API Web avec Plumber

Marion DANYACH - Imen DERROUICHE - Olfa LAMTI

20/12/2020

I - Introduction

Plumber est un package R open source qui convertit le code R en API REST. Plumber permet d'exploiter le code R à partir d'autres plates-formes ou langages de programmation. Il peut être utilisé pour intégrer un graphique créé dans R dans un site Web ou pour exécuter une prévision sur certaines données à partir d'une application Java.

Plumber va donc nous permettre de communiquer de manière dynamique les analyses et travaux qui ont été fait avec du code R.

Dans ce document, nous vous expliquerons dans un premier tmeps qu'est ce qu'une API Web puis nous vous présenterons les bases de Plumber et nous expliquerons la création d'une API Web à partir de zéro.

II - Qu'est ce qu'une API Web ?

1 - API

API est l'acronyme d'Application Programming Interface), c'est un moyen d'interagir par programmation avec un composant logiciel ou une ressource distincte.

Une API répertorie un ensemble d'opérations que les développeurs peuvent utiliser, ainsi qu'une description de ce qu'ils font. Le développeur n'a pas nécessairement besoin de savoir comment, par exemple, un système d'exploitation se construit et présente une boîte de dialogue «Enregistrer sous». Ils ont juste besoin de savoir qu'il est disponible pour une utilisation dans leur application.

Les API permettent aux développeurs de gagner du temps en tirant parti de la mise en œuvre d'une plate-forme pour faire le travail de fond. Cela permet de réduire la quantité de code que les développeurs doivent créer et de créer plus de cohérence entre les applications pour la même plate-forme. Les API peuvent contrôler l'accès aux ressources matérielles et logicielles.

2 - API Web

L'API Web, comme son nom l'indique, est une API sur le Web accessible via le protocole HTTP. C'est un concept et non une technologie. Nous pouvons créer une API Web en utilisant différentes technologies telles que Java, .NET, etc.

3 - API REST

REST est un ensemble de principes directeurs auxquels un développeur doit adhérer avant de pouvoir considérer son API Web comme « RESTful ». Voici les 6 principes q'une API REST doit adéherer :

Architecture client-serveur: Les clients de l'API utilisent des appels HTTP pour demander une ressource (une méthode GET) ou envoyer des données au serveur (une méthode POST), ou l'une des autres méthodes HTTP prises en charge par l'API. GET et POST sont les méthodes les plus fréquemment utilisées, mais d'autres méthodes comme HEAD, PUT, PATCH, DELETE, CONNECT, OPTIONS ET TRACE peuvent également être prises en charge.

Sans état: Une application sans état ne maintient pas de connexion ni ne stocke d'informations entre deux requêtes du même client. Un client fait une requête, l'API exécute l'action définie dans la requête et répond. Une fois que l'API a répondu, elle se déconnecte et ne conserve aucune information sur le client dans sa mémoire active. L'API traite chaque requête comme une première demande.

Avec mise en cache: Une API REST doit normalement permettre la mise en cache des données fréquemment demandées. Pour réduire la bande passante, la latence et la charge du serveur, une API doit pouvoir identifier les ressources pouvant être mises en cache, déterminer qui peut les mettre en cache et décider pendant combien de temps elles peuvent rester dans le cache.

Interface uniforme: Le client interagit avec le serveur selon une manière définie, indépendamment de l'appareil ou de l'application.

Système en couches: Une API peut avoir plusieurs couches, telles que des serveurs proxy ou des dispositifs de répartition de charge, et le serveur d'extrémité peut déployer des serveurs supplémentaires pour formuler une réponse. Le client ne sait pas quel serveur répond à la requête. Un système en couches rend une API plus évolutive.

Code sur demande (facultatif): L'API peut envoyer du code exécutable tel que des applets Java ou JavaScript.

Nous allons donc voir comment à l'aide du package Plumber nous pouvons créer des API REST pour pouvoir communiquer notre travail de manière dynamique et utiliser le code R pour qu'il soit disponible pour une utilisation dans une application.

