ECOLE SUPERIEURE
D'INGENIEURS DE RENNES

Université de Rennes

Réalisation d’un Boîtier de Commande (Stream deck) connecté en Bluetooth

BROU Justin
EL NAGGAR Sofiane
HANY Matéo
KOBON Charles

Sommaire

I. Présentation du projet

A. Introduction

Ce projet a pour objectif de créer un boîtier de commande, également appelé Stream Deck, connecté en Bluetooth. Ce dispositif permet à l'utilisateur d'interagir facilement avec son ordinateur pour exécuter diverses commandes en appuyant simplement sur des boutons physiques. L'idée est de faciliter et d'accélérer l'accès à des fonctions courantes telles que le contrôle du volume, l'ajustement de la luminosité, l'ouverture de fichiers ou de pages web, et bien plus encore.

B. Situation au début du semestre

Pour rappel, au début du semestre, nous avions déjà quelques fonctionnalités comme la gestion de la luminosité ou du son, les bases du Front étaient déjà là, mais avec beaucoup moins de profondeur et nous avions un prototype de serveur. En revanche, il s'agissait plus de plusieurs pièces prototype fonctionnant à leur échelle, mais n'étant pas capable de s'emboîter correctement. L'objectif du semestre a donc été d'avoir un serveur plus propre, de nouvelles fonctionnalités avec notamment l'apparition de sous-boutons et enfin l'intégration du Raspberry à l'ensemble.

C. Répartition des tâches

- **Charles :** a principalement travaillé sur la page web et tout ce qui concerne la récupération et l'envoi des données au serveur, en plus d'avoir également contribué au développement des traitements de données côté serveur.
- **Justin :** a majoritairement réalisé toutes les configurations nécessaires sur la carte Raspberry Pi, en plus d'avoir configuré la communication Bluetooth entre le PC et le boîtier. Il s'est également occupé de la soudure des différents éléments du boîtier de commande.
- **Matéo:** a principalement mis en place les différentes fonctionnalités du projet et a aussi travaillé sur la communication Bluetooth entre le PC, le boîtier et le serveur.
- **Sofiane :** a majoritairement travaillé sur le backend avec le serveur Node.js pour les différents traitements nécessaires côté serveur, la récupération et l'envoi des données au client. Il a également contribué au développement de l'interface utilisateur.

II. Architecture

Notre projet est composé de plusieurs éléments distincts, il a donc fallu mettre en place une architecture claire dès le début du semestre pour ne pas se perdre dans de mauvaises directions.

Nous avons donc en point central du projet un serveur NodeJs. Pour communiquer avec ce dernier, il y a deux possibilités :

- **Via le Front :** il s'agit de la page web pouvant permettre à l'utilisateur de paramétrer les boutons, mais aussi de simplement interagir avec le serveur. Ce dernier communique via requête HTTP au serveur et les messages sont envoyés dans un format JSON.
- **Via le boîtier de commande :** cette fois-ci, l'utilisateur ne peut pas modifier les paramètres des boutons, il peut simplement interagir avec le serveur en lui demandant d'exécuter les actions liées aux boutons. Le boîtier de commande communique via Bluetooth avec le serveur, une fois de plus au format JSON.

