**VUE(CLI)**

**Dure totale du module : 21H00**

Table des matières

[Objectifs pédagogiques : 3](#_Toc109898136)

[VUE + Firebase (communiquer avec une BDD) 4](#_Toc109898137)

[setup de Firebase (console web) 4](#_Toc109898138)

[SetUp de l’application VUE 11](#_Toc109898139)

[Exercice : Setup de l’app 11](#_Toc109898140)

[Router 12](#_Toc109898141)

[Exercice : installation / configuration du router 13](#_Toc109898142)

[Exercice : Utilisation du router (ajout navbar) 14](#_Toc109898143)

[Installation / configuration de firebase 15](#_Toc109898144)

[Installer Firebase dans le projet 15](#_Toc109898145)

[Exercice Config de la Connexion à Firebase 16](#_Toc109898146)

[Exercice : Mise en place du Service (utilisation de class JS) via ajout en BDD 17](#_Toc109898147)

[Cycle de vie des composants (LifeCycle Hooks) 20](#_Toc109898148)

[Exercice : Liste des articles 22](#_Toc109898149)

[Exercice :Création du composant article 26](#_Toc109898150)

# Objectifs pédagogiques :

Être capable de comprendre les enjeux des Frameworks

Être capable d'initialiser et paramétrer un projet grâce à un Frameworks front-end

Être capable de réaliser des interface utilisateur grâce à un Frameworks front-end

Être capable de mettre en place un système de Routing (url)

Être capable de gérer des requêtes HTTP (BDD)

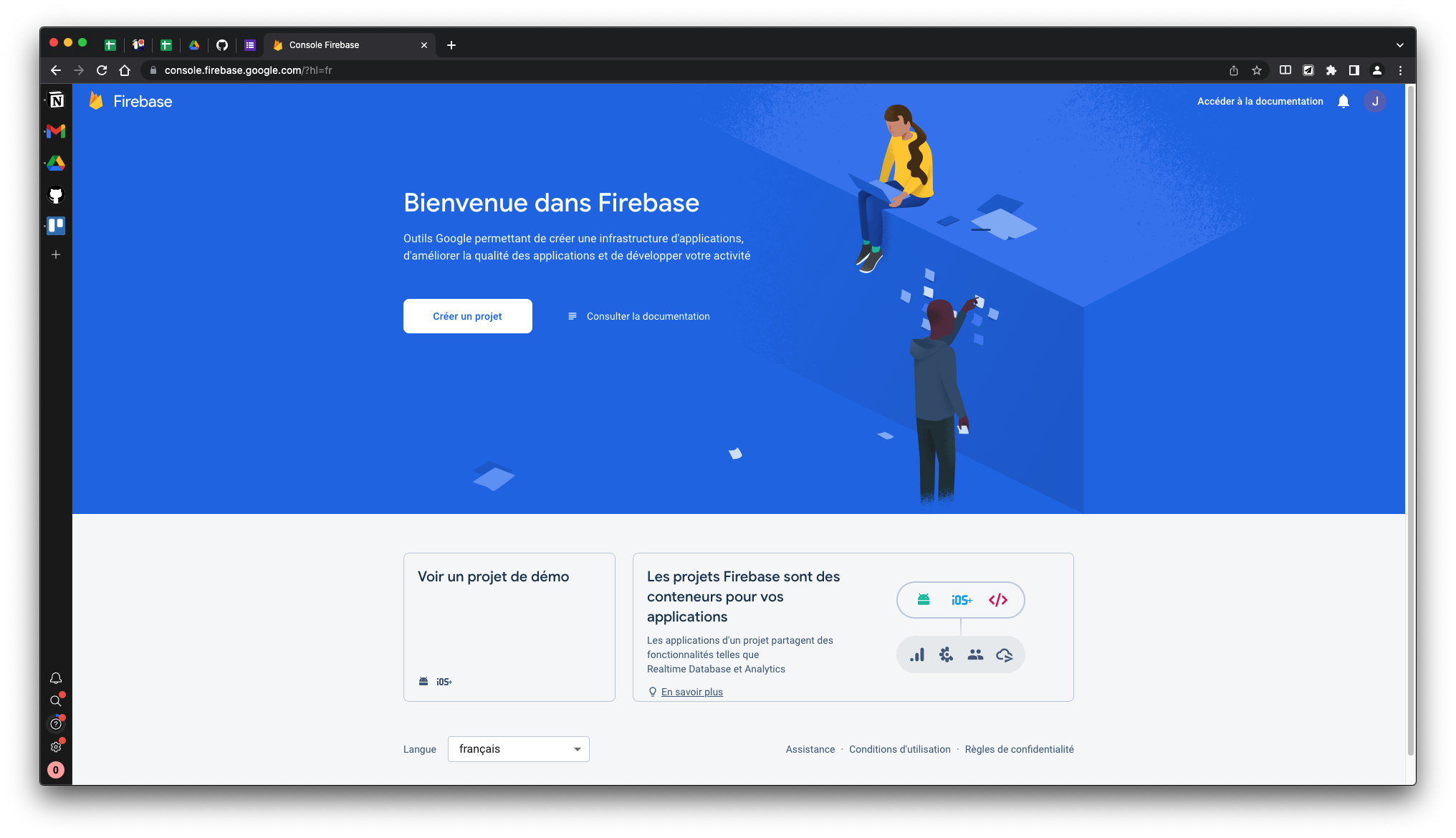
Être capable de connecter une application à une base de données