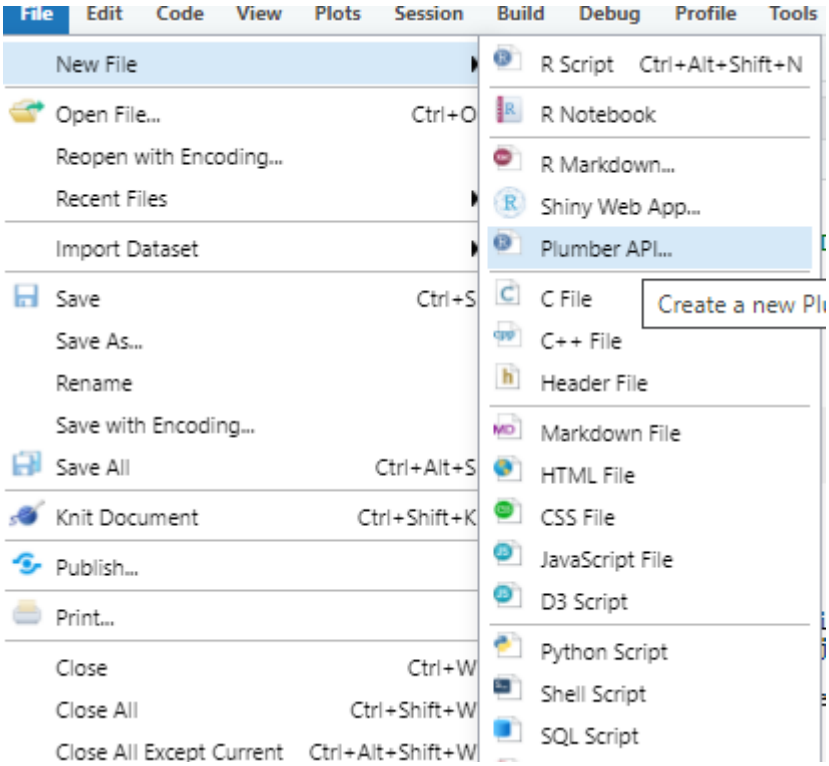
II - Le package Plumber

Plumber est une manière simple de convertir son code R en API Web. A partir de simples commentaires à ajouter dans son code nous pouvons créer des API. Le code à ajouter est puissant car c'est uniquement des commentaires à ajouter.

Etudions maintenant les différents type de commentaires.

Avant de créer le fichier Pumber API, veillein staller dans la console le âckages plotly en écrivant dans la console : 'install.packages(plumber)'

En créant un nouveau fichier de type Plumber API, comme ci-dessous, vous trouverez un code que nous allons étudier.



Voici le code affiché une fois que nous ouvrons un nouveau fichier de type PLumber API :

```
#
# This is a Plumber API. You can run the API by clicking
# the 'Run API' button above.
#
# Find out more about building APIs with Plumber here:
#
#    https://www.rplumber.io/
#

library(plumber)

#* @apiTitle Plumber Example API

#* Echo back the input
#* @param msg The message to echo
#* @get /echo
function(msg = "") {
    list(msg = paste0("The message is: '", msg, "'"))
}

#* Plot a histogram
#* @png
#* @get /plot
function() {
    rand <- rnorm(100)
    hist(rand)
}
```

```
#* Return the sum of two numbers
#* @param a The first number to add
#* @param b The second number to add
#* @post /sum
function(a, b) {
    as.numeric(a) + as.numeric(b)
}
```

Etudions maintenant les paramètres un par un:

```
library(plumber)
```

Cette fonctionalité va nous permettre de charger le package plumber.

Plumber avec la méthode GET et le chemin echo

```
#* @apiTitle Plumber Example API

#* Echo back the input
#* @param msg The message to echo
#* @get /echo
function(msg = "") {
    list(msg = paste0("The message is: '", msg, "'"))
}
```

“#” : Pour Plumber ce sont des annotations Swagger, cela permet de décrire la structure de nos API afin que les machines puissent les lire. Ces annotations vont permettre de de faire des descriptions de l’API, de décrire les paramètres, . . .