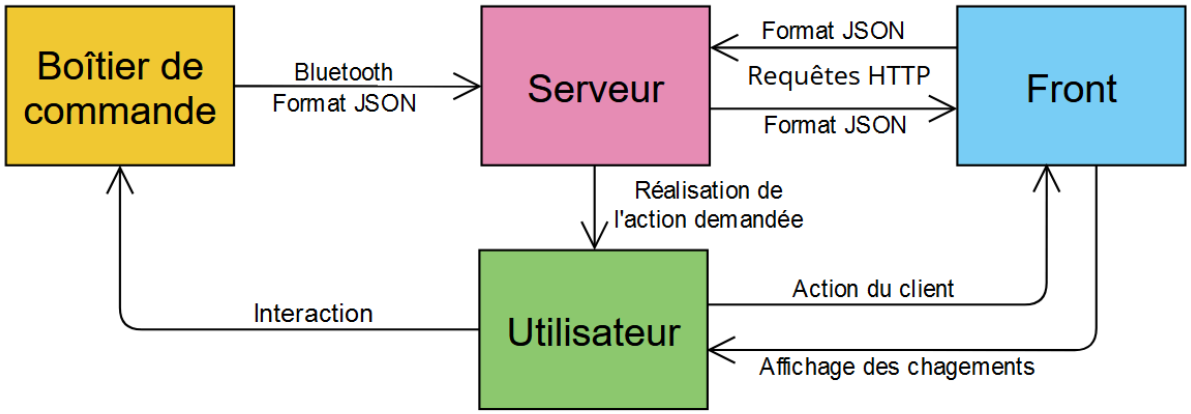

Figure 1 : Architecture globale du projet

A. Front

1. Fonctionnement global

Notre interface utilisateur est constituée de boutons et d'un menu. Nous avons structuré cette interface de la manière suivante :

- **La structure HTML :** qui contient les parties et les éléments qui composent notre page web.
- **Les boutons :** Chaque bouton est associé à un événement onclick qui appelle la fonction clic (this.id) avec l'ID du bouton comme argument. Cette fonction est responsable de la gestion des clics sur les boutons.
- **Le Menu :** Le menu contient plusieurs boutons avec des fonctionnalités différentes. Certains boutons sont "draggables" (peuvent être glissés) tandis que d'autres ont des actions directes. Les boutons qui nécessitent une action supplémentaire ouvrent un popup dans lequel l'utilisateur peut saisir des informations spécifiques.

- **Les fonctions JavaScript :**
    - *allowDrop(event) :* qui autorise le drop d'éléments (utilisé pour le glisser-déposer).
    - *drag(event) :* déclenché lorsque l'utilisateur commence à glisser un élément.
    - *drop(event) :* déclenché lorsque l'utilisateur lâche l'élément glissé.
    - *clic (id) :* Gère les clics sur les boutons en envoyant des données au serveur.
    - *openPopup (option) :* Ouvre un popup en fonction de l'option choisie.
    - *send (jsonData) :* Envoie des données au serveur via une requête POST et met à jour l'interface en fonction des réponses.

2. Choix des technologies

Pour le front, nous avons utilisé 3 outils qui sont :

- **HTML :** Structure de base de la page web, utilisé pour définir la disposition et le contenu.
- **CSS :** Utilisé pour styliser les éléments HTML et rendre la page attrayante et cohérente.
- **JavaScript :** Utilisé pour rendre la page interactive et pour gérer les événements utilisateur (DOM, Event Listeners …)

B. Back

1. Fonctionnement global

Notre serveur est un serveur NodeJs basé sur le module *express*. Ce dernier nous permet d'effectuer facilement des communications entre la partie client et le serveur.

a) Parcours du JSON

Les communications entre le client et le serveur sont réalisées au format JSON comme déjà expliqué dans la partie Front. Le parcours du JSON est le suivant :