# VUE + Firebase (communiquer avec une BDD)

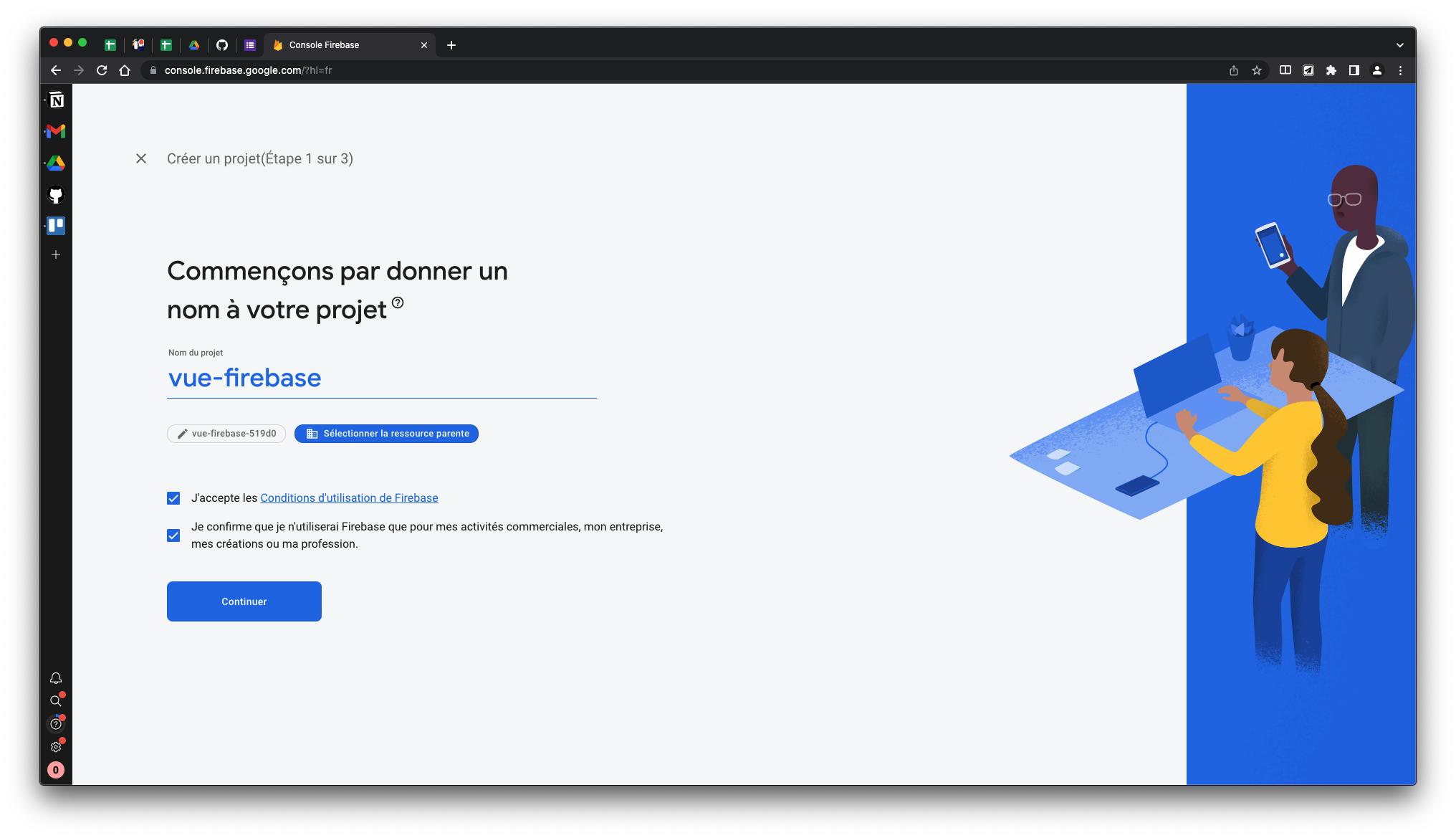
## setup de Firebase (console web)

Rdv à cette adresse <https://console.firebase.google.com/>, Se connecter

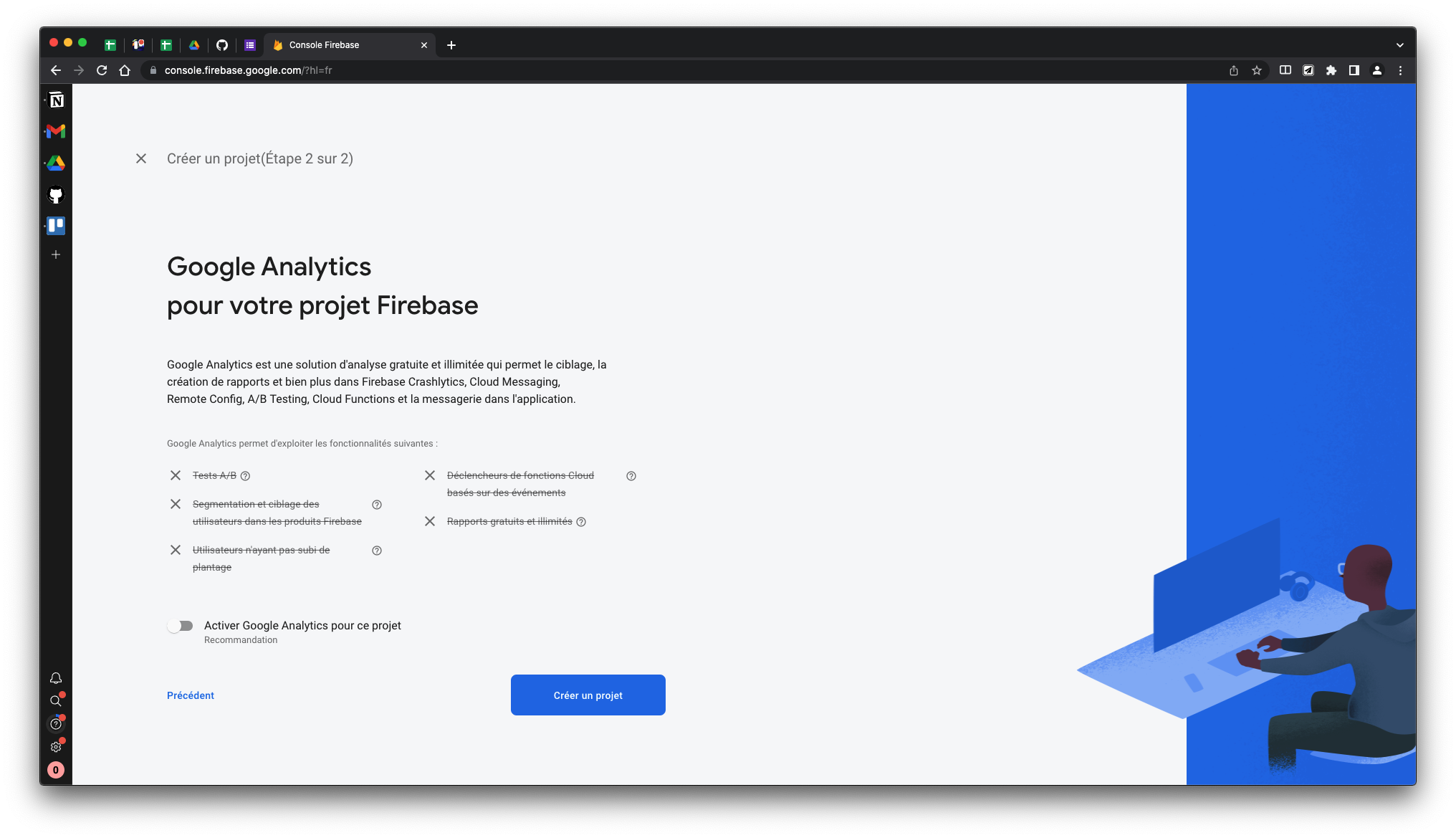
Une fois connecté :



