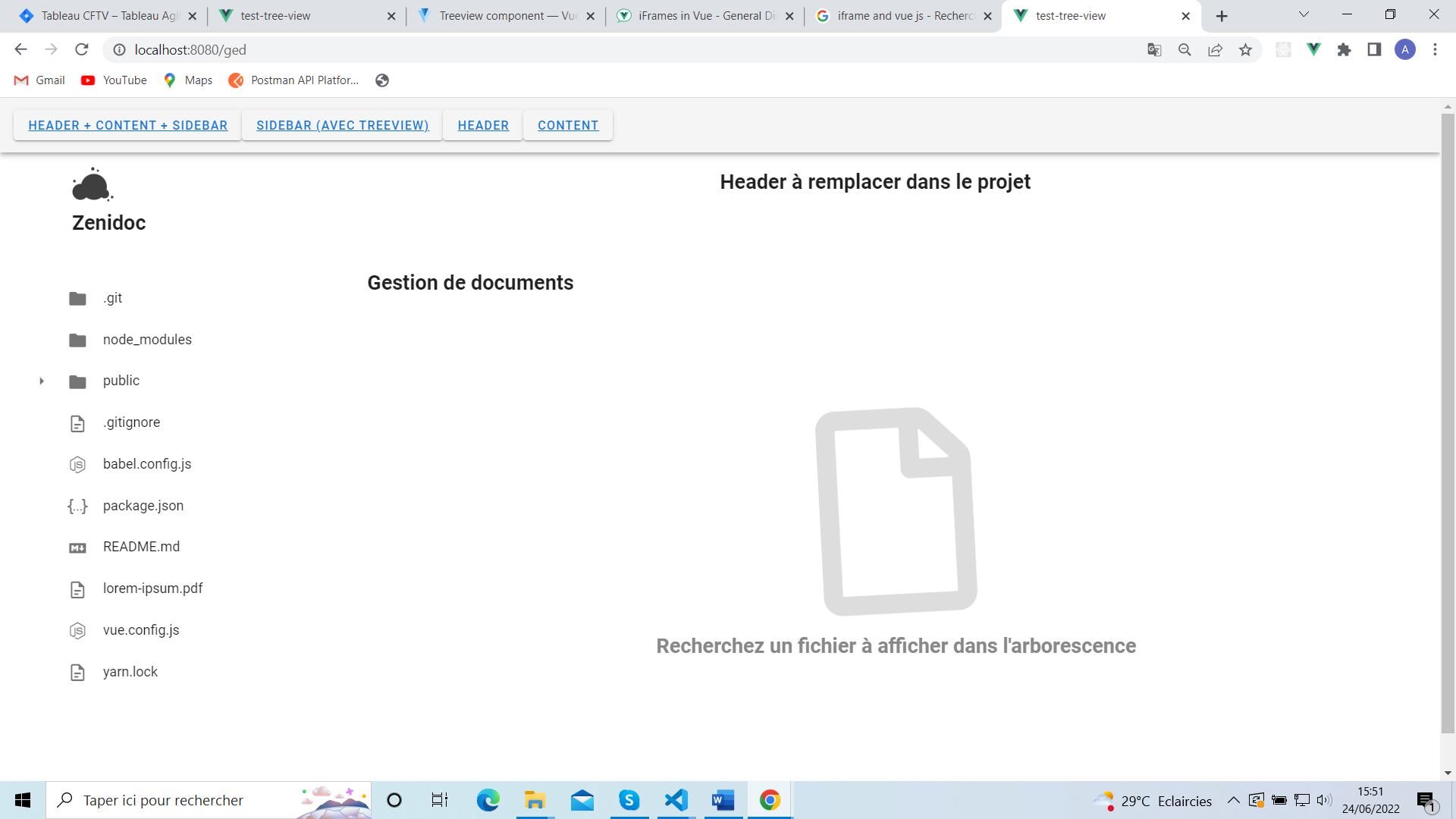
        },

      ],

    },

*Utilisation de Vue Router pour le chemin /ged dans le projet*

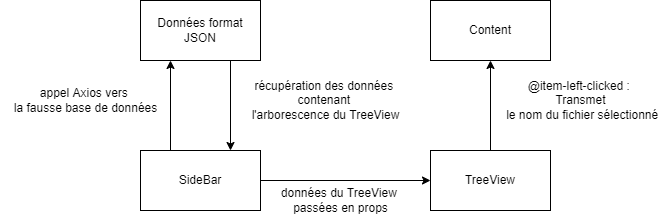
Nous retrouvons ainsi à la route /ged, l’interface suivante :



Cette interface combine l’ensemble des composants du projet, soit le SideBar (contenant le TreeView), le Content et le Header en une seule et unique page. Elle peut ainsi afficher divers types de fichiers dont la liste est disponible directement dans le composant TreeView. Un aperçu de l’ensemble de fichiers pouvant être affichés par le site est disponible en Annexe 6.

Pour résumer le fonctionnement de l’affichage du contenu très brièvement, lorsque l’on clique sur un élément du TreeView, le nom de cet élément est récupéré et est communiqué à l’ensemble des composants de l’application. Le composant Content récupère cette information et affiche l’élément correspondant.

Pour l’instant, dans le composant Content, cet élément renvoi à des liens statiques mais lorsque l’accès au serveur sera possible, on peut envisager que l’on pourra rechercher directement les liens menant à la ressource demandée grâce à un appel vers PortalAPI. Le schéma ci-dessous permet de mieux illustrer les relations qui existent entre chaque élément du projet.



**C – Adaptations**

Le composant n’étant pas, à l’heure actuelle, directement intégré au projet principal, l’utilisation de certains outils aura été nécessaire afin de permettre le bon fonctionnement de ce dernier. Ces outils seront potentiellement amenés à disparaître lorsque le composant aura été intégré au projet.

**1 – Fausse API REST : json-server**

Pour que le composant se comporte de la manière dont on le souhaite, c’est-à-dire qu’il puisse utiliser **Axios**, il était nécessaire de créer une fausse API REST renvoyant des données sur les dossiers/fichiers à afficher au format JSON. **json-server** permet ainsi de créer une fausse API sur laquelle on peut faire des appels pour récupérer des données au format JSON.

