INSTITUT
UNIVERSITAIRE
DE TECHNOLOGIE
UNIVERSITÉ D'ANGERS

université
angers

MacroPad

Projet réalisé par

Pierre Alexandre HERVE, Mathis RUDKIEWICZ et Hugo DELARUELLE

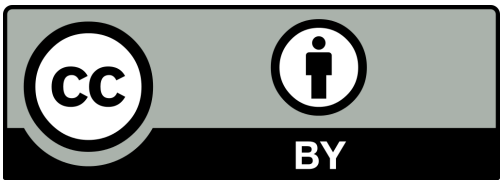
Projet encadré par

Philippe Lucidarme

IUT Angers-Cholet
4 boulevard Lavoisier - BP 42018
49016 - Angers Cedex

Année 2022-2023

Ce rapport présente le travail réalisé par un groupe d'étudiants dans le cadre d'un projet pédagogique. Les auteurs et l'Université d'Angers ne garantissent pas que l'information, les documents, la méthodologie et le matériel présentés dans ce document soient complets, conformes à l'état de l'art et exacts ni n'assurent en toutes circonstances la sécurité des biens, des personnes et des utilisateurs. Les auteurs et l'Université d'Angers ne seront pas tenus responsables des dommages éventuels qui pourraient résulter de l'utilisation du contenu du présent rapport.



Remerciements

Tout d’abord, nous souhaitons remercier l'ensemble des personnes citées ci-dessous pour leur contribution à ce projet :

- ★ M. Philippe Lucidarme, enseignant-chercheur à l’Université d’Angers, pour le temps qu’il nous a consacré, les connaissances qu’il nous a apporté et sa bienveillance
- ★ L’IUT d’Angers-Cholet et plus particulièrement le département GEII pour le matériel et les lieux mis à disposition ainsi que l’achat des composants nécessaires au projet
- ★ Mme. Denécheau pour la vérification et la commande des PCB

Enfin, nous remercions de manière générale toute personne ayant, de près ou de loin, participé à la réalisation de ce projet de SAE.

Sommaire

Introduction

Dans le cadre du 4e semestre de notre formation en BUT GEII, nous avons entrepris un projet visant à concevoir un clavier muni de raccourcis personnalisables. Ce projet a pour objectif d'offrir une solution pour faciliter et optimiser le temps de plusieurs catégories d'utilisateurs. En effet, ce clavier polyvalent s'adresse aussi bien aux fervents joueurs de jeux vidéo qu'aux développeurs et à tous ceux qui utilisent régulièrement un ordinateur.

De plus, notre choix s'est porté sur ce projet car nous avons tous reconnu la nécessité d'un tel produit dans nos expériences quotidiennes. Que ce soit pour des tâches professionnelles, des activités ludiques ou des projets personnels, nous avons ressenti le besoin d'un clavier personnalisable doté de raccourcis afin d'accroître notre efficacité et d'améliorer notre expérience utilisateur.

Ce rapport détaillé présentera les différentes phases de conception et de développement de notre clavier, ainsi que les choix technologiques et méthodologiques. Nous décrirons également les tests et les évaluations effectués pour garantir la qualité et les performances du produit final.

I. Le cahier des charges

A. Les spécifications techniques

Notre objectif est de réaliser un clavier de raccourcis sans fil configurable via une application. Nous avons donc deux parties dans ce cahier des charges : une première qui définit les spécifications du clavier et une deuxième qui détermine les aspects de l'application de configuration.

Nous avons choisi les caractéristiques suivantes pour notre clavier : Tout d'abord, tout se fait de manière sans fil. La configuration se fait en Wifi via un serveur websocket qui est hébergé au sein d'une l'application Electron. Une trame JSON est envoyée à chaque changement de configuration sur l'IHM. Les actions, quant à elles, sont envoyées via Bluetooth à l'appareil cible. Les intéractions entre l'utilisateur et le clavier se font par le biais de switchs et d'encodeurs qui ont respectivement pour rôle de réaliser des actions et de modifier les différents volumes. Enfin, le clavier est alimenté par une batterie ou par un câble micro-USB.