On créer un projet, on donne un nom de projet



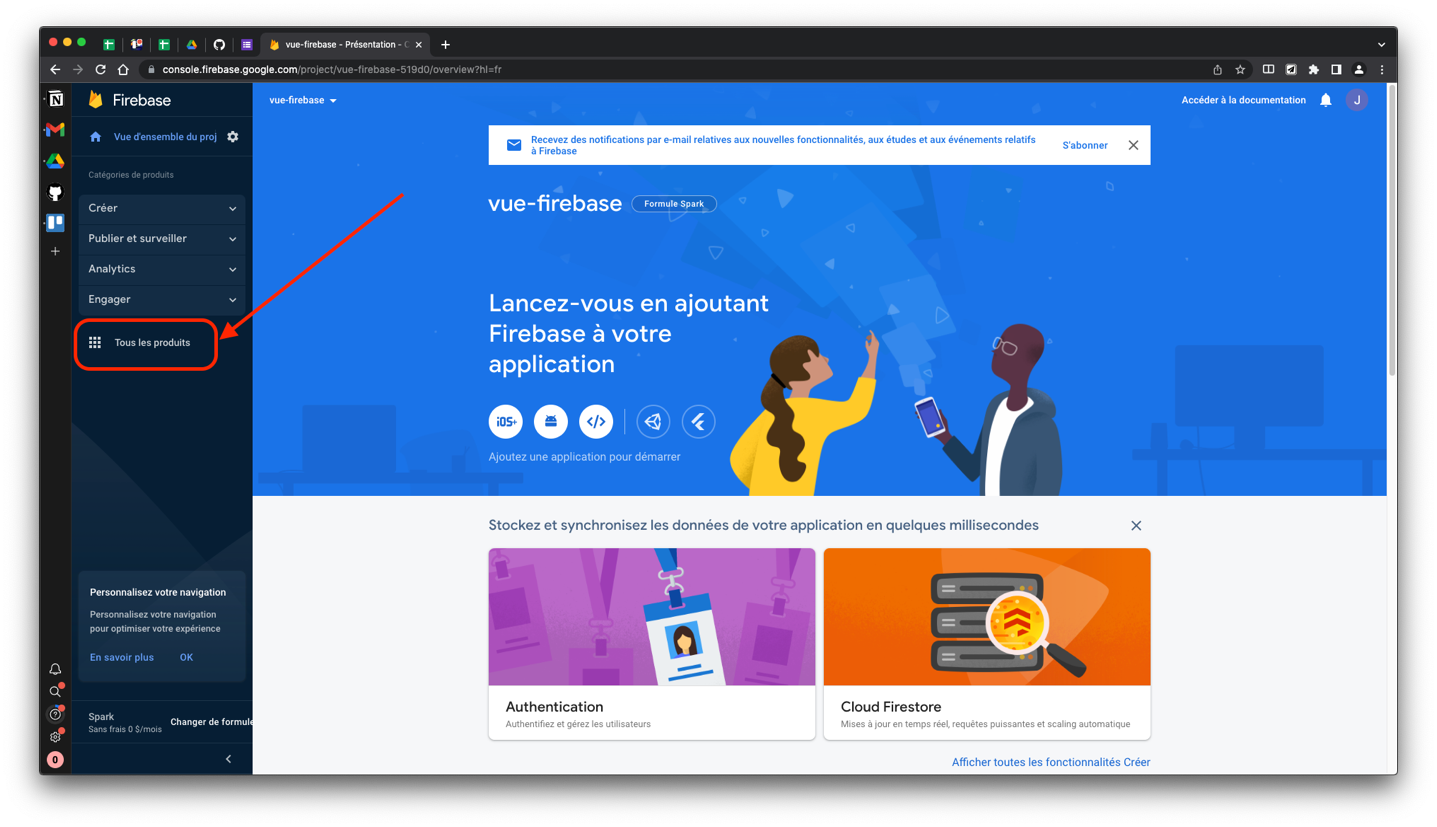
Pour notre app de test pas besoin de google analytics :

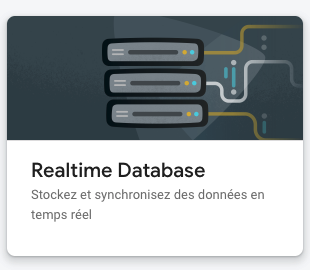


Le projet se crée :



On va dans Tous Les Produits pour sélectionner REALTIME DATABASE





Créer une REALTIME DATABASE



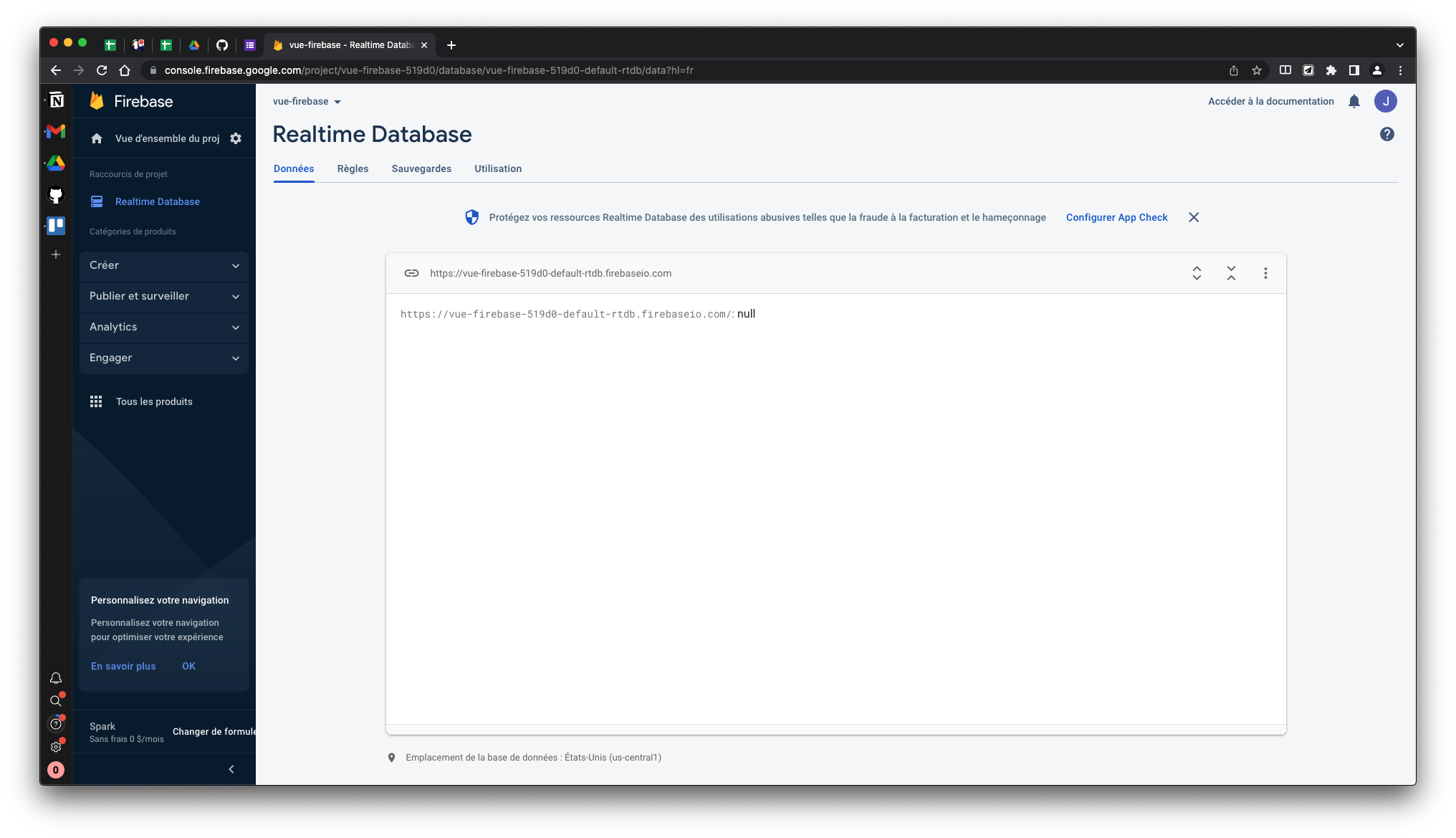
Emplacement des données (pas important pour notre test)