Ce serveur local ne sera alors plus nécessaire dès lors que le composant sera intégré au projet.

**2 – Détail des versions utilisées**

Pour assurer une pleine compatibilité entre le composant TreeView et le projet principal, les mêmes versions de dépendances ont été utilisées, soit pour les principales :

"vue": "^2.6.11",

"vue-json-tree-view": "^2.1.6",

"vue-loader": "^15.7.0",

"vue-property-decorator": "^8.4.0"

"axios": "^0.18.0",

"vue-router": "^3.1.6",

L’ensemble des versions utilisées est disponible dans les fichiers package-lock.json et package.json situés à la racine du projet.

**3 – Format de données à respecter**

Du fait que le TreeView soit complètement géré par une librairie externe, un format de données doit être respecté pour assurer le bon traitement des données par Vuetify.

Un exemple de données au format JSON pouvant être transmises avec succès au composant TreeView est disponible en Annexe 5.

Ces dernières doivent ainsi posséder un unique élément parent appelé ici « items », qui possèdera des enfants qui eux-mêmes pourront posséder des enfants à condition qu’ils soient tous compris dans un élément « children ». Ces derniers pourront également avoir des enfants et ainsi de suite. Un nœud doit être constitué du nom du fichier auquel il correspond suivi de son extension.

Également, concernant l’affichage de contenu, il faut que la source du iframe permettant d’afficher le contenu sélectionné, renvoi à une ressource externe et non pas une ressource stockée localement sur l’appareil. Car sinon, ce n’est pas la ressource demandée qui s’affichera mais la page actuelle.

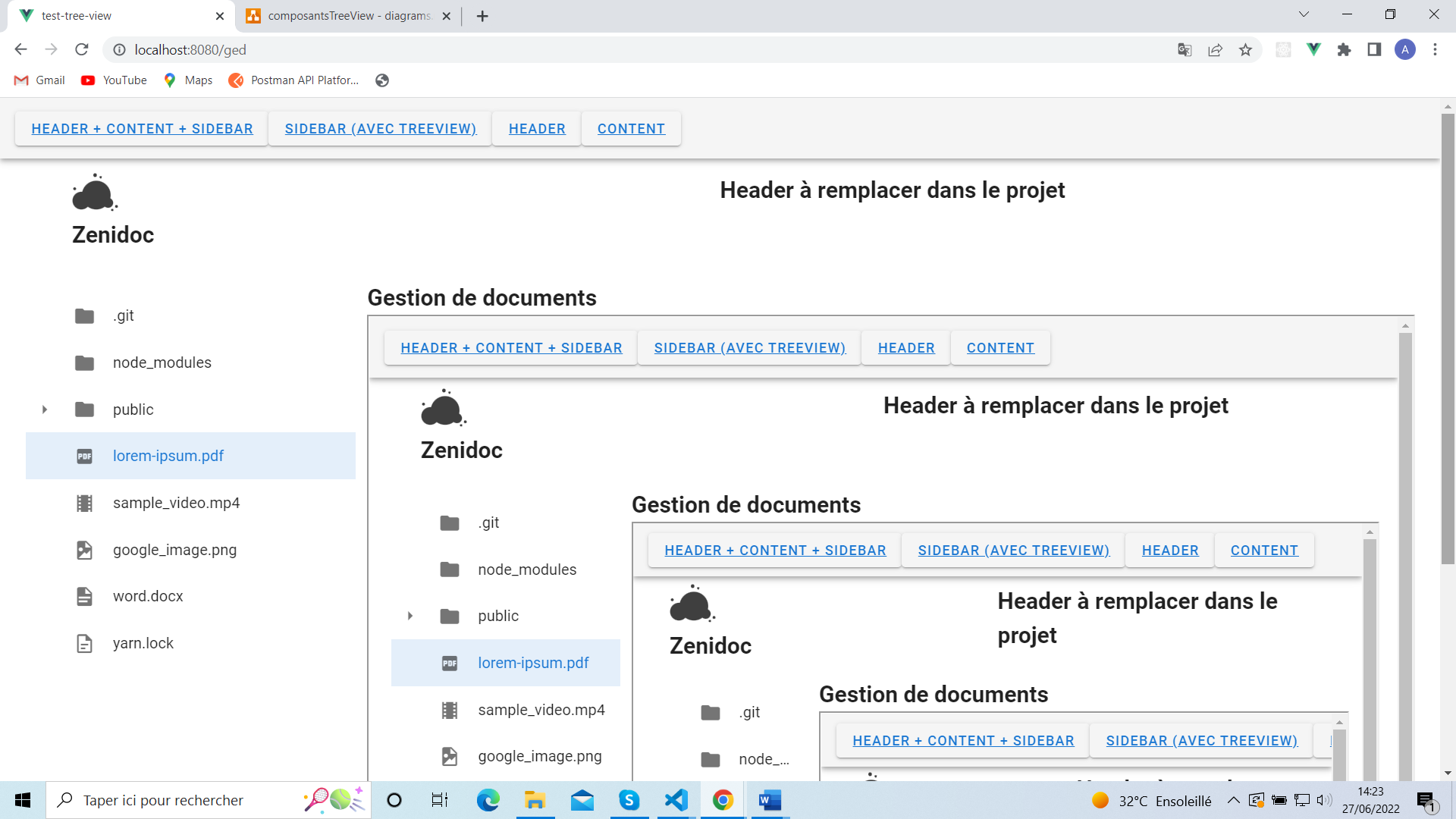
**Exemple** :

Fichier local en tant que source :

Code :

this.source = '@/fake\_files/lorem-ipsum.pdf';

Affichage Navigateur :



**III – Commandes pour le lancement du projet**

* Installer Axios

vue add axios@**0.18.0**

↑

(correspond à la version d’Axios installé sur le projet)