Le cahier des charges de notre application de configuration du clavier repose sur plusieurs critères essentiels. Notre objectif principal est de fournir un logiciel simple, intuitif et facile à prendre en main pour permettre aux utilisateurs de personnaliser leur clavier de manière efficace. Le but est, pour l'utilisateur, d'avoir la possibilité de mapper des fonctions spécifiques à chaque touche en fonction de ses besoins.

B. Les livrables

Nous rendons 3 livrables à côté de ce rapport : le clavier de raccourci sans fil qui est constitué d'un PCB avec 9 touches, 3 encodeurs et un ESP32 pluggé dessus. Avec ce PBC est livré le code arduino qui pilote l'ESP32.

Le code a pour fonction de faire le lien entre le hardware et le software : il gère l'envoi des inputs en bluetooth, le client du serveur WebSocket pour la configuration et la détection de l'appui des touches et des mouvements des encodeurs. Enfin, il y a également l'application de configuration qui est fournie avec.

Nous accordons une importance primordiale à la simplicité de l'application. Nous voulons concevoir une interface épurée, éliminant toute complexité inutile. La facilité de prise en main de l'application est une composante essentielle pour nous. Les utilisateurs doivent pouvoir commencer à configurer leur clavier rapidement, sans avoir à passer par des étapes complexes.

En respectant ces exigences du cahier des charges, nous visons à créer un logiciel qui réponde aux attentes des utilisateurs en matière de facilité d'utilisation, de personnalisation et d'accessibilité. Notre objectif est de permettre à chacun de tirer pleinement parti des avantages offerts par notre clavier personnalisable, en lui offrant un usage agréable et adapté à ses besoins individuels.

II. Les choix technologiques

A. Les choix hardwares

1. La carte de développement ESP32

Nous allons vous présenter les différentes raisons qui nous ont poussé à choisir la carte de développement ESP32 pour piloter les différentes fonctions de notre clavier de raccourcis personnalisable.

Tout d'abord, nous avions besoin d'un microcontrôleur capable de gérer des technologies sans fil, comme le Bluetooth et le Wifi, pour répondre à notre cahier des charges. De plus, ce microcontrôleur devait pouvoir prendre en charge l'environnement de développement Arduino pour gagner du temps. Enfin, il devait consommer peu afin que notre clavier puisse tenir longtemps sur batterie. Pour répondre à nos besoins, nous avions deux choix qui s'offraient à nous : un ESP32 et un EPS8266.

Ensuite, nous avons consulté la documentation technique des deux microcontrôleurs et nous avons pu lire dans celle de l'ESP8266 qu'il possède un seul cœur, ce qui signifie que l'on peut exécuter une seule tâche à la fois. L'ESP32 en possède deux et nous permet donc d'effectuer 2 tâches en parallèle. De plus, L'ESP32 possède également deux fois plus d'entrées / sorties, ce qui ouvre plus de possibilités d'évolutions.

Enfin, nous avons choisi ESP32 DevKit au lieu d'un ESP32 SoCs. En effet, cette option avait l'avantage d'être simple à mettre en œuvre et fiable. Elle nous assurait le bon fonctionnement des entrées / sorties et des périphériques de l'ESP32. Nous n'avions pas le droit à l'erreur à cause des timings serrés et avons décidé de ne pas nous engager dans cette voie. En revanche, la carte de développement ESP32 a le désavantage de prendre énormément de place sur notre PCB.

2. Un PCB plug & play

Un PCB plug and play est conçu pour être compatible avec un large éventail de périphériques et de composants. Cela signifie qu'il peut être utilisé avec différents types d'ESP32 sans avoir besoin de modifications ou d'adaptations supplémentaires. Cela possède plusieurs avantages :

-Une réduction du temps de développement : En utilisant un PCB plug and play, la conception électronique est réduite en termes de temps. Au lieu de concevoir et de

fabriquer un PCB personnalisé, nous avons simplement intégré un PCB plug and play existant dans notre produit.