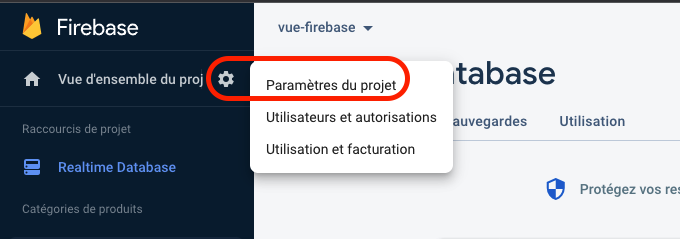
Mettre la RealTime Database en mode test :



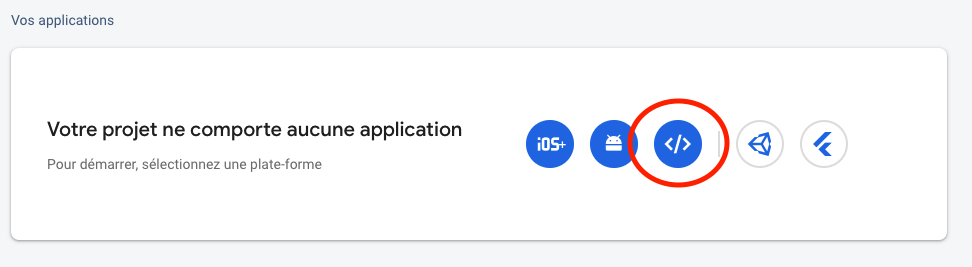
On arrive dans notre RealTime DataBase :



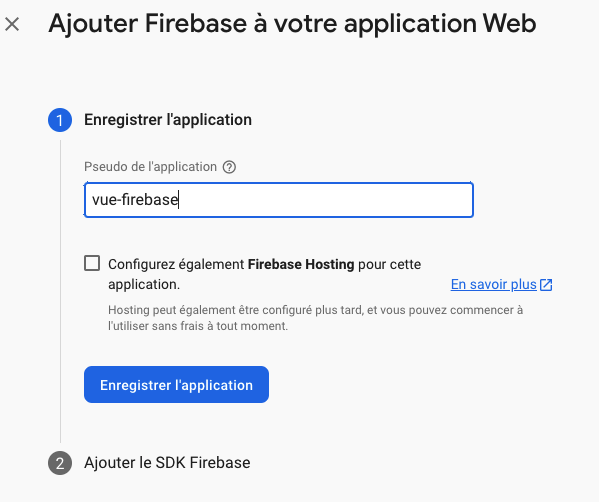
On va connecter notre BDD à notre application web :



En bas de la page Paramètre du projet :



Renseignez un nom d’application (pas besoin de firebase Hosting) :



On arrive sur un écran avec une const firebaseConfig) copier-coller tout le code (mettre de côté),   
on peut retourner dans la console.

## SetUp de l’application VUE

Nous allons réaliser une application VUE, qui va gérer des articles (ou users ou produits…), Nous mettrons en place un système de **routing** pour naviguer à travers plusieurs **composants** et nous allons mettre en place un système de **service** pour se **connecter** et **interagir** avec une **BDD**.

### Exercice : Setup de l’app

Via le CLI , créer une nouvelle application VUE

vue create vue-firebase

(Default ([Vue 3] babel, eslint).)

**Dans le dossier src :**   
° Créer un fichier router.js et un fichier firebase.js  
° Créer un dossier services  
**Dans le dossier services :**   
°créer un fichier ArticleDataService.js  
**Dans le dossier components :**   
° Créer des fichiers pour les composants : AddArticle.vue, Article.vue, ArticlesList.vue

Via la commande npm install dans le terminal, installer Bootstrap (ou équivalent, à voir).

**Dans main.js :**   
° Rajouter l’import de bootstrap

## Router

Nous créons les routes de notre application en passant par ce tableau routes = []

Chaque route sera un objet dans le tableau qui va avoir des propriétés :   
- Path : l’URL   
- Name (optionnel)   
- Component : le composant à charger quand on arrive sur cette URL

La f° createWebHistory permet a vue de créer un historique de navigation au sein de l’application

### Exercice : installation / configuration du router

**Dans le terminal :**   
° Installer le router de vue dans le projet (npm install vue-router@4)

**Dans main.js :**   
°Rajouter l’import du router  import **router** from './router'  
°Utiliser la fonction use en lui passant router

createApp(App).use(router).mount('#app')

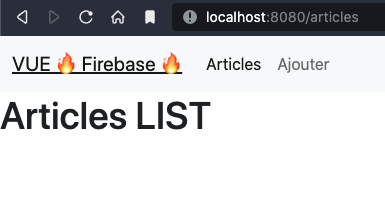
**Dans router.js :**  
° Importer les ƒ° createWebHistory et createRouter from “vue-router“    
import { createWebHistory, createRouter } from "vue-router";  
° Créer une const **routes** à laquelle on assigne un **tableau**   
° Dans ce tableau rajouter un objet JS { } dans lequel on a une propriété   
 path à laquelle on assigne “/“,   
 alias à laquelle on assigne “/articles“,   
 name à laquelle on assigne “articles“,   
 component à laquelle on assigne : () => import("./components/ArticlesList")  
°Dans ce tableau **routes** rajouter un 2nd objet JS { } dans lequel on a une propriété   
 path à laquelle on assigne “/add“,   
 name à laquelle on assigne “add“,   
 component à laquelle on assigne : () => import("./components/AddArticle")  
° Après le tableau des **routes**: créer une const router à laquelle on assigne la ƒ° createRouter()  
° la ƒ° createRouter(), prend en paramètre un objet JS { }  
° Dans cet objet JS, on a :   
 une propriété history à laquelle on assigne la ƒ° createWebHistory()  
 une propriété routes

À la fin on fait un export default de router

### Exercice : Utilisation du router (ajout navbar)

**Dans App.vue :**   
° Garder juste la balise <template> et la <div class=’app’>  
°Dans la <div class=’app’>, rajouter une navbar avec 3 liens (home , articles, ajouter)  
° Entourer un lien <a> avec une balise <router-link></router-link>  
Sur <router-link> rjouter un attribut **to="/"**° Sur le 2è lien <a> remplacer la balise <a> par <router-link></router-link>  
Sur <router-link> rjouter un attribut **to="/articles"**° Sur le 3è lien <a> remplacer la balise <a> par <router-link></router-link>  
Sur <router-link> rjouter un attribut **to="/add"  
° En dessous de la <nav> rajouter une balise <router-view />**(aller mettre du template dans les composants ArticlesList et AddArticle)





[GitHub](https://github.com/jefff404/cours-vue-cli/tree/21-vue-firebase-setup-router)

## Installation / configuration de firebase

Techniquement notre BackEnd C’est Firebase, là-dedans se trouve notre Base de données   
(RealTime Database), que l’on installe Firebase ou non, on va dans tous les cas contacter une API de manière asynchrone.