* Installer json-server

npm install -g json-server

* Lancer json-server

json-server --watch **db.json**

↑

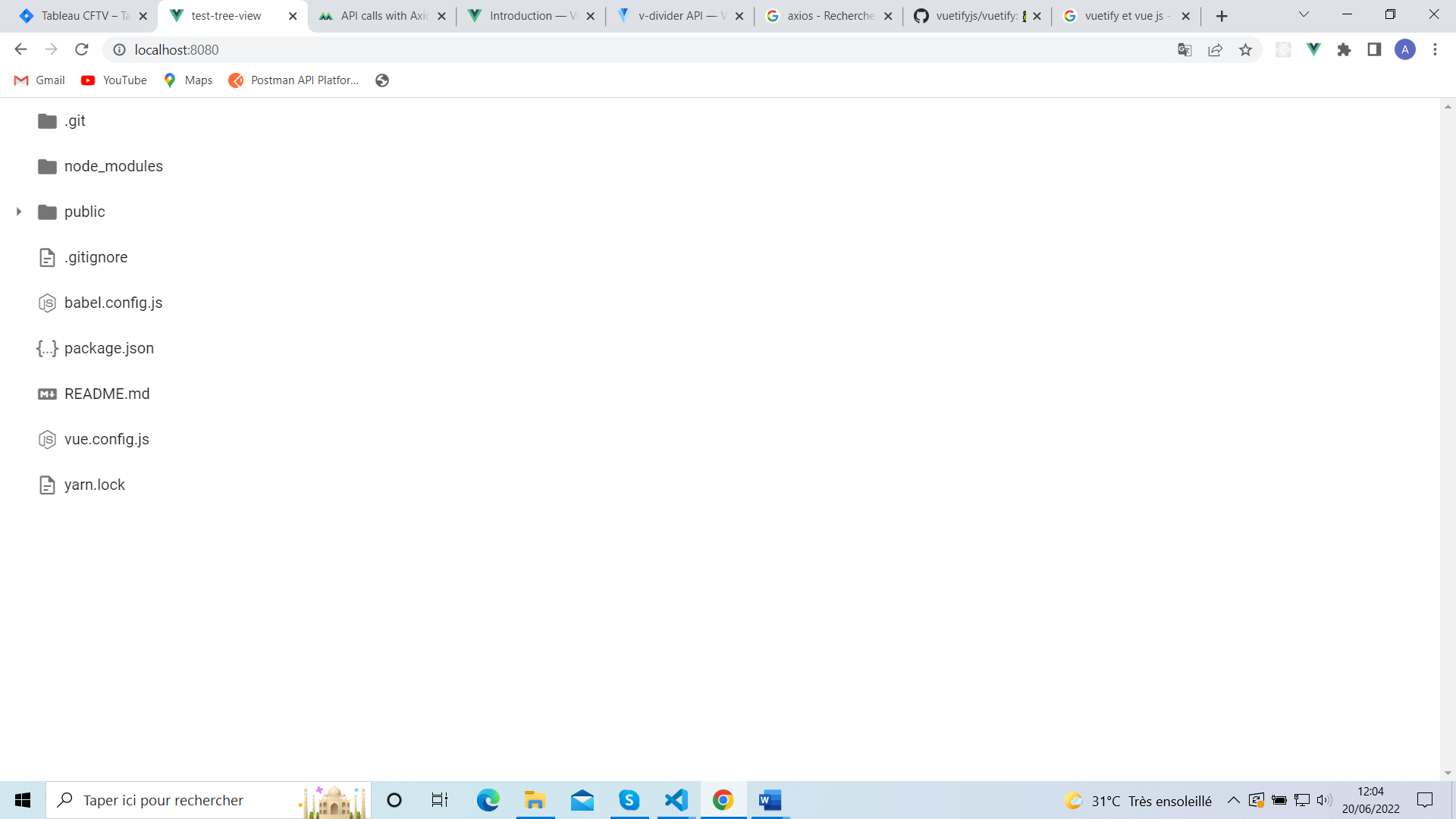
(nom du fichier contenant les données au format JSON)

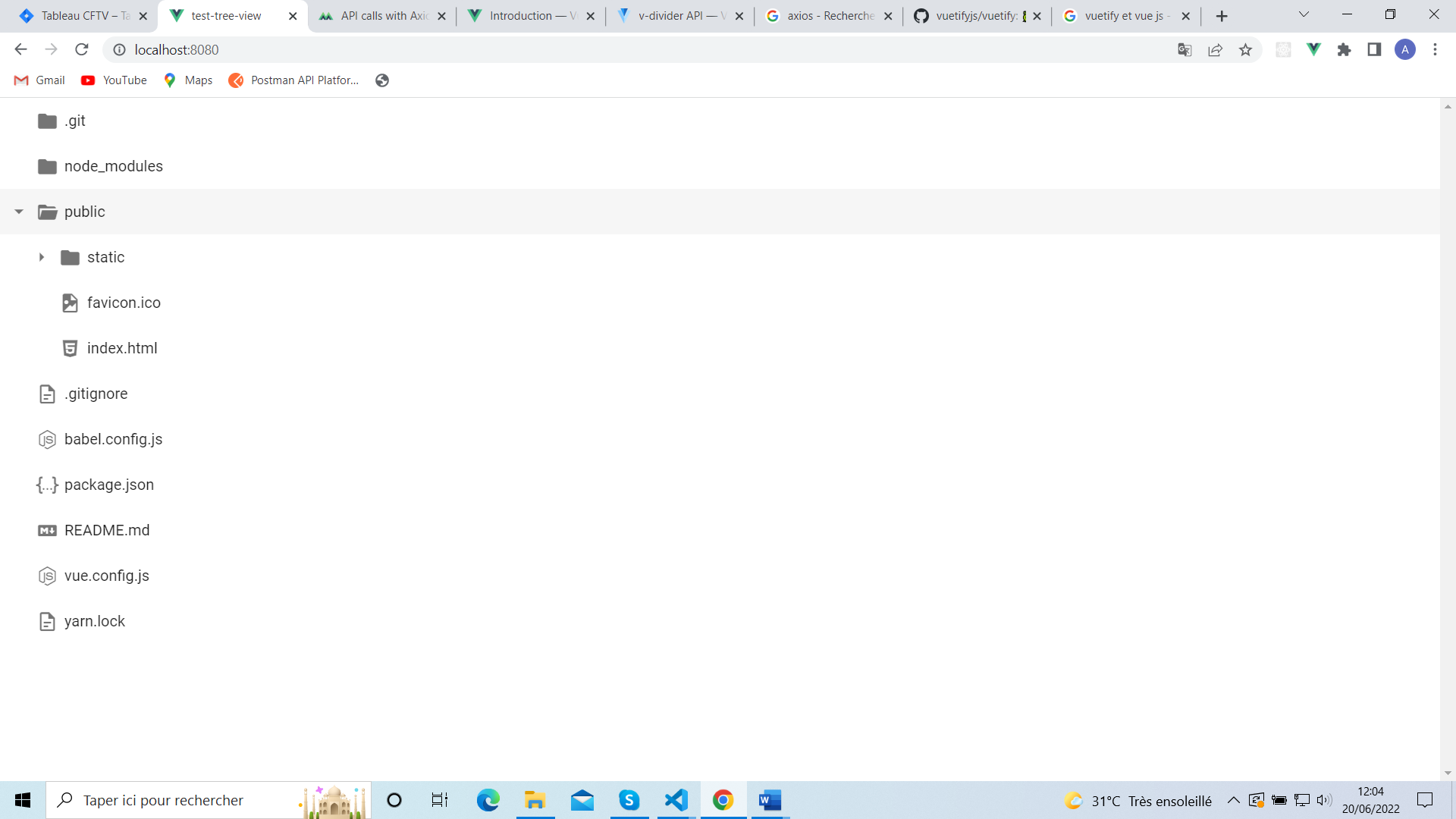
* Lancer l’exécution de l’application en mode développement