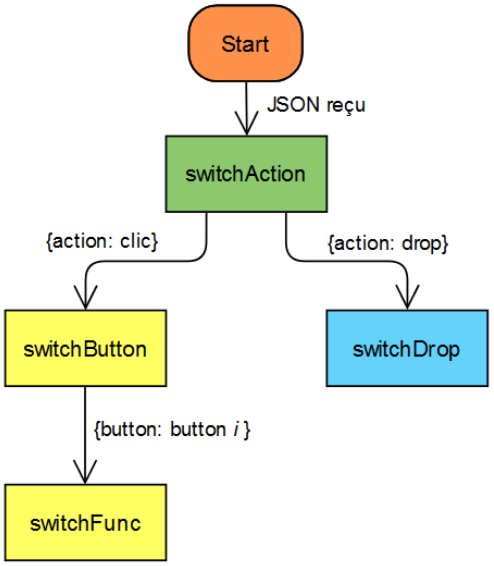

Figure 2 : Parcours du JSON côté serveur

Pour commencer, nous regardons à quel type d'action le JSON reçu fait référence. En effet, soit le JSON reçu fait référence à un *drop* (le fait de faire glisser une fonctionnalité sur l'un des boutons), dans ce cas, l'objectif est de mettre à jour les valeurs des boutons faisant référence à ce drop, soit l'action est un *clic* (le clic du client sur l'un des boutons), et dans ce cas, l'objectif est de réaliser l'action demandée par le client. Si l'action est un *clic*, nous regardons ensuite à quel bouton fait référence l'action et enfin quelle fonctionnalité est associée à ce bouton pour ensuite exécuter cette fonctionnalité. Si l'action est un *drop*, nous mettons à jour les valeurs associées aux boutons.

b) Sauvegarde des données

La sauvegarde des données est gérée de deux façons différentes, mais elle a surtout une structure commune particulière. Dans l'objectif de pouvoir gérer les "sous-boutons", l'option "new buttons" dans nos fonctionnalités, c'est-à-dire, les nouveaux boutons paramétrables sous un premier bouton, les données des boutons sont stockées sous la forme d'arbre. Chaque bouton contient une valeur (ex : brightness up), une option (utile pour "new buttons", "File", "Web" et "Raccourci", indique un paramètre entré par l'utilisateur, dont la fonctionnalité va avoir besoin, ex : CTRL+c) et enfin une liste children, une liste contenant les 5 boutons présents sous ce dernier, les 5 boutons ayant le même format que désigné précédemment avec eu même une valeur, une option et des childrens.

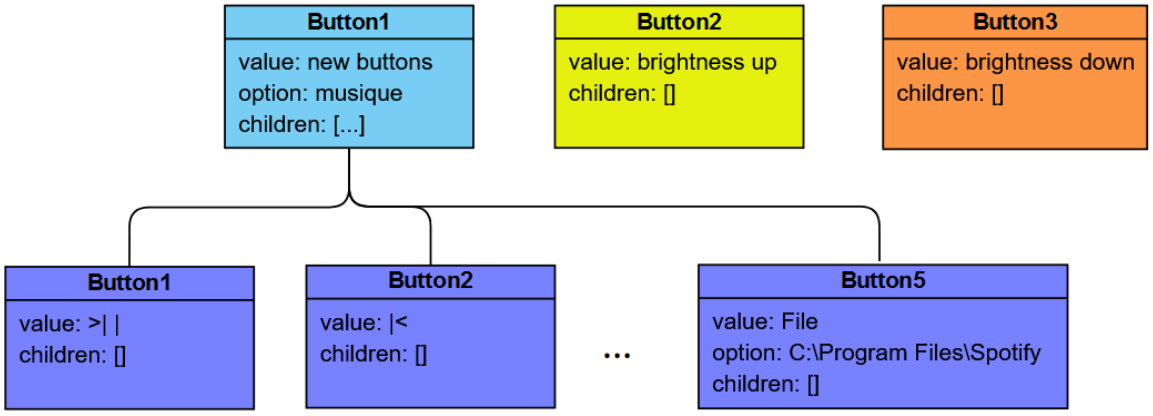

Figure 3 : Exemple de bouton

Nous pouvons par exemple voir sur l'image suivante trois premiers boutons, deux servant à gérer la luminosité, ces derniers n'ont pas d'option et leurs children sont une simple liste vide. En revanche, le Button1 est un new buttons, il a donc une option (dans le cas de new buttons, l'option est un nom servant à différencier les différents sous-boutons) et surtout des enfants, ici les deux premiers sont des gestions de pause et de retour arrière de média, on suppose que le 3 et le 4 sont similaires, on retrouve en bouton 5 un File avec donc en option le chemin du fichier à ouvrir (ici Spotify) et pas d'enfant. Nous notons qu'il n'y a pas de bouton 6 dans les sous-boutons, ce dernier est réservé au bouton retour pour pouvoir remonter d'un niveau dans l'arbre.