"#*" : Pour ajouter des commentaires

“#* @apiTitle Plumber Example API” : Indique le nom de l’API

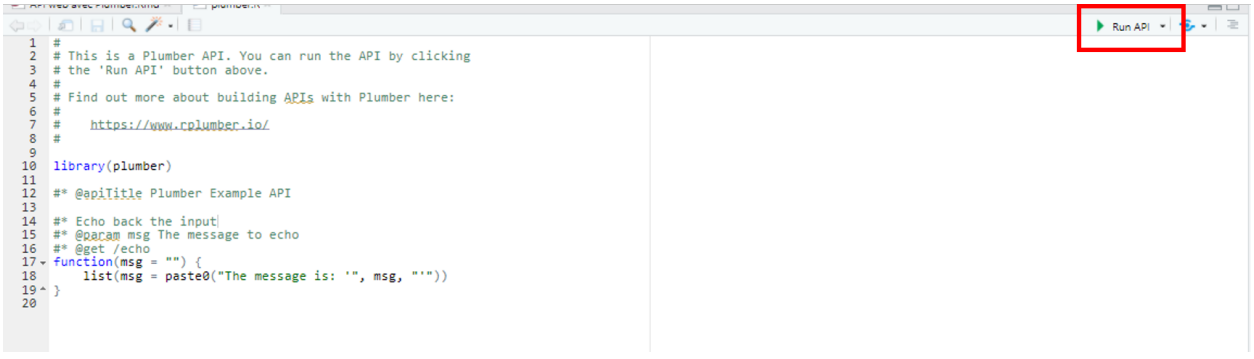
“#* Echo back the input” : Permet d’avoir un commentaire de la fonction

“#* @param msg The message to echo” : Décrit le les paramétres d’une fonction

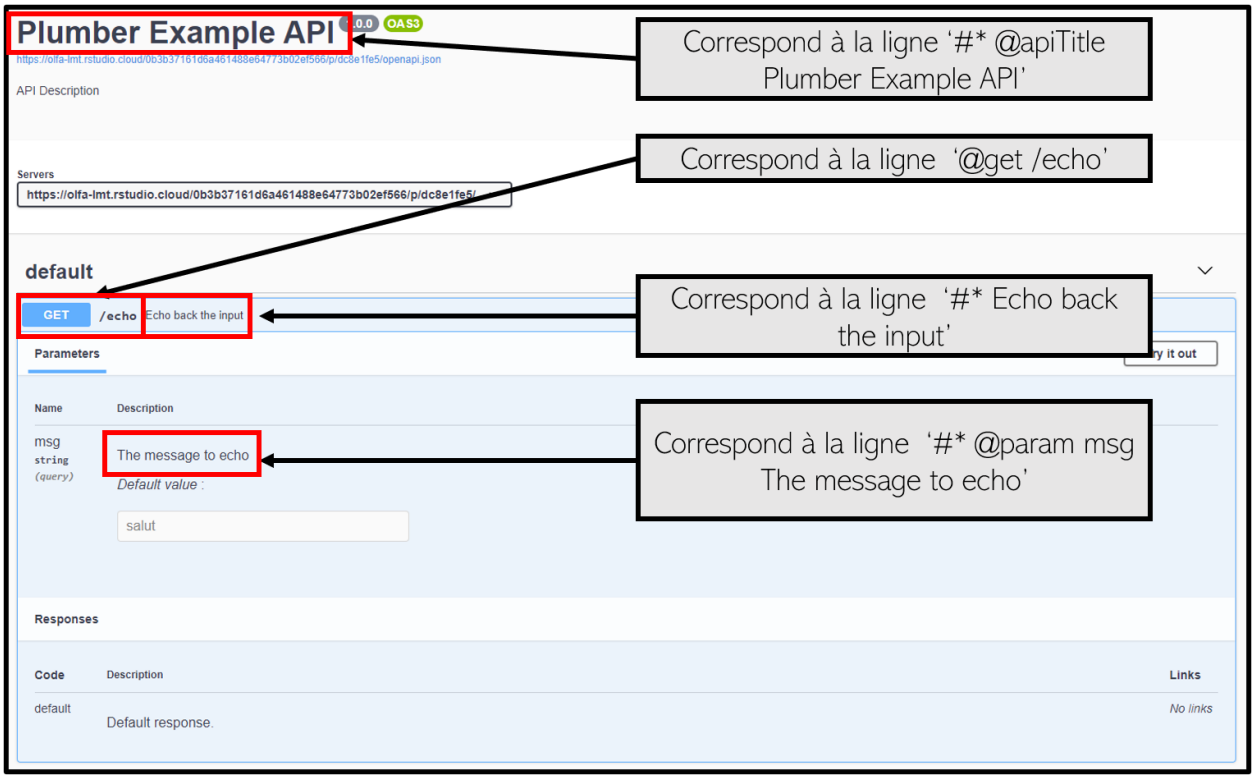
“#* @get /echo” : Indique la méthode HTTP utilisée, ici la méthode utilisée est GET. Indique égelement le chemin utilisé, ici echo.