npm run serve

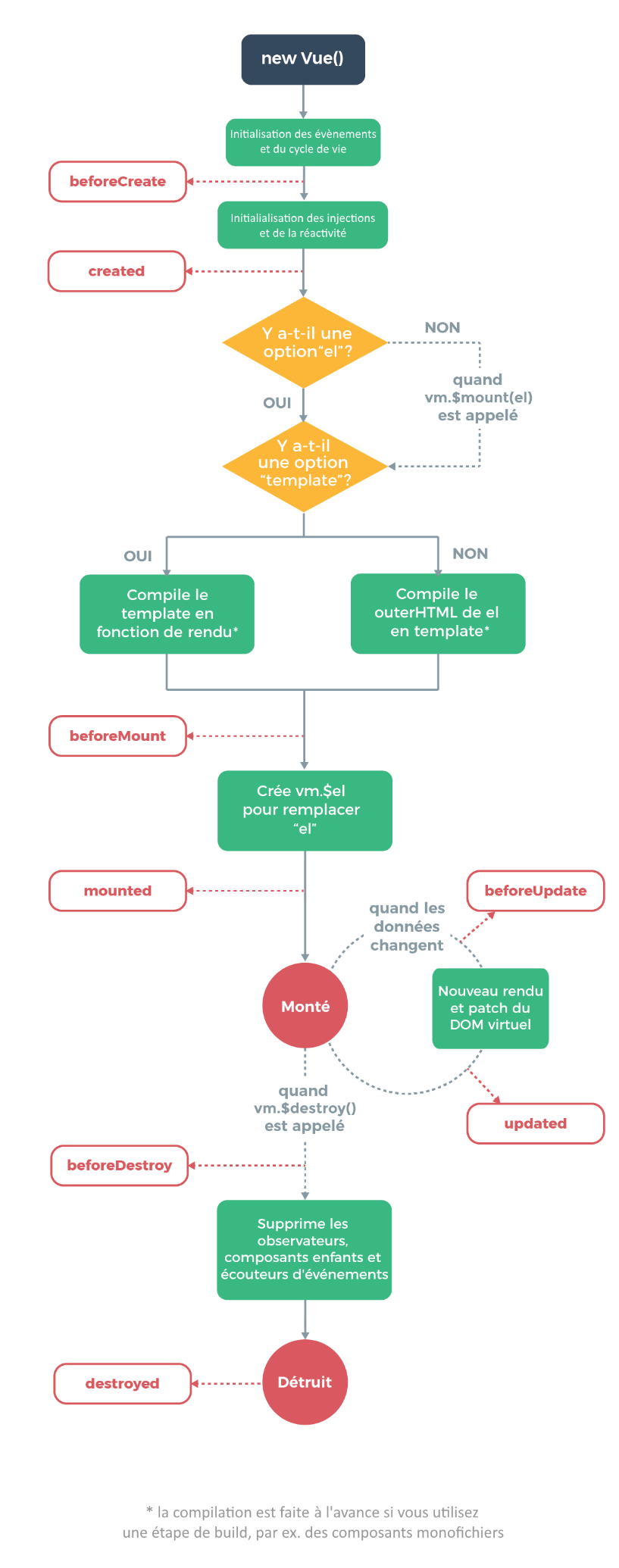
**IV – Annexes**

**Annexe 1 :** Représentation visuelle du TreeView seul

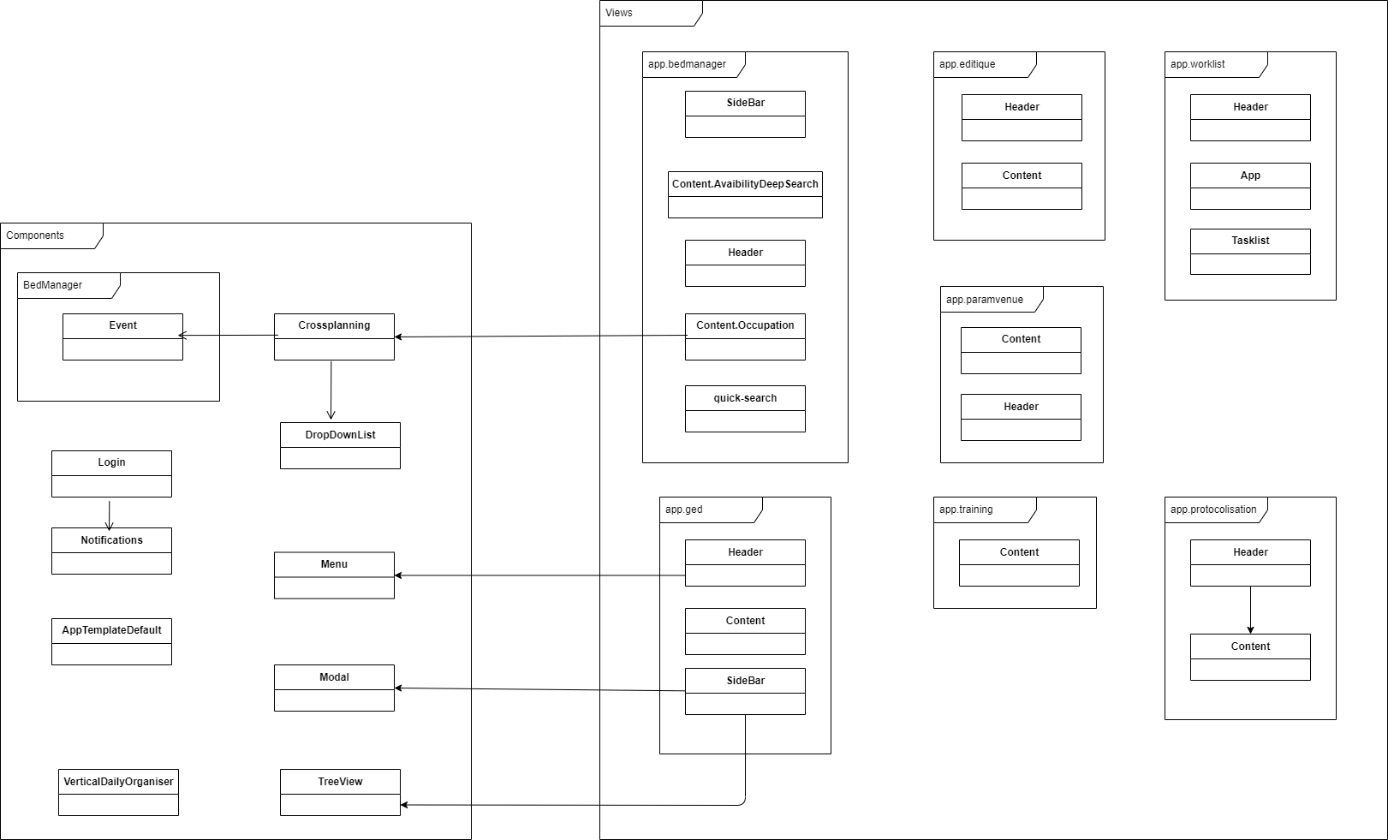




**Annexe 2 :** Schéma du cycle de vie d’un composant

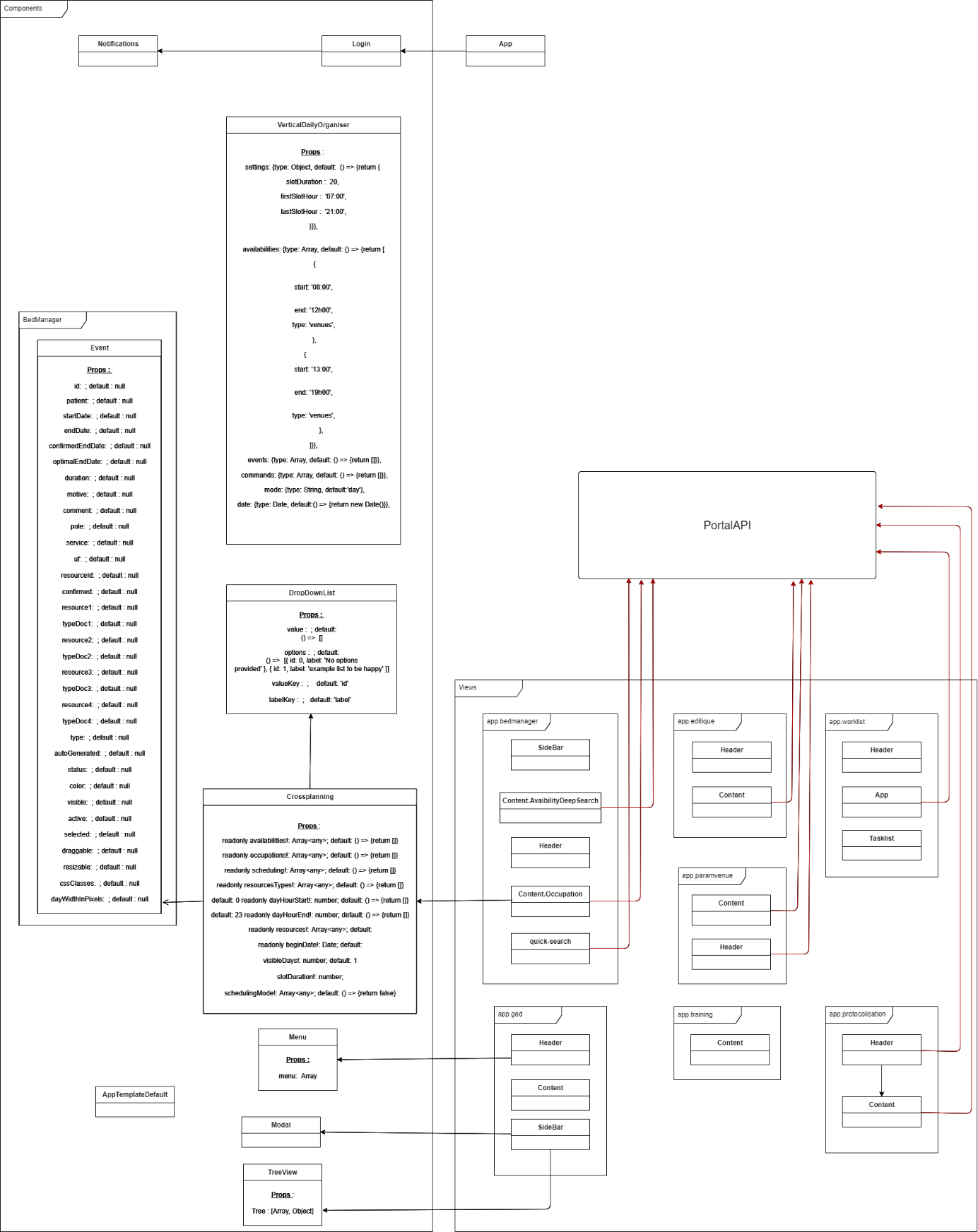


**Annexe 3 :** Structure des composants sans les props



*Source : /docs/composantsSansProps.png*

**Annexe 4 :** Structure des composants avec les props et les appels PortalAPI



*Source : /docs/composants.drawio*

*/docs/composants.png*

**Annexe 5 :** Exemple de format de données utilisées pour l’affichage du TreeView

{

"items": [

  {

    "name": ".git"

