Rapport IOT B3

Projet : Dé Connecté

CANAVAGGIO-DIANA MANON

Description du projet

J'ai créé un dé 20, qui fonctionne avec un bot discord. Il fonctionne sans fil et est rechargeable.

Mise en scène ;

Nous sommes 4 personnes et nous jouons à un jeu de rôles à distance, communiquant grâce à Discord. 3 d'entre nous ont un dé, et le 4eme joueur n'en a pas (parce que c'est long à fabriquer...).

Lorsqu'un joueur "lance" son dé, il voit le résultat dessus, et ce résultat est aussi envoyé sur le channel discord où tous les joueurs sur le channel de la partie peuvent le voir.

Celui qui n'a pas encore son dé peut simplement écrire /rand dans le chat discord : cette commande simule un dé, permettant à ceux qui n'ont pas de dé de jouer quand même !

Fonctionnement

Le dé est équipé d'un switch ON/OFF permettant de mettre en route l'objet et d'une batterie rechargeable. Une fois sur ON, il cherchera un réseau wifi connu auquel se connecter (s'il n'en trouve pas, il fonctionnera quand même, mais n'enverra pas de message sur discord.)

Lorsque l'on secoue le dé, il génère un nombre random. Ce résultat est affiché sur la matrice de LEDs lui servant d'écran. Ces deux opérations sont gérées par l'Arduino. Ensuite, l'Arduino envoie ce résultat à l'ESP. L'ESP va se comporter en tant que client du bot discord. Il envoie le numéro obtenu au bot discord via une requête GET sur la REST API du bot.

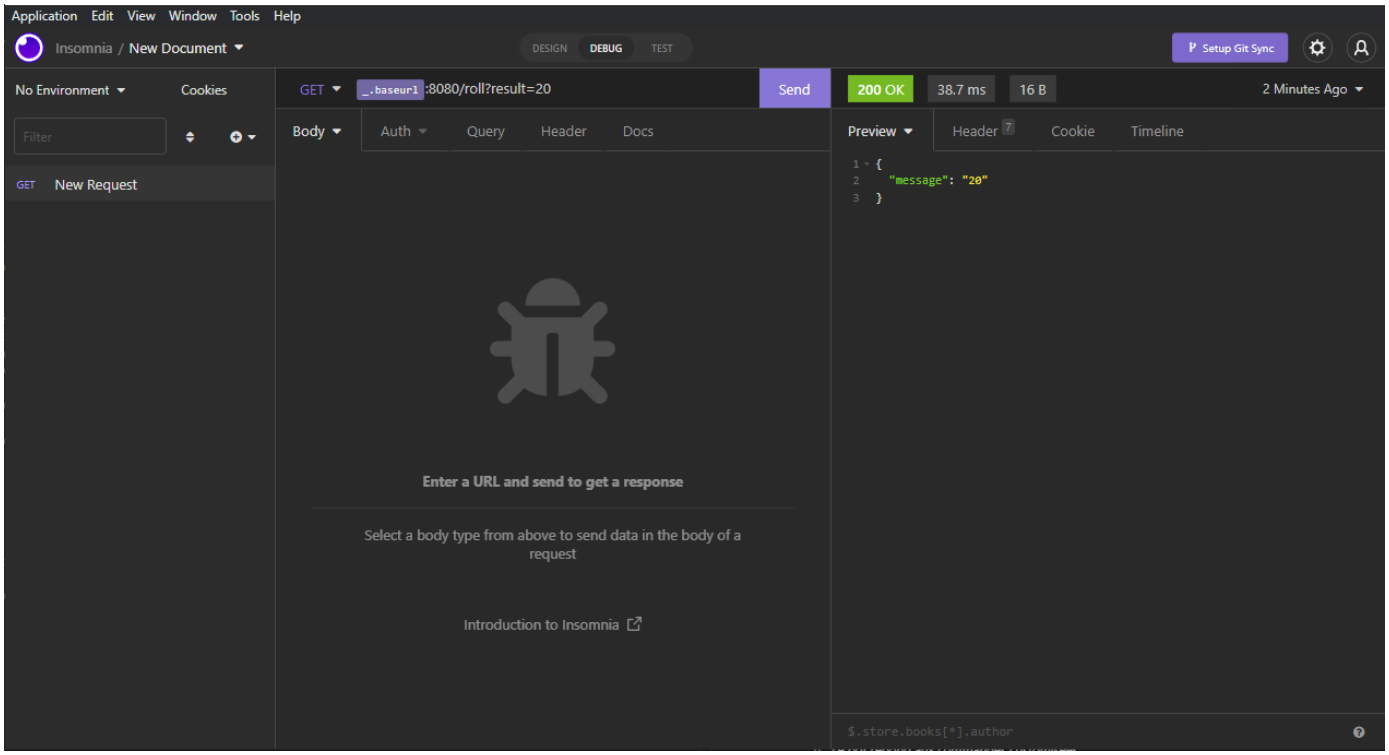
Le bot discord est sur un serveur OVH et a une IP dédiée.