- L'évolutivité : Les PCB plug and play offrent une certaine flexibilité en termes de mise à niveau et de remplacement des composants. Il est souvent plus facile d'ajouter de nouvelles fonctionnalités ou de remplacer des pièces défectueuses sur un PCB plug and play par rapport à un PCB personnalisé.

B. Les choix softwares

1. Le bluetooth : effectuer les actions sans-fil

Un de nos premier besoin était de pouvoir effectuer de manière sans fil des actions sur l'appareil auquel notre clavier est connecté. Pour ce faire, nous avons tout de suite pensé à la technologie bluetooth qui répond totalement à nos besoins. En effet, cette technologie est présente sur la plupart des appareils modernes et peut s'implémenter facilement au moyen d'une clé peu coûteuse. De plus, il consomme peu d'énergie avec la norme BLE (Bluetooth Low Energy)

Par ailleurs, nous avons déniché une librairie arduino permettant de simuler un clavier bluetooth. Cette librairie se nomme Ble-keyboard et a été développée par T-vk. Elle permet d'effectuer directement des actions sur un appareil équipé d'une puce Bluetooth comme simuler l'appui d'une touche ou encore de régler le volume. Cette librairie clé en main nous a permis de gagner énormément de temps de développement.

2. Le websocket pour la configuration des touches

Nous avons eu plusieurs idées pour la configuration. Notre première idée était d'envoyer la configuration via Bluetooth avec la librairie node Bluetooth. Cette librairie permettait de créer une communication série entre le logiciel Electron et l'ESP32. Or, il s'est avéré que cette librairie n'était pas compatible avec la version que nous utilisions de node js. Nous avons donc opté pour une autre solution qui est un serveur WebSocket.

Un serveur WebSocket est un type de serveur qui permet une communication bidirectionnelle en temps réel entre un serveur et un client via une connexion persistante. Contrairement aux protocoles HTTP traditionnels, qui sont basés sur un modèle de requête-réponse, les WebSockets permettent une communication en temps

réel continue et sans surcharge supplémentaire. Voici comment fonctionne généralement un serveur WebSocket :

Tout d'abord, un handshake initial : Lorsqu'un client souhaite établir une connexion WebSocket avec un serveur, il envoie une demande spéciale appelée "handshake" pour initier la communication. Le serveur répond ensuite avec une réponse de handshake pour confirmer la connexion.

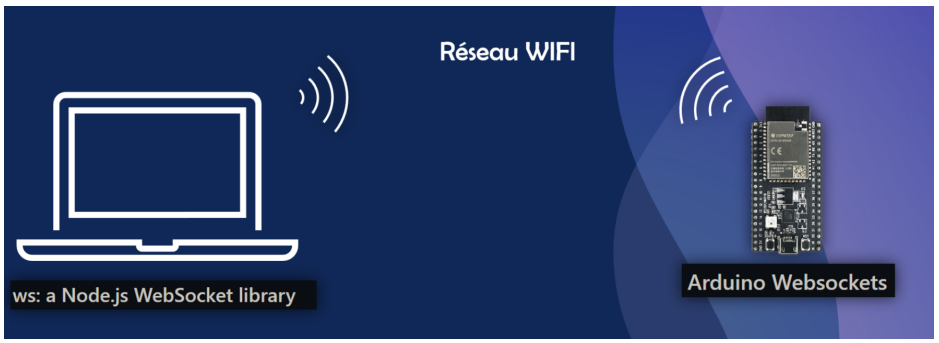
Ensuite, une fois que le handshake initial est réussi, une connexion persistante est établie entre le client et le serveur. Cela signifie que le serveur et le client peuvent envoyer des données l'un à l'autre sans avoir à établir une nouvelle connexion pour chaque communication.

Une fois la connexion établie, le serveur et le client peuvent s'échanger des messages à tout moment. Un message peut être une simple chaîne de caractères, des données binaires ou même des objets JSON. Les messages sont encapsulés dans des trames (frames) WebSocket pour être transmis sur la connexion.