Ces 6 boutons (et leurs enfants) sont stockés dans une liste et c'est cette dernière qui interagit majoritairement avec le serveur, que ce soit pour l'exécution de fonctionnalité ou l'envoie d'information au front.

En revanche, il y a un autre emplacement où nous sauvegardons les valeurs des boutons. Dans l'objectif de sauvegarder les paramètres des boutons en cas de fermeture du serveur (pratique attendue et logique de l'utilisateur), il nous faut un format plus dur qu'une simple liste. Pour ce faire, nous sauvegardons également les valeurs dans un fichier JSON. L'objectif est de regarder à chaque démarrage de l'application le contenu de ce fichier et de venir modifier les valeurs des boutons contenues dans la liste montrée précédemment. Pour cela, il faut également que le contenu du JSON au démarrage soit bon, c'est pour cela qu'à chaque modification de la liste initiale, nous modifions également le JSON pour que ce dernier contienne les bonnes informations en cas d'arrêt du serveur.

Pour des problématiques de complexité suite à l'implémentation de la sauvegarde en JSON, il n'est pas possible d'avoir des sous-boutons à l'infini. Bien que théoriquement possible, nous avons choisi de limiter le maximum de niveau à 3, ce qui nous permet tout de même d'avoir accès à un maximum de 150 possibilités de boutons configurables.

Dans le cas où nous voudrions effacer le contenu complet de nos boutons (avec Reset), nous disposons d'un fichier JSON contenant l'équivalent du fichier JSON de sauvegarde, mais en vierge. Nous modifions donc juste le contenu de JSON de sauvegarde par le JSON vierge et appliquons le contenu de la sauvegarde à la liste initiale.

2. Choix des technologies

Pour le serveur, nous avons décidé de faire ce dernier en NodeJs pour les raisons suivantes :

- **Facilité :** En effet, pour une application web de notre type, faire un serveur NodeJs est ce qui est le plus répandu et donc ce qui contient le plus de documentation et module prés existant
- **Connaissance :** Nous avions déjà utilisé NodeJs en cours et plutôt que de vouloir apprendre une nouvelle technologie, nous avons préféré approfondir nos connaissances sur cette dernière.
- **Appréciation de la technologie :** Il faut aussi l'admettre, nous apprécions tous travailler sur NodeJs, notamment l'équipe qui s'est occupée du serveur et travailler avec une technologie appréciée est toujours un plus.

C. Communication Front/Back

Les communications entre le Front et le Back sont gérées à l'aide du module *express* de NodeJs, il s'agit donc de requêtes HTTP. Nous avons fait le choix d'utiliser le format JSON pour la communication. En effet, ce format facilite grandement le partage d'information avec un format clair et efficace.

1. Front vers Back

Côté Front, le message à deux formats différents dû aux deux cas de figure :

- **Drop :** On retrouve dans le message une action (ici toujours drop), un bouton correspondant au bouton sur lequel est trop la nouvelle fonctionnalité, un texte qui est à la fois le nouveau contenu du bouton, mais aussi une indication sur quelle fonctionnalité a été drop et enfin une option dans certains cas. Le JSON a donc le format suivant :

```
{
     "action": "drop",
     "button": "button i",
     "text": "fonctionnalité",
     "option": "default"
}
```

- **Clic :** Dans ce cas, nous retrouvons toujours l'action qui sera cette fois toujours clic, avec cette dernière, nous ne retrouvons que l'indication du bouton cliqué. Le JSON a donc le format suivant :

```
{
     "action": "clic",
```

```
    "button": "button i"
}
```

Par la suite, le serveur traite les données JSON comme décrit dans la partie **Parcours du JSON**.