En JS il y a plusieurs façons de faire des fonctions asynchrones principalement **async await** et **then().catch()**

Quand on installe Firebase dans notre projet via la commande npm (ou autre), on installe juste la bibliothèque JS de firebase, ce qui donne accès aux fonctions de firebase.

Pour interagir avec le BDD nous utiliserons des fonctions en mode asynchrone comme ceci :

deleteArticle() {

ArticleDataService.delete().then(Du code).catch(Du code);

},

### Installer Firebase dans le projet

npm install [firebase@8.10.0](mailto:firebase@8.10.0)   
(Si jamais la V9 pose problème)

### Exercice Config de la Connexion à Firebase

**Dans firebase.js :**   
° Rajouter les import de firebase :   
import firebase from 'firebase/app';  
import  "firebase/database";  
°Rajouter la const firebaseConfig (la variable de config dispo dans le site firebase)  
° Sur firebase, utiliser la ƒ° initializeApp() qui prend en paramètre firebaseConfig  
° Enfin on fait un export default de firebase, sur laquelle on utilise la ƒ° database()

**Dans ArticleDataService.js :**   
° import firebase from ‘../firebase';  
°Créer une const db à laquelle on assigne firebase sur laquelle on utilise la ƒ° .ref().   
° ref() prend en paramètre “/articles“  
° Créer une classe ArticleDataService { } (vide pour le moment)  
° Ensuite on fait un export default d’une nouvelle instance de cette classe ArticleDataService

[GitHub](https://github.com/jefff404/cours-vue-cli/tree/22-vue-firebase-setup-firebase)

### Exercice : Mise en place du Service (utilisation de class JS) via ajout en BDD

**Dans ArticleDataService.js** (**dans la CLASS**):   
° Créer une ƒ° getAll() dans laquelle on retourne db  
° Créer une ƒ° create() qui prend article en paramètre et dans laquelle on retourne db sur laquelle on utilise la ƒ° push(article)  
° Créer une ƒ° update() qui prend key et value en paramètre et dans laquelle on retourne db sur laquelle on utilise la ƒ° child(key) et la ƒ° update(value)  
° Créer une ƒ° delete() qui prend key en paramètre et dans laquelle on retourne db sur laquelle on utilise la ƒ° child(key) et la ƒ° remove()  
°Créer une ƒ° deleteAll() dans laquelle on retourne db sur laquelle on utilise la ƒ° remove()

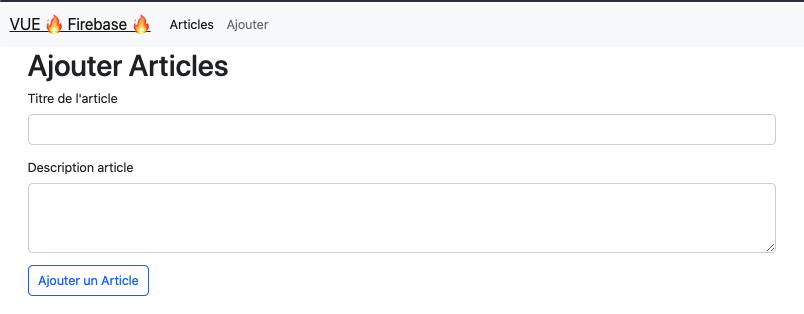
**Dans AddArticle.vue dans la partie <template> :**° mettre en place un formulaire avec 2 inputs (1 pour le **title**, 1 pour la **description**) et un bouton **Dans AddArticle.vue dans la partie <script> :**   
°Importer ArticleDataService  
**°Dans l’export default :**   
° Une propriété **name** à laquelle on assigne « add-article »  
°Dans **data(){}**, on **retourne** dans un objet : une variable **submitted** initialisée à false et un objet **article** {}  
° Dans l’objet **article** on a des propriétés :   
 **title** et description initialisé avec ‘’ (string vide quoi)  
 **published** initialisée à false

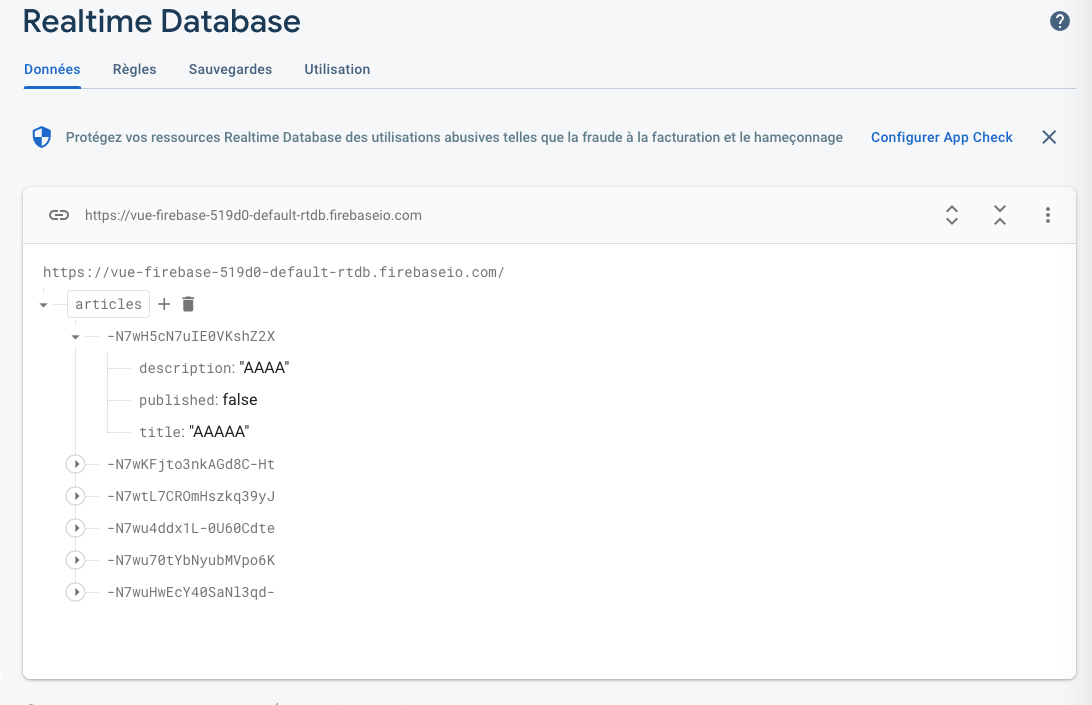
°Dans **methods**:   
° Rajouter une ƒ° saveArticle () dans laquelle on déclare une variable **data** qui est un objet (vide pour le moment)  
° Dans cet objet **data** :  
 On assigne à **title** le this.article.title  
 On assigne à **description** le this.article.description  
 On assigne à **published**, false  
°**Après cet objet , toujours dans la ƒ° saveArticle:**   
°Sur **ArticleDataService** utiliser la ƒ° create() qui prend data en paramètre **data**chainé à create() utiliser la ƒ° .then()  
chainé à then() utiliser la ƒ° .catch() ca nous donne cà pour le moment **ArticleDataService.create(data).then().catch();**  
°Dans les paramètres de **then()** faire une fonction fléchée anonyme   
()=>{}   
dans cette fonction fléchée on va :   
assigner true à **submitted** (c’est dans les data de vue)  
faire un console log qui dit ‘article crée’  
°Dans les paramètres de **catch()**, faire une fonction fléchée anonyme avec une variable error en paramètre   
 **error**=>{}  
Dans cette dernière fonction fléchée: on fait un console log de **error**