Le serveur WebSocket peut écouter et répondre à différents types d'événements. Par exemple, il peut être configuré pour exécuter une action spécifique lorsqu'un nouveau client se connecte, lorsqu'un message est reçu ou lorsqu'un client se déconnecte.

Lorsque le client ou le serveur souhaite mettre fin à la connexion WebSocket, il envoie un message de fermeture pour indiquer son intention. Les deux parties échangent ensuite des messages de fermeture jusqu'à ce que la connexion soit correctement fermée.

Le protocole WebSocket est conçu pour être compatible avec les infrastructures existantes, telles que les serveurs HTTP et les pare-feu, ce qui le rend largement utilisé dans les applications Web en temps réel, telles que les chat en ligne, les jeux en ligne, les tableaux de bord en direct, etc.



Pour mettre en œuvre ce protocole, nous avons utilisé la librairie Arduino Websockets. Elle nous permet de faire communiquer l'ESP32 avec l'application Electron. Les données de configurations sont envoyées depuis l'application via des trames JSON qui sont ensuite décodées sur l'ESP.

3. Electron : une solution simple pour un logiciel multiplateformes

Le Framework Electron est une technologie open-source qui permet de créer des logiciels multiplateformes en utilisant des langages web tels que HTML, CSS et JavaScript. Il est largement adopté par de nombreuses entreprises et développeurs pour la création d'applications comme Twitch, Visual Studio Code et d'autres...

Électron est basé sur une architecture en deux parties : le processus principal d'un côté (main process) et de l'autre les processus de rendu (renderer processes). Le processus principal gère les fonctionnalités du système d'exploitation et les interactions avec l'interface utilisateur, tandis que les processus de rendu sont responsables de l'affichage des pages web dans les fenêtres de l'application.

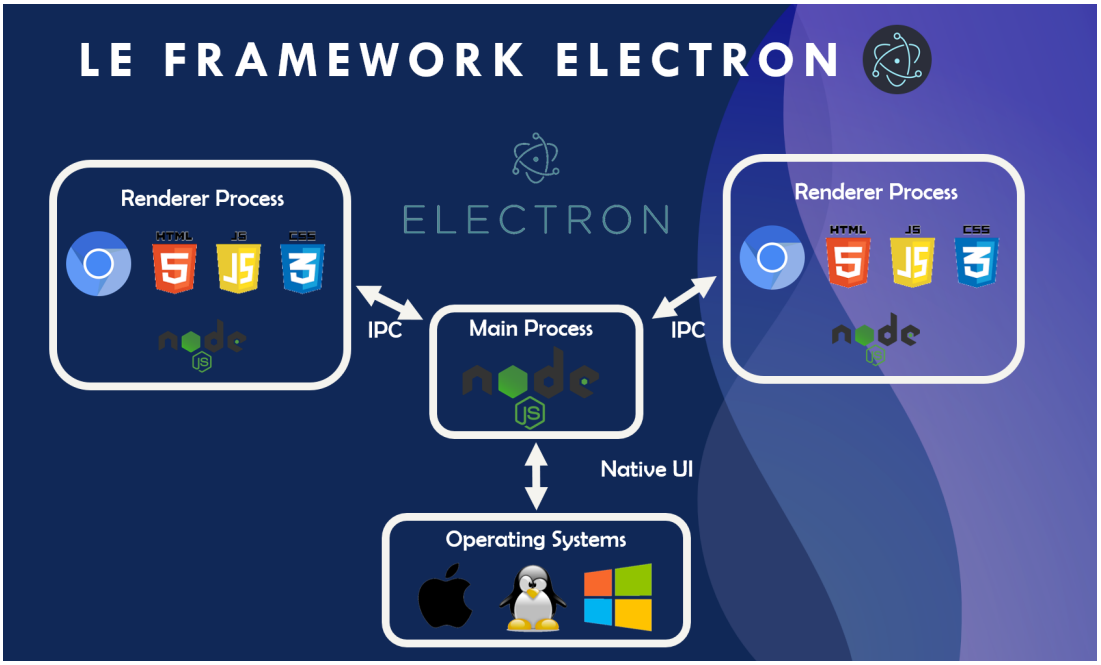


Schéma présentant le fonctionnement global d'Electron

Une fois que nous avions développé notre application Électron, nous pouvions la packager sur différentes plateformes (Windows, macOS, Linux). Cela permet de distribuer l'application sous forme de fichiers exécutables, prêts à être installés sur les ordinateurs des utilisateurs.