“function(msg =””) { . . . ” : C’est le code en R

Lançons maintenant l’API comme ci-dessous, en cliquant sur “Run API”, encadré en rouge:

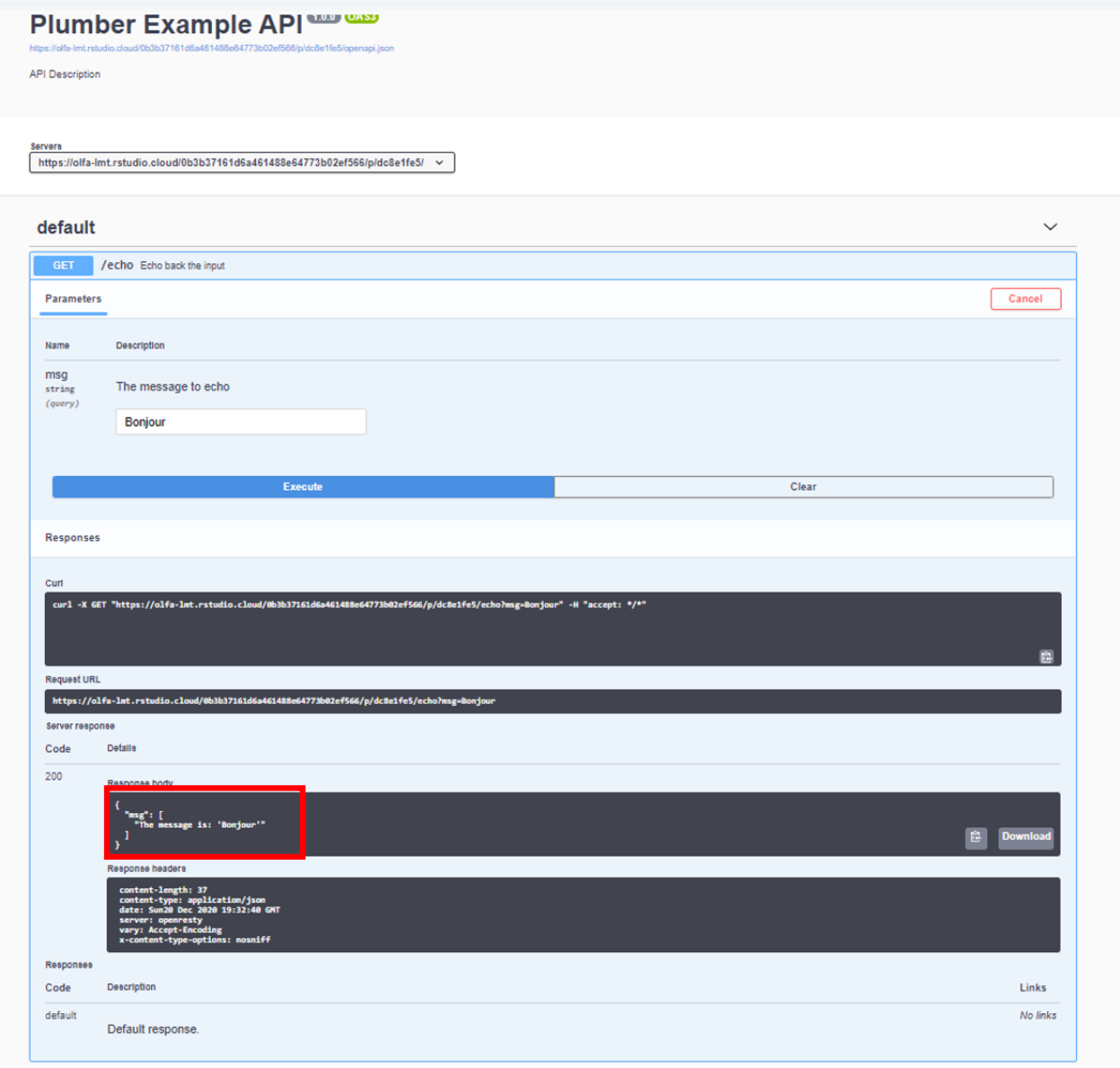


Voici le résultat :



On peut voir ci-dessus que les Swagger que nous avons indiqué se sont intégrés à la fonction automatiquement.

Dans l'API, en cliquant sur "Try it out", nous pouvons utiliser la fonction msg. Vous pouvez le voir ci-dessous en ajoutant la message "Bonjour" et cliquant sur Execute.



L'encadré en rouge retourne la réponse de la fonction msg.

Plumber avec la méthode GET et le chemin plot

```
#* Plot a histogram
#* @png
#* @get /plot
function() {
    rand <- rnorm(100)
    hist(rand)
}
```