Via l'IP on accède au serveur, via le port on accède au service (le backend du bot/serveur), via une route on accède à la logique associée écrite dans le code.

Le bot discord reçoit le résultat, et en fait un message embed.

Améliorations envisageables

- Utilisation avec Azure (en cours)
- Des options : D4, D6, D10 via une application React Electron
- Une interface pour se connecter au wifi (sans devoir l'hard-coder dans l'ESP)
- Faire le rand sur le serveur directement pour plus de sécurité (si on a l'IP et la route, on peut faire dire n'importe quoi au bot). J'aurais aussi pu authentifier la route avec des credentials, mais la meilleure solution aurait été de laisser le serveur faire toute la logique. Il faudrait enlever l'argument result = 20.
- Améliorer le design
- Trouver une solution au problème suivant : Comment connecter le dé à l'interface, si l'interface sert à connecter le dé au Wifi ?
- Digital Twin



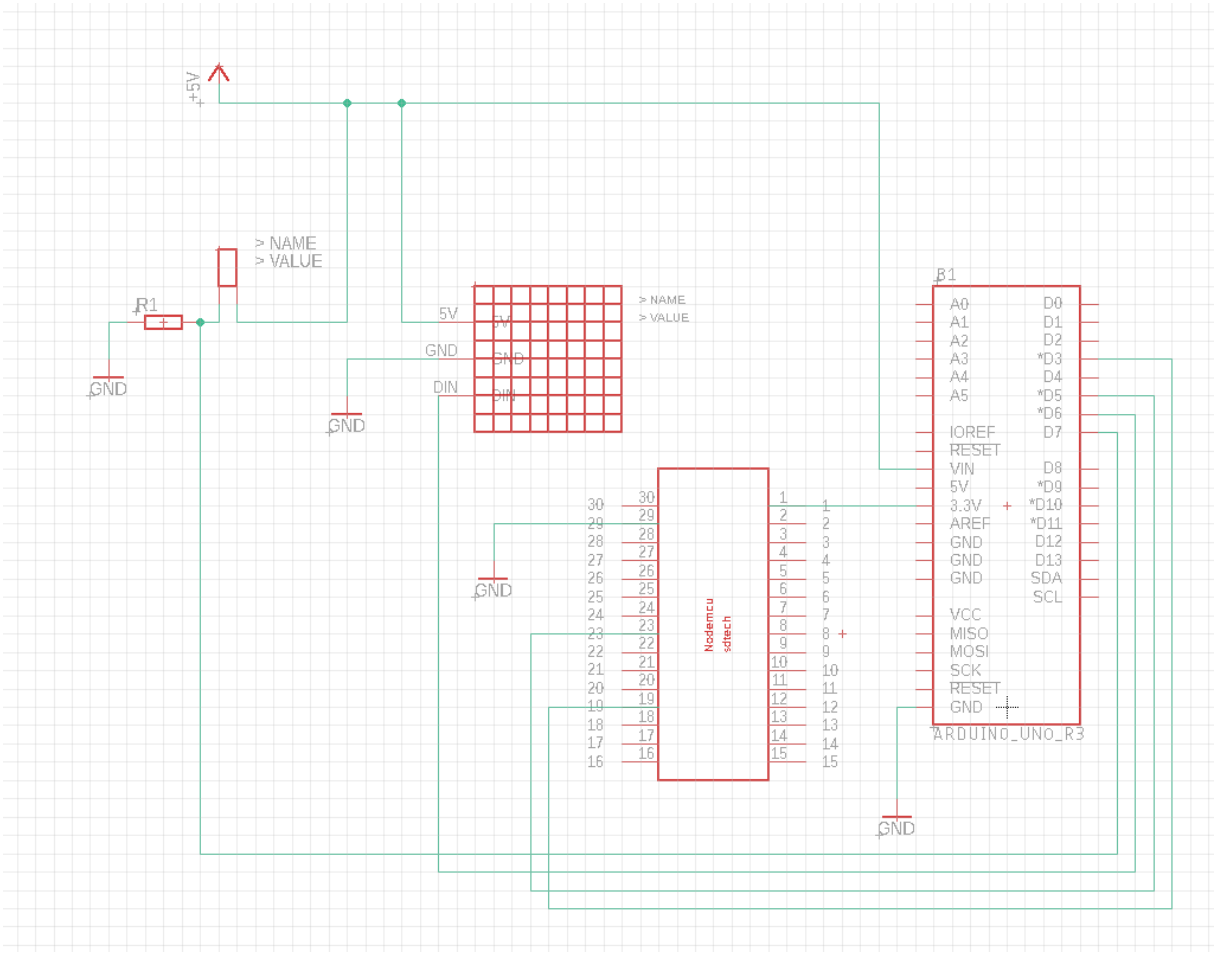
- Dans le cadre de cette requête, ça devrait être un POST. Mais si j'avais fait l'amélioration citée précédemment, j'aurais pu laisser un GET (puisque ça aurait été l'ESP qui recevrait une info du serveur(bot))