2. Back vers Front

Côté Back, les informations envoyées au Front ont pour but de l'informer de l'état actuel des boutons à afficher. La majorité du temps, les informations envoyées ont seulement pour but de confirmer la bonne réception des messages reçus, mais, en cas de changement de niveau si l'utilisateur vient à descendre dans un new buttons ou à remonter d'un niveau dans l'arbre, le Front n'a aucune idée de ce qu'est censé contenir le nouveau niveau de bouton. Pour correctement informer le Front, nous indiquons l'action que nous faisons (soit nous montons, soit nous descendons, soit nous ne faisons que confirmer). Le JSON a donc la forme suivante :

```
{
    "action": "ok",
    "level": [0],
    "path": [0],
    "buttonsValues": ["brightness up", "brightness down", ..., "new
buttons"],
    "option": [null, null, ..., "Musique"],
}
```

Les différentes informations du JSON désignent :

- ***action :*** soit *ok* (simple confirmation, soit *down* (indique que nous descendons dans un niveau), soit *up* (indique que nous remontons d'un niveau).
- ***level*** : indique le niveau de profondeur dans lequel nous nous trouvons (maximum 2 donc dû à la limitation de profondeur).
- ***path*** : indique le trajet pris pour arriver à ce niveau de profondeur, le numéro du bouton sera utilisé comme indication.
- ***buttonsValues*** : indique les valeurs des boutons dans le niveau actuel. L'information est utilisée pour modifier l'affichage des boutons dans le Front.
- ***option :*** tout comme la valeur précédente, cette dernière change les informations dans les boutons sur le Front. Cette fois-ci, seules les informations d'option sont contenues dans la liste, s'il n'y en a pas, elle est remplacée par *null*.

D. Bluetooth

1. Fonctionnement global

La communication Bluetooth est une partie importante de notre projet, permettant au boîtier de commande (Stream Deck) de se connecter sans fil à un ordinateur et de transmettre les actions des boutons au serveur. L'objectif est de créer une interaction fluide et réactive entre l'utilisateur et les fonctionnalités de l'ordinateur via Bluetooth Low Energy (BLE).

Voici une image de notre boîtier de commande actuel, figure 4, équipé de six boutons connectés à notre Raspberry Pi Pico. Les boutons sont numérotés de la droite vers la gauche (bouton 1, bouton 2, bouton 3, bouton 4, bouton 5 et bouton 6).

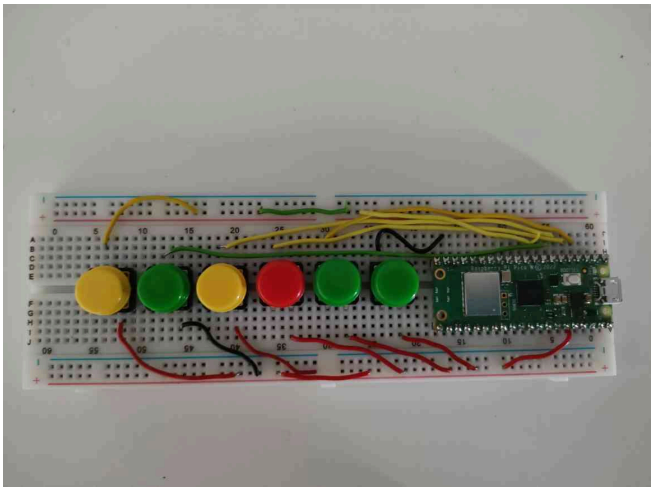

Figure 4 : Montage du Raspberry Pi

2. Choix des technologies

Pour la communication Bluetooth, nous avons choisi d'utiliser la bibliothèque **bleak** en Python. **Bleak** est une bibliothèque légère et compatible avec plusieurs plateformes, qui facilite la gestion des connexions et des communications BLE. Les principales raisons de ce choix sont :

- **Compatibilité Multi-Plateforme :** Fonctionne sous Windows, macOS et Linux.
- **Facilité d'Utilisation :** API simple et intuitive pour les opérations BLE.
- **Documentation et Support :** Bien documentée et largement utilisée, avec une bonne communauté de supports.