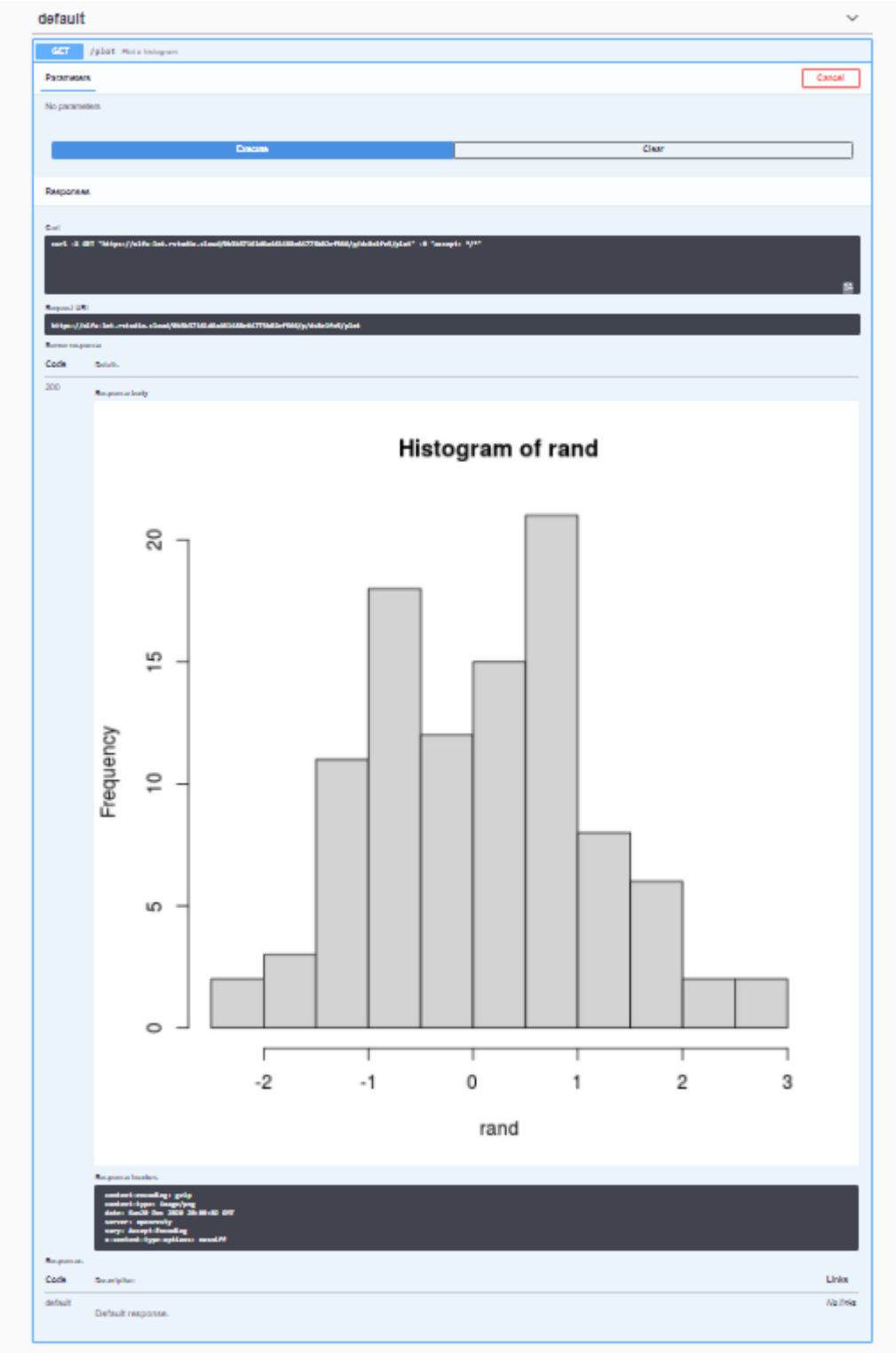
"#* Plot a histogram" : Permet d'avoir un commentaire de la fonction

"#* @png" : ???????????????

“#* @get /plot” : Indique la méthode HTTP utilisée, ici la méthode utilisée est GET. Indique égelement le chemin utilisé, ici plot.

“function() { . . . ” : C’est le code en R

Voici le résultat :



On peut voir que dans l'API, une fois que l'on execute la fonction, le graphique est affiché.

On constate également qu'il n'y a pas de description des paramètres.

Plumber avec la méthode POST et le chemin sum

```
#* Return the sum of two numbers
#* @param a The first number to add
#* @param b The second number to add
#* @post /sum
function(a, b) {
    as.numeric(a) + as.numeric(b)
}
```