  },

  {

    "name": "node\_modules"

  },

  {

    "name": "public",

    "children": [

      {

        "name": "static",

        "children": [

          {

            "name": "logo.png",

            "file": "png"

          }

        ]

      },

      {

        "name": "index.html",

        "file": "html"

      }

    ]

  },

  {

    "name": ".gitignore",

    "file": "txt"

  },

  {

    "name": "babel.config.js",

    "file": "js"

  },

  {

    "name": "package.json",

    "file": "json"

  },

  {

    "name": "README.md",

    "file": "md"

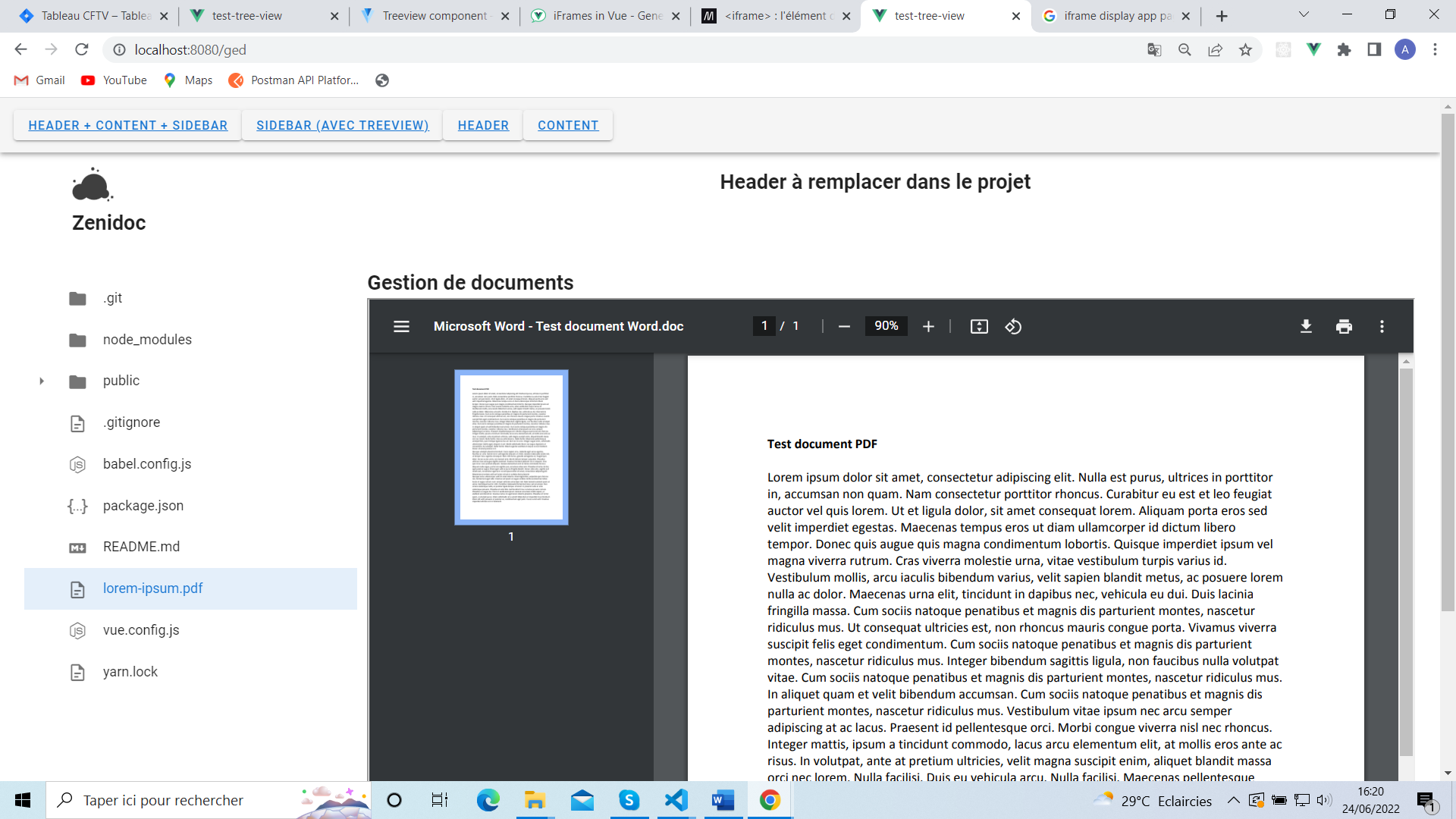
  },

]