Arduino / ESP

Liste des composants :

- Arduino uno
- ESP8266 NODEMCU
- Matrice LEDs RGB 8x8
- Capteur d'inclinaison SW520D
- Resistance 10k x1
- Batterie lithium 18650 3,7V 3000mAh
- Porte-piles convertisseur 5v/3v + chargeur micro USB

J'ai créé un shield à clipser sur l'Arduino afin de limiter le nombre de câbles. Je n'ai cependant pas su reproduire ce shield avec Eagle.



Bot Discord / serveur

Les fichiers principaux sont main.js où on retrouve la partie principale du code, comme par exemple la gestion de l'API REST du bot (qui lui permet de communiquer avec l'ESP) et deploy-command.js qui est le gestionnaire des commandes "slash", commandes que l'on retrouve dans le dossier "commands". Dans le dossier "local-ressources" on retrouve les images utilisées pour faire des jolis messages embed.

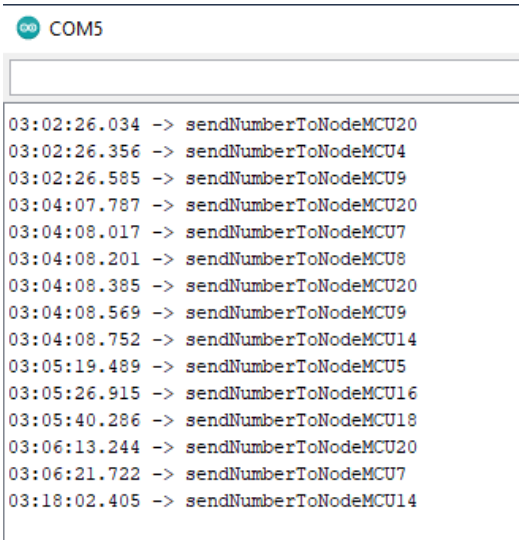
J'ai utilisé express.js, qui permet de faire une API, avec une route /roll sur laquelle je peux faire des GET. Grâce à Discord.js on peut manipuler l'API de discord, et c'est comme ça qu'on crée un bot Discord. EN ajoutant les commandes de Discord.js au sein du GET de express.js, on déclenche des actions dans le bot discord via un trigger WEB.