En outre, Electron offre une intégration étroite avec le système d'exploitation, permettant d'accéder à des fonctionnalités avancées telles que les notifications, le système de fichiers et les raccourcis clavier. Il nous permet de tirer parti de nos compétences en développement web pour créer des applications riches et performantes.

III. La réalisation

A. Notre première idée

Dans le cadre de notre projet, nous avions initialement envisagé de reprendre le PCB développé par Noah Justeau et Dorian Benech comme solution de secours pour pallier les contraintes de temps. Cependant, après réflexion, nous avons préféré nous concentrer sur notre idée initiale et partir de zéro afin de concevoir un clavier personnalisable répondant à nos propres besoins spécifiques.

Cette décision nous a permis d'explorer de nouvelles fonctionnalités telles qu'un clavier sans fil, muni d'encodeurs, ce qui n'étaient pas réalisables avec la carte déjà existante. Ainsi, en choisissant de partir de zéro, nous avons pu créer un produit unique et adapté à nos exigences.

B. La solution choisie

1. Le schéma électronique

Pour choisir les ports d'entrées sorties les plus appropriés aux différents composants nous sommes allés voir la documentation technique de l'ESP32, et avons regardé les fonctions que propose chaque port. Nous utilisons un ESP32 wroom 32D de la marque Firebeetle, mais les fonctions des ports entrées/sorties restent les mêmes que celles de la marque Espressif Systems. Le Soc est identique

Le document ci-dessus est le pin definition. Il répertorie toutes les fonctions que peut réaliser chaque port de la carte. Il est important de prendre en compte chaque fonction afin d'optimiser les possibilités futures, lors de la programmation d'un firmware.

Nous avons donc fait le choix d'utiliser au maximum les pins compatibles avec le Deep sleep, afin d'optimiser dans le futur la consommation du système au niveau du software. Les switchs ainsi que les codeurs incrémentaux sont tous connectés au Deep sleep/RTC.