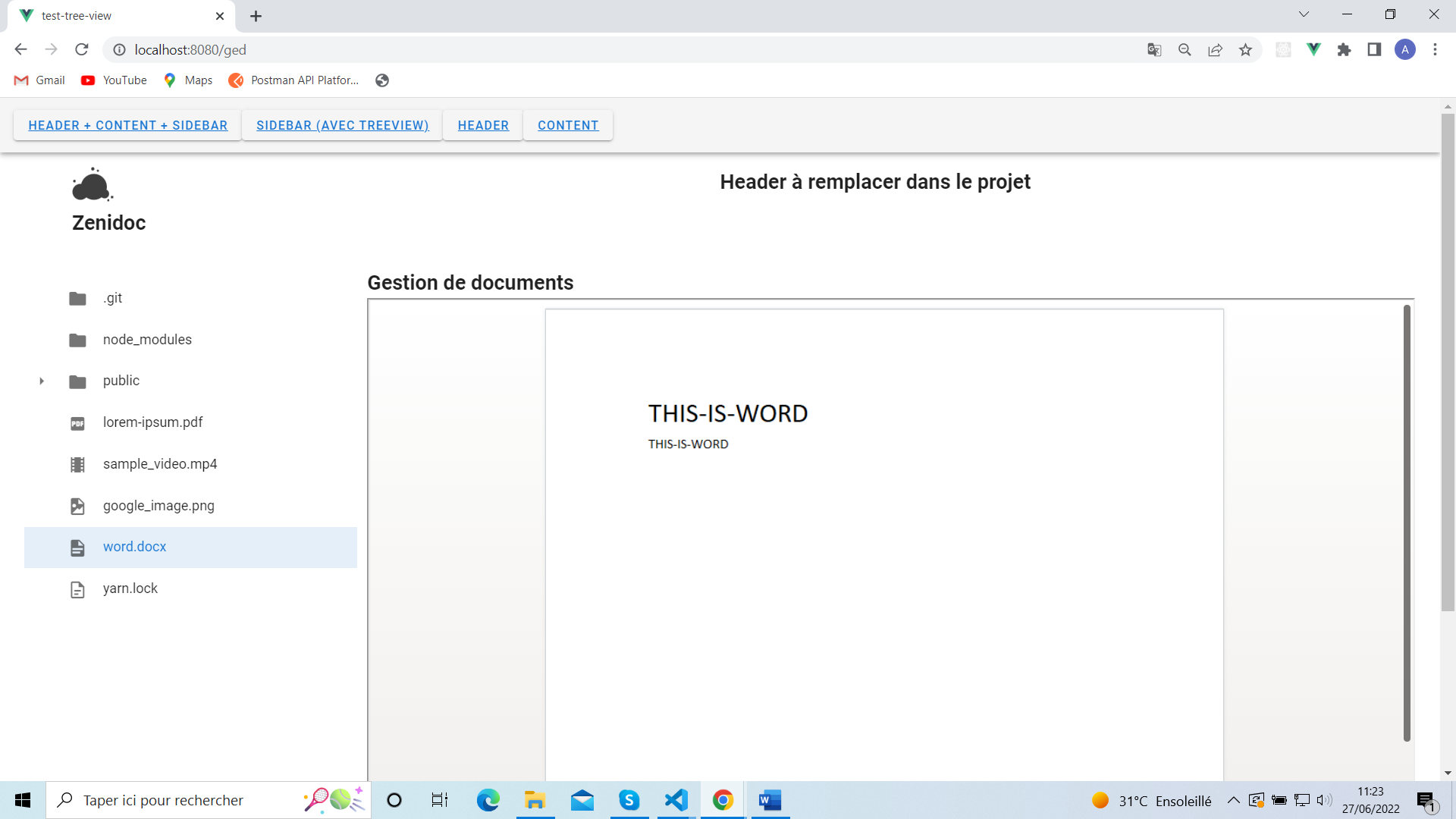
}

*Source : /db.js*

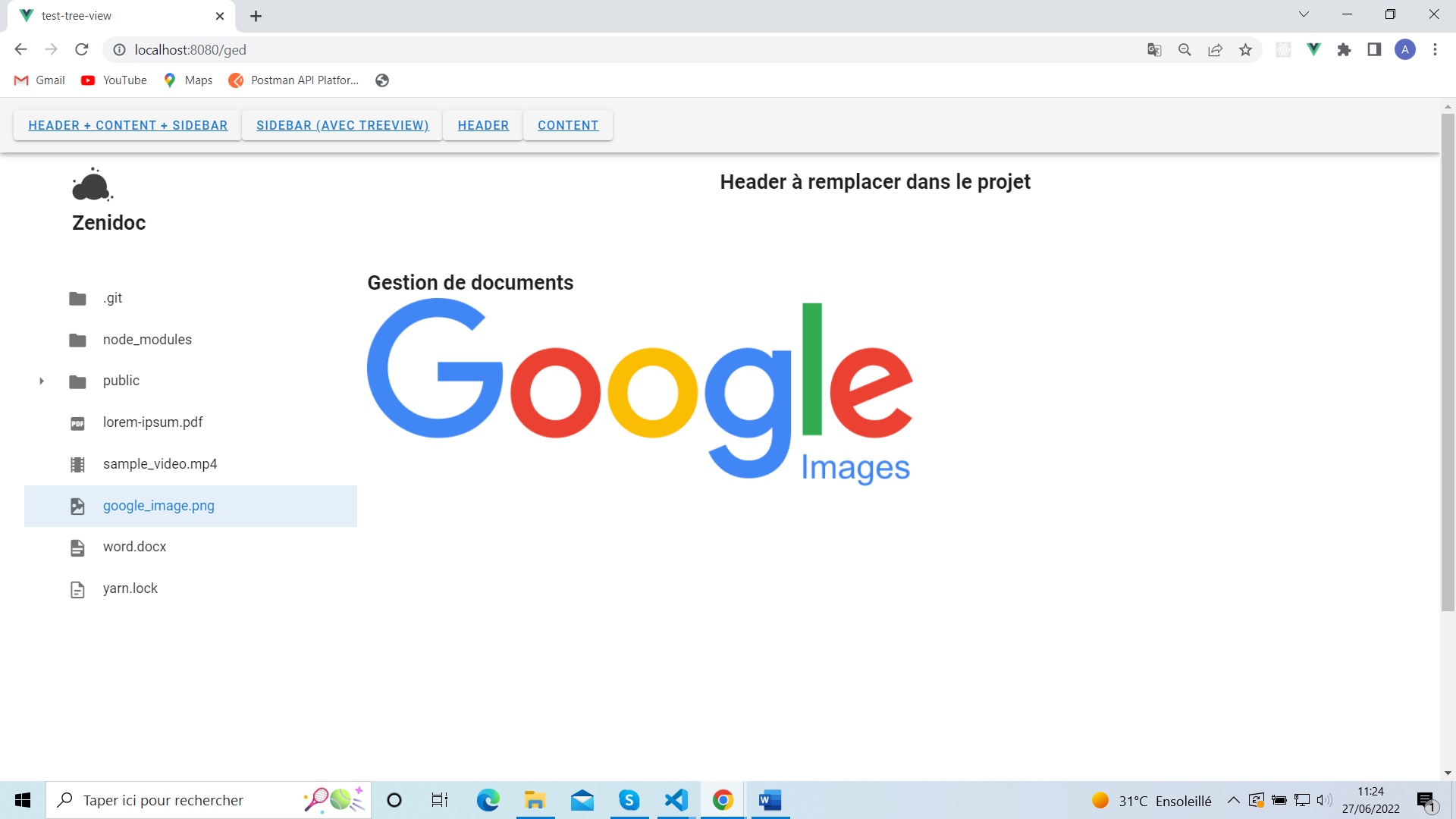
**Annexe 6 :** Exemples de divers types de fichiers affichés