Améliorations apportées par la suite

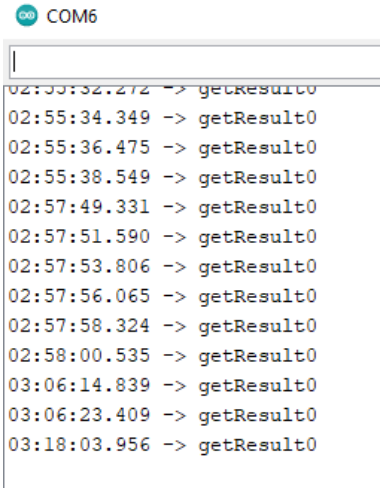
Avant, c'était l'Arduino qui gérait le rand, l'affichage, et le déclanchement pour l'ESP. Maintenant, c'est l'API rest qui gère le rand.

Voici le nouveau fonctionnement du projet :

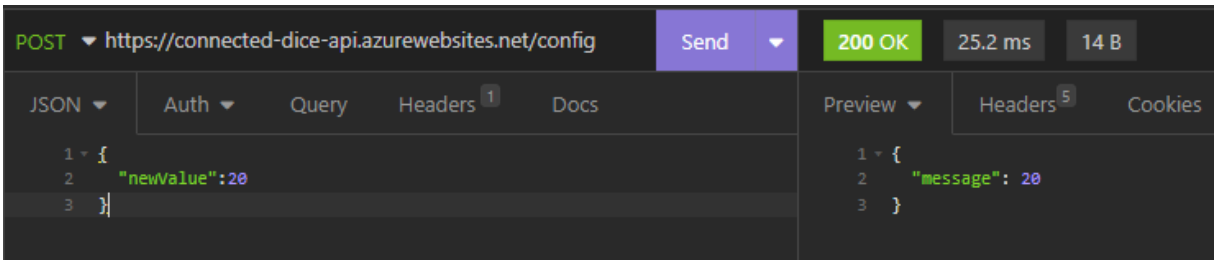
Le dé attend que l'état du Tilt passe à LOW, cela veut dire qu'on secoue le dé. L'état du tilt est envoyé à l'Arduino, qui envoie l'état du tilt à l'ESP (-1 dans result, envoyé via serial). L'ESP fait ensuite un GET sur mon API rest (mon serveur Node.js avec le bot discord). L'API fait le rand et envoie le résultat via un GET à l'ESP. Dans le screen ci-dessous, on voit sur la console que l'on reçoit bien le résultat sur l'ESP.



Ensuite, j'ai rencontré un problème ; en modifiant tout mon code, je n'ai pas prévu que l'on ne pouvait pas read et write sur le même canal série. J'ai essayé d'en ajouter un mais j'obtenais des résultats aléatoires. J'ai finalement opté pour une valeur mock pour l'affichage, afin d'éviter ces résultats inconsistants. Sur le screen ci-dessous, voici la valeur que l'Arduino obtient. Elle devrait être identique à celle du screen précédent. Maintenant, la matrice affichera toujours le chiffre 5.



En ce qui concerne mon application React, j'ai finalement choisi le la faire en Web. Elle permets de configurer le dé et de choisir dans quel pool de valeur il doit faire son rand (Dé 6, dé 4, dé 10, …). Elle envoie la valeur entrée dans l'input à l'API via un POST, et sera prise en compte au prochain « lancer » de dé.



Dans le screen ci-dessus, on peut voir un exemple de ce POST, ici, on demande au dé de faire un lancer dans le pool de valeurs de 1 à 20.

Utilisation d'Azure

J'utilise dans mon projet les API Application, j'ai fait une intégration continue en déployant le code. A chaque fois que je mets ma branche main à jour, ça rebuild le projet et le met à jour sur l'environnement de production. C'est un genre de VM mais seulement pour les projets. Comme j'ai tous mes projets dans un seul repository, j'ai dû faire une commande custom dans l'onglet configuration.