**Toujours dans methods (toujours dans AddArticle.vue) :**   
°Rajouter une methods newArticle () dans laquelle :  
 ° on passe la variable submitted à false  
 ° On assigne à article un objet dans lequel :   
 ° On assigne à title :“ “  
 ° On assigne à description :“ “  
 ° On assigne à published : false

**Dans AddArticle.vue dans la partie <template> :**

°Sur le bouton ajouter placer un v-on qui réagit au click et fait référence à la ƒ° saveArticle  
° sur le un input place un v-model pour le lier avec article.title  
° sur le un input place un v-model pour le lier avec article.description





[GitHub](https://github.com/jefff404/cours-vue-cli/tree/23-vue-firebase-service)

### Cycle de vie des composants (LifeCycle Hooks)

Dans l’exercice suivant nous allons déclencher des fonctions À un certain moment du cycle de vie d’un composant :

En arrière fond VUE gère avec précision la vie d’un composant en effet lorsque nous arrivons sur une page qui appelle un composant celui-ci passe pas plusieurs états et VUE met à disposition des fonction pour exécuter n’importe quel code JS à certains moment de la vie du composant :   
Si vous vous souvenez dans chaque application VUE, dans main.js on utilise une fonction mount(), pour monter le composant sur une div du template (pour que vue prenne le contrôle de cette <div>)  
Dans l’exercice ci-après nous allons utiliser la fonction mount() pour signifier à notre composant   
(**AU MOMENT** où celui-ci est **MONTÉ** dans l’application) de surveiller la BDD (avec la fonction .on() pour firebase).  
Ensuite nous allons utiliser la fonction **beforeUnmount()** pour signifier à notre composant (Juste **AVANT LE MOMENT** ou celui-ci est **DÉMONTÉ** (donc juste avant qu’il ne soit plus afficher dans la page)) de ne plus surveiller notre BDD avec la fonction .off() de firebase  
Sur la page suivante vous trouverez un schéma récapitulatif des états du cycle de vie d’un composant  
Les fonction de VUE sont assez explicites :

beforeCreate : avant que le composant ne soit crée

created : Quand le composant est crée

beforeMount :Avant qu’il soit monté (placé) dans l’application

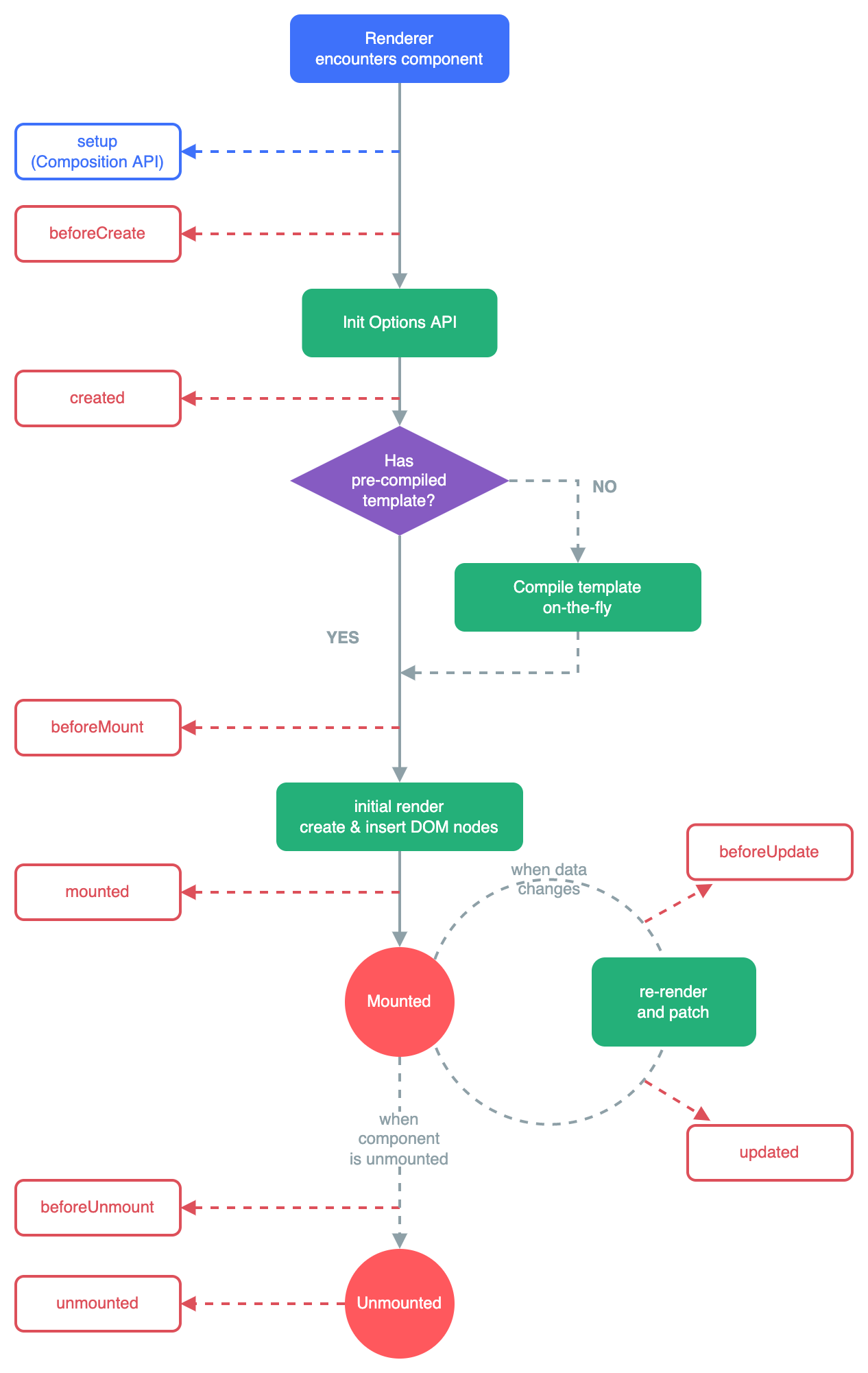
mounted : Quand il vient d’être monté (placé) dans l’application

beforeUpdate : Avant que le composant se mettre à jour

update : Quand le composant se met à jour

beforeUnmount : Avant que le composant ne soit (démonté / retiré) de l’application

unmount : Quand le composant est démonté (retiré) de l’application



### Exercice : Liste des articles

**Dans ArticlesList.vue <script> :**   
°Importer ArticleDataService  
°Importer ArticleDetails from ./Article.vue (on peut importer un composant sous un autre nom)  
**Dans l’export default :**   
°Une propriété name : “articles-list“  
°Une propriété component : { ArticleDetails}  
°Dans data() { } :   
 articles, un tableau vide  
 currentArticle à null  
 currentIndex à -1  
  