3. Protocole de communication

Le protocole de communication entre le boîtier de commande et le serveur est basé sur Bluetooth Low Energy (BLE), avec un format de données en JSON pour assurer une

transmission claire et structurée des informations. Voici les détails du protocole de communication :

a) Bluetooth Low Energy (BLE)

Dans ce projet, le BLE est utilisé pour établir une connexion entre le boîtier de commande et l'ordinateur.

b) Services et Caractéristiques GATT

Le protocole BLE utilise le profil GATT (Generic Attribute Profile) pour la communication. Les informations sont échangées sous forme de **services** et de **caractéristiques :**

- **Service UART (Universal Asynchronous Receiver-Transmitter) :** Ce service est utilisé pour la communication série entre le boîtier de commande et le serveur.
    - **UUID du service UART** : **6E400001-B5A3-F393-E0A9-E50E24DCCA9E**
    - **Caractéristiques :**
        - **TX (Transmission) :** Utilisée pour envoyer des données du boîtier de commande au serveur.
            - **UUID** : **6E400003-B5A3-F393-E0A9-E50E24DCCA9E**
            - **Propriétés :** Notification
        - **RX (Réception) :** Utilisée pour recevoir des données du serveur au boîtier de commande.
            - **UUID** : **6E400002-B5A3-F393-E0A9-E50E24DCCA9E**
            - **Propriétés :** Write, Write Without Response

c) Format des Données

Les données échangées entre le boîtier de commande et le serveur sont formatées en JSON pour garantir une communication structurée et compréhensible. Voici un exemple de format des données :

```
{
  "button": "button1"
}
```

- **button** : Indique quel bouton a été pressé sur le boîtier de commande.

d) Processus de Communication

Ci-dessous, nous avons la description du processus:

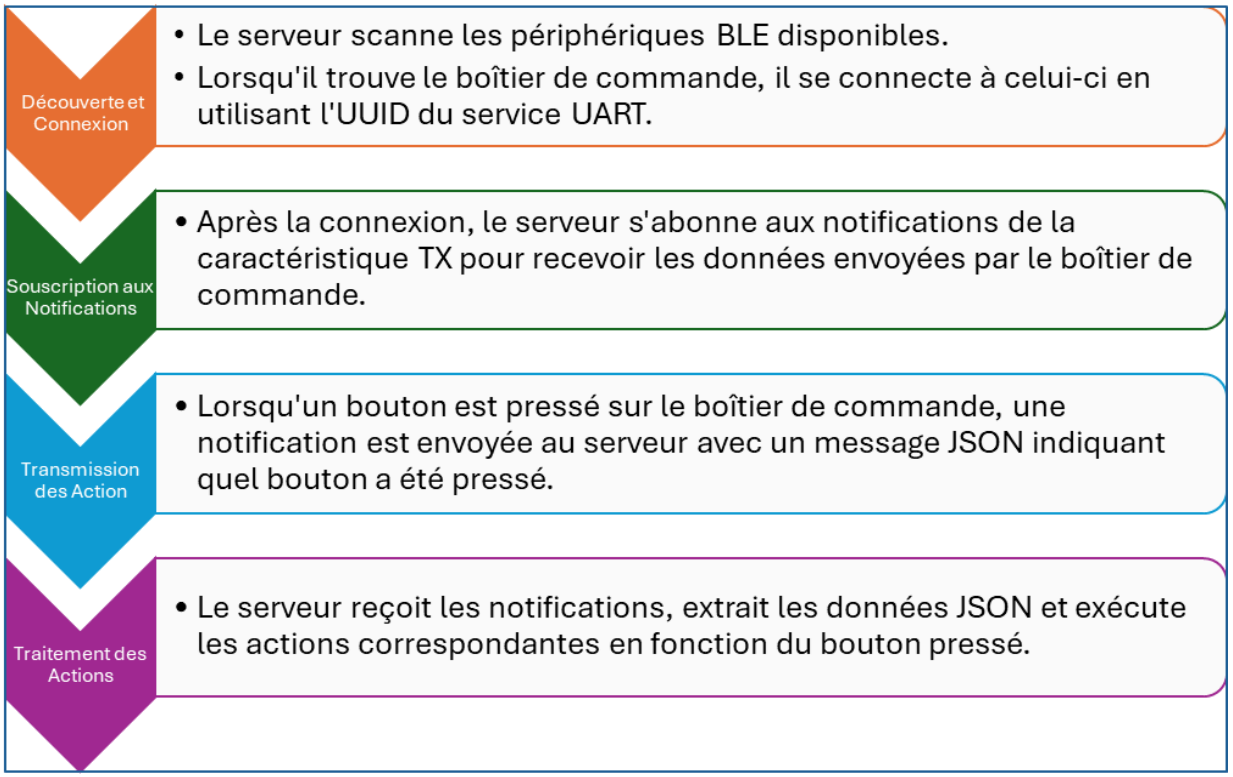

Figure 5 : schématisation du processus de connexion et communication Bluetooth

III. Fonctionnalités

Les fonctionnalités du projet de Stream Deck connecté au Bluetooth ont été soigneusement conçues pour offrir une expérience utilisateur riche et intuitive. Matéo a pris en charge cette partie du projet et a développé les fonctionnalités suivantes :

Contrôle du Son et de la Luminosité : L'utilisateur peut ajuster le volume sonore ainsi que la luminosité de son environnement directement depuis le Stream Deck, offrant ainsi un contrôle pratique et rapide sans avoir à naviguer à travers plusieurs menus. La fonctionnalité de contrôle du volume sonore utilise le module “loudness” pour gérer les niveaux de volume de l’ordinateur. Ce module Node.js permet d’obtenir et de régler le volume système. L’utilisateur peut augmenter ou diminuer le volume sonore par palier de deux unités et de mettre en sourdine le son si le volume atteint 0. La fonctionnalité de contrôle de la luminosité utilise le module “brightness”. Ce module Node.js permet de lire et de régler la luminosité d'écran. L’utilisateur peut augmenter ou réduire la luminosité d'écran par paliers de 10%.

Ouverture de Fichiers et de Pages Web : Une fonctionnalité clé permet à l'utilisateur d'ouvrir des fichiers locaux ou des pages web en appuyant simplement sur un bouton du Stream Deck. Pour tester ces fonctionnalités, il suffit de rentrer le parcours du fichier et le

lien du site web dans la case prévus à cet effet et de le valider Cela facilite l'accès à des ressources essentielles sans avoir à interrompre le flux de travail. En entrant le chemin du fichier exécutable(.exe) d'une application, l'utilisateur peut lancer des applications directement depuis le Stream Deck. Cette fonctionnalité élimine la nécessité de naviguer manuellement dans le menu démarrer ou sur le bureau pour trouver et lancer une application.

Modules Utilisés
child_process : Ce module, fourni par Node.js, permet d'exécuter des commandes shell depuis un script Node.js. Il est utilisé pour ouvrir des pages web et lancer des applications.

Capture d'Écran : En un seul clic, l'utilisateur peut capturer l'écran de son ordinateur, offrant ainsi une solution rapide et efficace pour sauvegarder des informations importantes ou partager des contenus visuels. Les captures d'écran sont enregistrées côté client, avec un nom de fichier formaté selon la date et l'heure de la capture, par exemple screenshot_16_05_2024_11_28_01.png.

Modules Utilisés

screenshot-desktop : Ce module permet de capturer l'écran entier de l'ordinateur.
Fs (File System) : Utilisé pour vérifier l'existence de dossiers, créer des dossiers si nécessaire, et écrire des fichiers sur le disque.
path : Utilisé pour manipuler les chemins de fichiers de manière compatible avec différents systèmes d'exploitation.