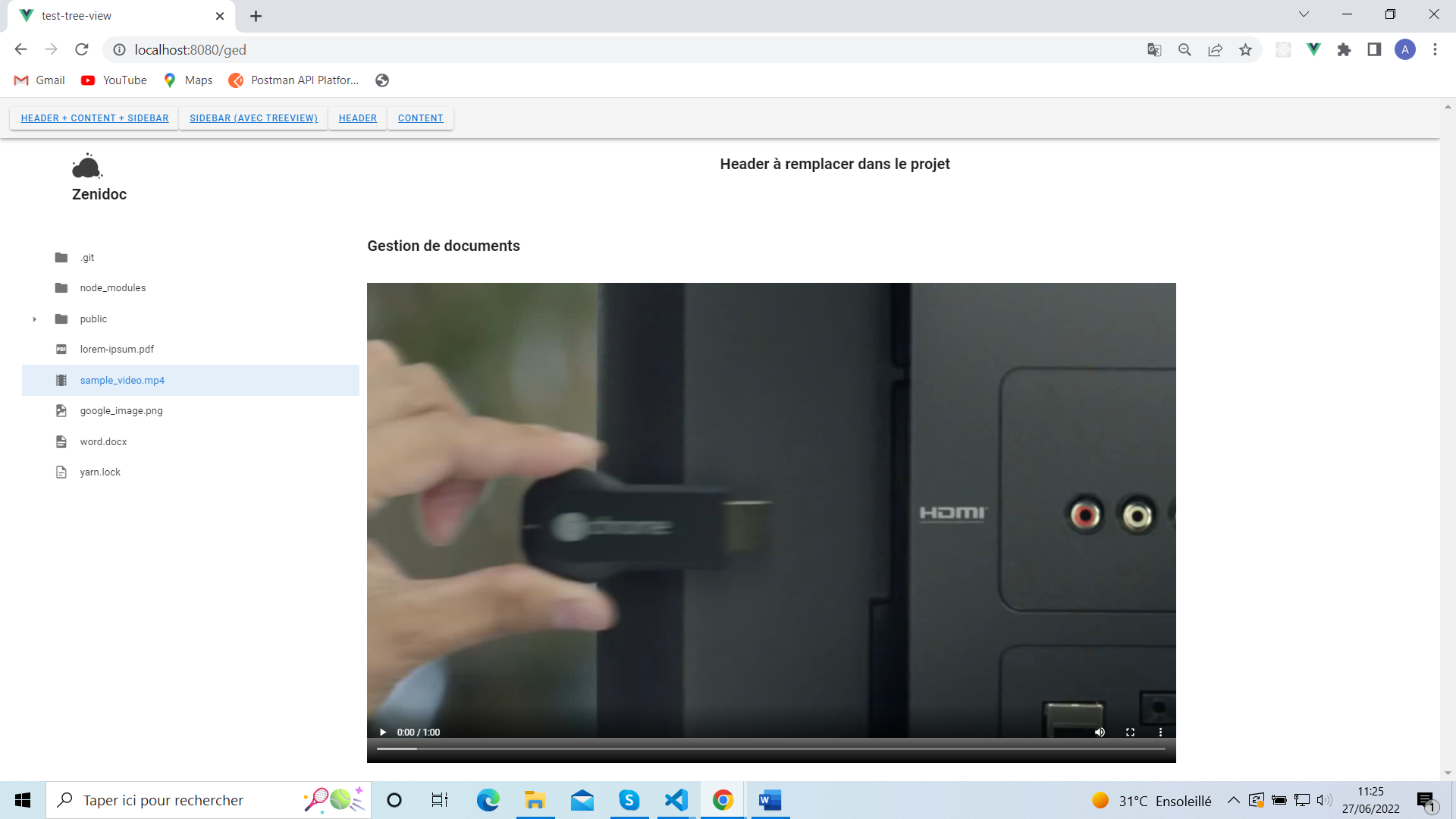
*Fichier .pdf*



*Fichier .docx*



*Fichier .png*



*Fichier .mp4*

**V – Bibliographie**

Documentation Vue.js 2

<https://v2.vuejs.org/v2/guide/>

<https://www.vuemastery.com/>

<https://openclassrooms.com/fr/courses/6390311-creez-une-application-web-avec-vue-js>

Documentation sur l’utilisation du composant TreeView et de Vuetify

<https://vuetifyjs.com/en/components/treeview/>

<https://vuetifyjs.com/en/api/v-treeview/>

<https://www.vuemastery.com/courses/beautify-with-vuetify/getting-started-with-vuetify>

Documentation sur l’utilisation d’Axios et de json-server

<https://axios-http.com/docs/intro>

<https://github.com/typicode/json-server>

<https://www.vuemastery.com/courses/real-world-vue-js/API-calls-with-Axios>

Différentes icônes pour la représentation des fichiers

<https://materialdesignicons.com/>

Documentation sur Vue Class Component

<https://class-component.vuejs.org/>

Documentation sur Vue-Router

<https://router.vuejs.org/guide/>

Afficher des documents de la suite Microsoft Office avec un iframe

<https://www.microsoft.com/en-us/microsoft-365/blog/2013/04/10/office-web-viewer-view-office-documents-in-a-browser/>