**Dans ArticlesList.vue <script>(methods)**  
°Créer une methods **onDataChange(),** qui prend en paramètre items  
°Dans cette ƒ° :   
 Créer une variable \_articles, un tableau vide   
 (le \_ devant c’est pour la différencier de articles que l’on a dans data)  
 Sur items, utiliser la ƒ° forEach(), qui prend en param une ƒ°=> qui elle a en param item  
 Dans la ƒ°=> :   
 ° Créer une variable key, on lui assigne item.key  
 ° Créer une variable data, on lui assigne item.val()  
 ° Sur \_articles, utiliser la ƒ° push() qui contient un objet  
 °Dans cet objet : à   
 key on lui assigne key,   
 À title on assigne data.title  
 À description on assigne data.description  
 À published on assigne data.published  
°En dehors du forEach (mais tjrs dans onDataChange), à this.articles, on assigne \_articles

**Dans ArticlesList.vue <script>(methods) :**

* + 1. °Créer une methods **refreshList(),**dans laquelle :   
        ° On assigne à currentArticle, null  
        ° On assigne à currentIndex, -1
  1. °Créer une methods **setActiveArticle(),**qui prend en param article et index, Dans cette ƒ°:   
      °On assigne à currentArticle, article  
      °On assigne à currentIndex, index

**Dans ArticlesList.vue <script>(methods)**

* + 1. °Créer une methods **removeAllArticles**(),dans laquelle :   
        °Sur ArticleDataService on utilise la ƒ° deleteAll()  
        °Chainé à deteleAll(), utiliser la ƒ° then() puis catch()  
        on a donc ceci : ArticleDataService.deleteAll().then().catch()  
         
        °Dans les paramètres de then() on a une fonction fléchée  
        ° Dans cette ƒ° =>, on execute la ƒ° refreshList  
        °Dans les paramètres de catch() on a une fonction fléchée qui a en paramètre e  
        °Dans cette ƒ°=>, on console.log(e)

**Dans ArticlesList.vue <script>(en. Dehors des methods) :   
°**Créer une ƒ**° mounted(),**dans laquelle :   
 °Sur ArticleDataService on utilise la ƒ° getAll()  
 ° Chainé à getAll(), utiliser la ƒ° on() comme ceci : ArticleDataService.getAll().on();  
° la fonction on() prend en 1er param “value“ et en 2nd param, this.onDataChange  
°Créer une ƒ° **beforeUnmount()**,dans laquelle :   
 °Sur ArticleDataService on utilise la ƒ° getAll()  
 ° Chainé à getAll(), utiliser la ƒ° off() comme ceci : ArticleDataService.getAll().off();  
° la fonction off() prend en 1er param “value“ et en 2nd param, this.onDataChange

* 1. **ArticlesList.vue <template> :**

<template>

<div class="container list row">

<h1>Articles LIST</h1>

<div class="col-md-6">

<ul class="list-group">

<li></li>

</ul>

<button class="m-3 btn btn-sm btn-danger">Tout Suppr.</button>

</div>

<div class="col-md-6">

<div>

<article-details></article-details>

</div>

<div>

<p>Veuillez selectionner un article.</p>

</div>

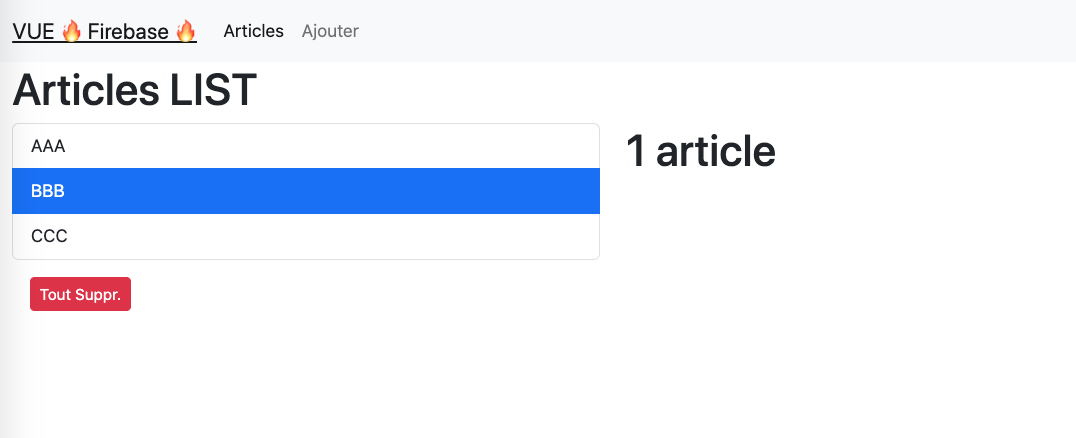
</div>

</div>

</template>

* 1. Dans **ArticlesList.vue <template> :**

°Sur la <li> :   
 ° un v-for qui parcourt le tableau **articles** avec une variable unArticle et l’index  
 ° un v-bind de key pour le lier à l’index du v-for  
 ° un v-on qui réagit au click et exécute la methods setActiveArticle avec en paramètre unArticle et index  
°Dans la <li> : avec l’interpolation afficher le title de unArticle  
°Sur le <button> Tout Suppr : réagir au click pour appeler la methods removeAllArticles  
°Sur la <div> (juste au-dessus de <article-details>) : un v-if qui vérifie si currentArticle est définie.  
°Sur la <div> (juste au-dessus u <p>) : un v-else  
°Sur la balise <article-details> :   
 °un v-bind de article pour le lier à currentArticle  
 °(pour plus tard)réagir à l’evènement refreshList pour appeler la **methods** refreshList

****

[GitHub](https://github.com/jefff404/cours-vue-cli/tree/23-vue-firebase-service-articles-list)