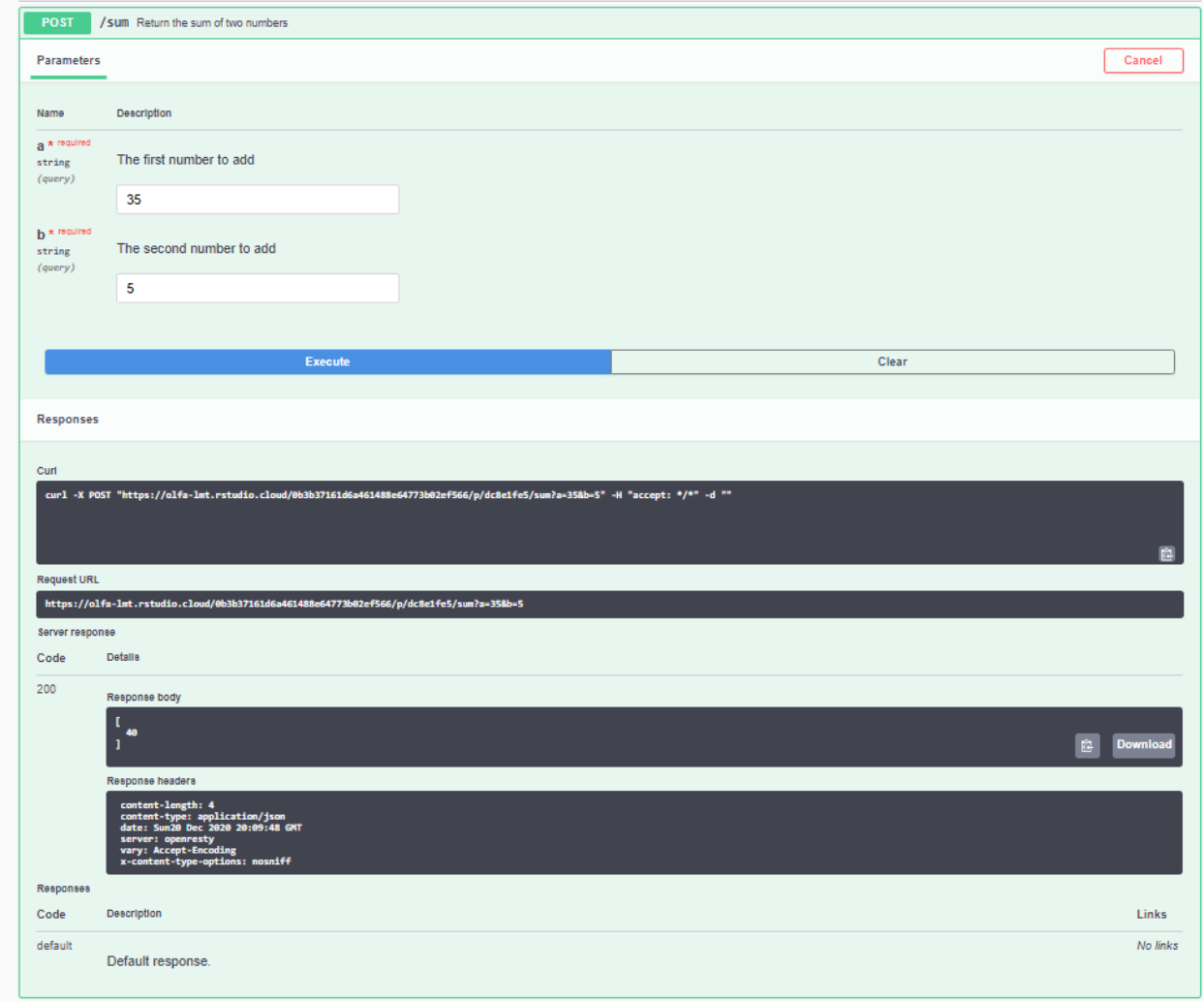
"#* Return the sum of two numbers" : Permet d'avoir un commentaire de la fonction

"#* @param a The first number to add" & "#* @param b The second number to add" : Description des paramètres de la fonction.

"#* @post /sum" : Indique la méthode HTTP utilisée, ici la méthode utilisée est GET. Indique égelement le chemin utilisé, ici plot.

"function(a,b) { ..." : C'est le code en R

Voici le résultat :



On peut voir que qu'en sortie nous avons le résultat de l'addition.

Conclusion

Vous savez mainteant comment faire une API en utilisant des fonction sous R et lançant ces fonctions avec le package Plumber.

Ressources

Voici les ressources que nous avons utilisé

https://www.mulesoft.com/resources/api/what-is-an-api
https://www.tutorialsteacher.com/webapi/what-is-web-api
https://www.uptrends.fr/qu-est-ce-que/rest-api
https://www.youtube.com/watch?v=pCXYzN0HbwA
https://rviews.rstudio.com/2018/07/23/rest-apis-and-plumber/
https://rstudio.com/resources/webinars/plumbing-apis-with-plumber/
https://www.rplumber.io/articles/introduction.html
https://www.youtube.com/watch?v=znHEW5Q6plw

https://thinkr.fr/api-r-deux-temps-trois-mouvements/
https://juanitorduz.github.io/intro_plumber/