Utilisation de Raccourcis Clavier :

Le Stream Deck connecté au Bluetooth propose un menu intuitif où l'utilisateur peuvent assigner des raccourcis claviers personnalisés. En entrant simplement le raccourci désiré dans le menu et en le validant, ils peuvent associer une action spécifique à chaque bouton. Lorsqu'ils appuient sur l'un de ces boutons, le Stream Deck envoie le raccourci clavier correspondant à l'ordinateur, permettant ainsi d'exécuter rapidement des commandes dans différentes applications. Cette approche offre une flexibilité totale pour automatiser des tâches courantes et accéder rapidement à des fonctionnalités spécifiques, améliorant ainsi l'efficacité et la productivité de l'utilisateur.

Contrôle de Lecture Audio/Video : Les fonctionnalités de contrôle de lecture permettent à l'utilisateur de mettre en pause, avancer ou reculer dans une playlist audio ou vidéo en cours de lecture, offrant ainsi un contrôle total sur le contenu multimédia.

Malgré plusieurs tentatives, nous n'avons pas réussi à mettre en œuvre la fonctionnalité permettant d'avoir une liste de toutes les applications et de choisir laquelle ouvrir directement depuis le Stream Deck. La complexité de récupérer tous les fichiers d'exécution (.exe) associés aux applications a posé un défi technique difficile à surmonter dans le cadre du projet.

Ces fonctionnalités offrent une expérience utilisateur complète et polyvalente, améliorant ainsi l'efficacité et la facilité d'utilisation du Stream Deck connecté au Bluetooth.

IV. Post Mortem

Bien que nous soyons fiers du résultat présenté pour ce projet, il y a toujours des améliorations possibles. Pour commencer, nous aurions aimé ajouter plus de fonctionnalités, notamment des fonctionnalités basées sur la communication avec des API d'application préexistante comme Discord ou Voicemod par exemple. Nous aurions également aimé pouvoir lister les applications et les ouvrir directement depuis notre application plutôt que de devoir fournir le path de cette dernière. Pour finir, l'un des modules que nous utilisons demande l'installation de NirCmd, nous aurions préféré ne pas avoir d'outil supplémentaire à installer, ce dernier nécessitant de plus de la paramétrer dans les variables d'environnement.

Conclusion

Le projet de réalisation d'un boîtier de commande connecté en Bluetooth, également appelé Stream Deck, a été une expérience enrichissante et formatrice pour toute l'équipe. Partant d'une base fonctionnelle avec des prototypes indépendants, nous avons su intégrer et développer de nouvelles fonctionnalités tout en consolidant l'architecture globale du projet.

Au début du semestre, nous avons établi des objectifs clairs pour l'amélioration et l'intégration de divers composants. Cela a inclus l'enrichissement du Front avec de nouvelles options, l'optimisation du serveur NodeJs, et la mise en place d'une communication efficace entre le boîtier, le PC et le serveur via Bluetooth. Chaque membre de l'équipe a apporté une contribution significative, permettant une répartition équilibrée des tâches et une progression constante.

Les choix technologiques, tant pour le front-end (HTML, CSS, JavaScript) que pour le back-end (NodeJs) et la communication Bluetooth (bibliothèque bleak en Python), ont été guidés par notre expérience et les besoins spécifiques du projet. La structure en JSON pour

les échanges de données a permis une gestion claire et efficace des informations entre les différentes parties du système.

Le projet a également permis d'explorer et de maîtriser diverses fonctionnalités pratiques et utiles, telles que le contrôle du son et de la luminosité, l'ouverture de fichiers et de pages web, la capture d'écran, l'utilisation de raccourcis clavier, et le contrôle de lecture audio/vidéo. Bien que certaines fonctionnalités, comme la liste des applications exécutables, n'aient pas pu être implémentées, l'ensemble des fonctionnalités développées offre une interface utilisateur riche et intuitive.

En conclusion, ce projet nous a permis de renforcer nos compétences techniques et de gestion de projet. Nous avons su surmonter les défis techniques et collaborer efficacement pour aboutir à un produit fonctionnel et polyvalent. Le Stream Deck connecté en Bluetooth représente non seulement un aboutissement de nos efforts collectifs, mais également une base solide pour de futurs développements et améliorations.