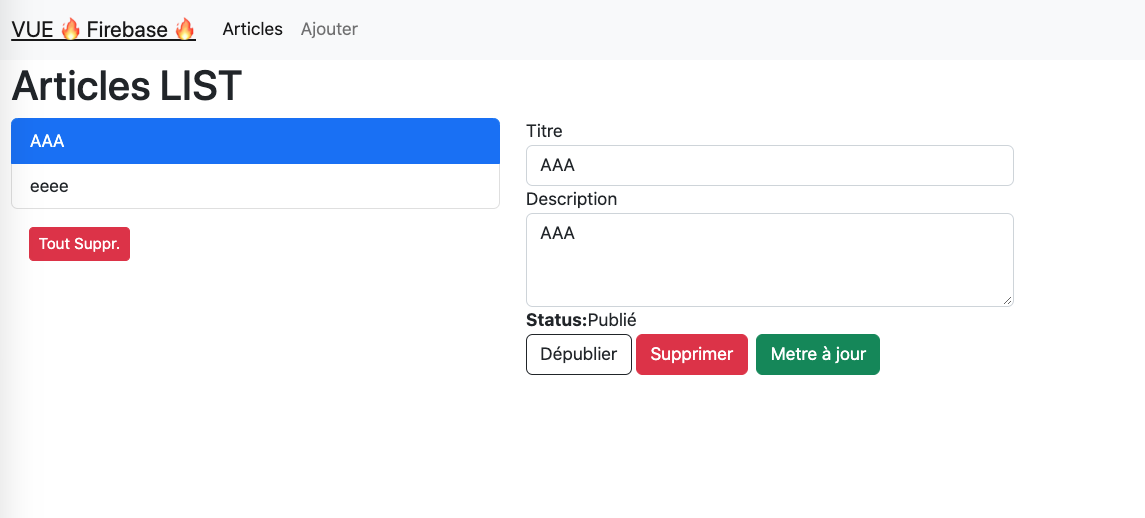
### Exercice :Création du composant article

**Dans Article.vue <script> :**   
°Importer ArticleDataService  
°Dans l’export default :   
 Une propriété name : “article“ ou « un-article » (à voir si on a des warning en console)  
 Une propriété props : [ “article“]

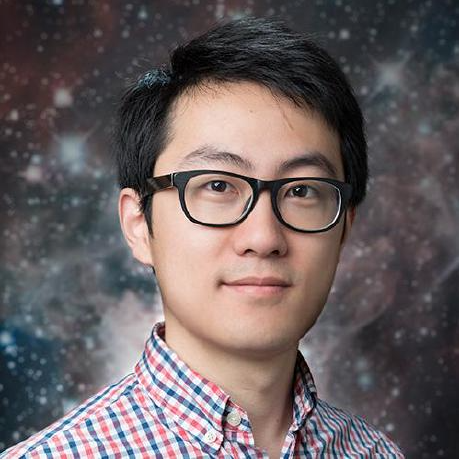
**Dans data() { } :**   
 °currentArticle à null  
 °message à “ “

**Dans watch { } :**   
 article à laquelle on assigne une ƒ° qui prend en param article, dans cette ƒ°:  
 On assigne à this.currentArticle, { ...article }  
 On assigne this.message, “ “

**Dans methods: { } :**  
°Créer une methods **updatePublished()** qui prend en param status,   
 °Dans cette ƒ° :   
 °Sur ArticleDataService utiliser la ƒ° update() qui prend   
 °en 1er param this.currentArticle.key   
 °en second param un objet {} dans lequel   
 °à published, on assigne status  
 °Chainé à update(), utiliser la ƒ° then() qui prend en param une fonction fléchée, dans la fonction :  
 °À this.currentArticle.published assigner status  
 °À this.message assigner « Le status à bien été MAJ »  
 °Chainé à then(), utiliser la ƒ° catch() qui prend en param une fonction fléchée qui a en param e   
 ° on console.log(e)

* + - * 1. **Dans methods: { } :**  
           °Créer une methods **updateArticle()** ,Dans cette ƒ° :   
            °Créer une const data, un objet, dans lequel :   
            °À title on assigne this.currentArticle.title  
            °À description on assigne this.currentArticle.description  
           Après l’objet data :  
           °Sur ArticleDataService utiliser la ƒ° update() qui prend   
            °en 1er param this.currentArticle.key   
            °en second param data   
            °à published, on assigne status  
            °Chainé à update(), utiliser la ƒ° then() qui prend en param une fonction fléchée, dans la fonction   
            °À this.message assigner « L’article a bien été MAJ »  
            °Chainé à then(), utiliser la ƒ° catch() qui prend en param une fonction fléchée qui a en param e   
            ° on console.log(e)  
             
           **Dans methods: { }**   
           °Créer une methods **deleteArticle**() ,Dans cette fonction :   
            °Sur ArticleDataService utiliser la ƒonction delete() qui prend en param this.currentArticle.key  
            °Chainé à delete(), utiliser la ƒ° then() qui prend en param une ƒonction fléchée dans laquelle :   
            °Avec la ƒ° $emit, on EMET l’event “refreshList“  
            °Chainé à then(), utiliser la ƒ° catch() qui prend en param une fonction fléchée qui a en param e   
            ° on console.log(e)
        2. **En dehors de methods{} :**°Créer une ƒ° **mounted()** dans laquelle,   
            °À this.message on assigne “ “  
            °À this.currentArticle on assigne { ...this.article }
        3. Dans Article.vue <template> :
* <template>
* <div class="edit-form">
* <form>
* <div class="form-group">
* <label for="title">Titre</label>
* <input type="text" class="form-control" id="title"/>
* </div>
* <div class="form-group">
* <label for="description">Description</label>
* <textarea class="form-control" rows="3"></textarea>
* </div>
* <div class="form-group">
* <label><strong>Status:</strong></label>
* </div>
* </form>
* <button class="btn btn-outline-dark">Dépublier</button>
* <button class="btn btn-info mx-1">Publier</button>
* <button class="btn btn-danger mx-1" >Supprimer</button>
* <button type="submit" class="btn btn-success mx-1">Metre à jour</button>
* <p>{{ message }}</p>
* </div>
* <div>
* <p>Cliquez sur un article Por Favor</p>
* </div>
* </template>
  + - * 1. **Dans Article.vue <template> :**
        2. °Sur la 1ère <div> un v-if vérifie si currentArticle est définie  
           °Sur l’input de title, placer un v-model relier à currentArticle.title  
           °Sur textarea de description, placer un v-model relier à currentArticle.description  
           °Dans le <label> avec une condition ternaire sur currentArticle.published on affiche ‘publié’ ou   
           ‘en attente’  
           °sur le bouton Dépublier  
            ° un v-if vérifie si currentArticle.published est définie (true ou false)  
            ° on réagit au click pour exécuter la ƒ° updatePublished en passant false en paramètre  
           °sur le bouton Publier  
            ° un v-else  
            ° on réagit au click pour exécuter la ƒ° updatePublished en passant true en paramètre  
           °sur le bouton Supprimer  
            ° on réagit au click pour appeler la ƒ° deleteArticle   
           °sur le bouton Mettre à jour  
            ° on réagit au click pour appeler la ƒ° updateArticle  
           °Dans le <p> on affiche la data message  
           °Sur la dernière <div> placer un v-else (rappel 1ère div)
        3. ****

[GitHub](https://github.com/jefff404/cours-vue-cli/tree/24-vue-firebase-article)

****