Name	No.	Type	Function
IO32	8	I/O	GPIO32, XTAL_32K_P (32.768 kHz crystal oscillator input), ADC1_CH4, TOUCH9, RTC_GPIO9
IO33	9	I/O	GPIO33, XTAL_32K_N (32.768 kHz crystal oscillator output), ADC1_CH5, TOUCH8, RTC_GPIO8
IO25	10	I/O	GPIO25, DAC_1, ADC2_CH8, RTC_GPIO6, EMAC_RXD0
IO26	11	I/O	GPIO26, DAC_2, ADC2_CH9, RTC_GPIO7, EMAC_RXD1
IO27	12	I/O	GPIO27, ADC2_CH7, TOUCH7, RTC_GPIO17, EMAC_RX_DV
IO14	13	I/O	GPIO14, ADC2_CH6, TOUCH6, RTC_GPIO16, MTMS, HSPICLK, HS2_CLK, SD_CLK, EMAC_TXD2
IO12	14	I/O	GPIO12, ADC2_CH5, TOUCH5, RTC_GPIO15, MTDI, HSPIQ, HS2_DATA2, SD_DATA2, EMAC_TXD3
GND	15	P	Ground
IO13	16	I/O	GPIO13, ADC2_CH4, TOUCH4, RTC_GPIO14, MTCK, HSPID, HS2_DATA3, SD_DATA3, EMAC_RX_ER
SHD/SD2*	17	I/O	GPIO9, SD_DATA2, SPIHD, HS1_DATA2, U1RXD
SWP/SD3*	18	I/O	GPIO10, SD_DATA3, SPIWP, HS1_DATA3, U1TXD
SCS/CMD*	19	I/O	GPIO11, SD_CMD, SPICS0, HS1_CMD, U1RTS
SCK/CLK*	20	I/O	GPIO6, SD_CLK, SPICLK, HS1_CLK, U1CTS
SDO/SD0*	21	I/O	GPIO7, SD_DATA0, SPIQ, HS1_DATA0, U2RTS
SDI/SD1*	22	I/O	GPIO8, SD_DATA1, SPID, HS1_DATA1, U2CTS
IO15	23	I/O	GPIO15, ADC2_CH3, TOUCH3, MTDO, HSPICS0, RTC_GPIO13, HS2_CMD, SD_CMD, EMAC_RXD3
IO2	24	I/O	GPIO2, ADC2_CH2, TOUCH2, RTC_GPIO12, HSPIWP, HS2_DATA0, SD_DATA0
IO0	25	I/O	GPIO0, ADC2_CH1, TOUCH1, RTC_GPIO11, CLK_OUT1, EMAC_TX_CLK
IO4	26	I/O	GPIO4, ADC2_CH0, TOUCH0, RTC_GPIO10, HSPIHD, HS2_DATA1, SD_DATA1, EMAC_TX_ER
IO16	27	I/O	GPIO16, HS1_DATA4, U2RXD, EMAC_CLK_OUT
IO17	28	I/O	GPIO17, HS1_DATA5, U2TXD, EMAC_CLK_OUT_180
IO5	29	I/O	GPIO5, VSPICS0, HS1_DATA6, EMAC_RX_CLK
IO18	30	I/O	GPIO18, VSPICLK, HS1_DATA7
IO19	31	I/O	GPIO19, VSPIQ, U0CTS, EMAC_TXD0
NC	32	-	-
IO21	33	I/O	GPIO21, VSPIHD, EMAC_TX_EN
RXD0	34	I/O	GPIO3, U0RXD, CLK_OUT2
TXD0	35	I/O	GPIO1, U0TXD, CLK_OUT3, EMAC_RXD2
IO22	36	I/O	GPIO22, VSPIWP, U0RTS, EMAC_TXD1
IO23	37	I/O	GPIO23, VSPID, HS1_STROBE
GND	38	P	Ground

Pinout de du DevKit Espressif ESP32 V4

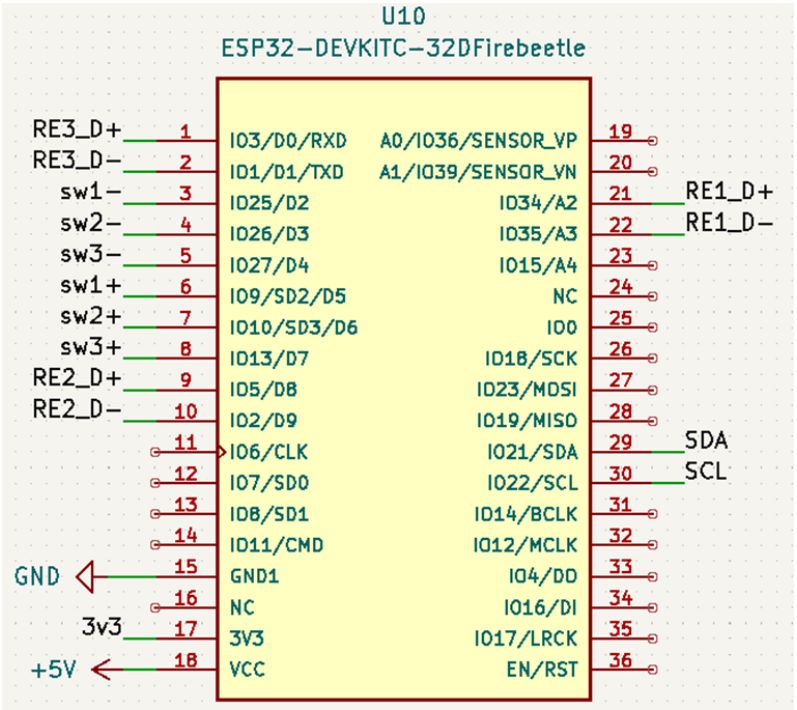
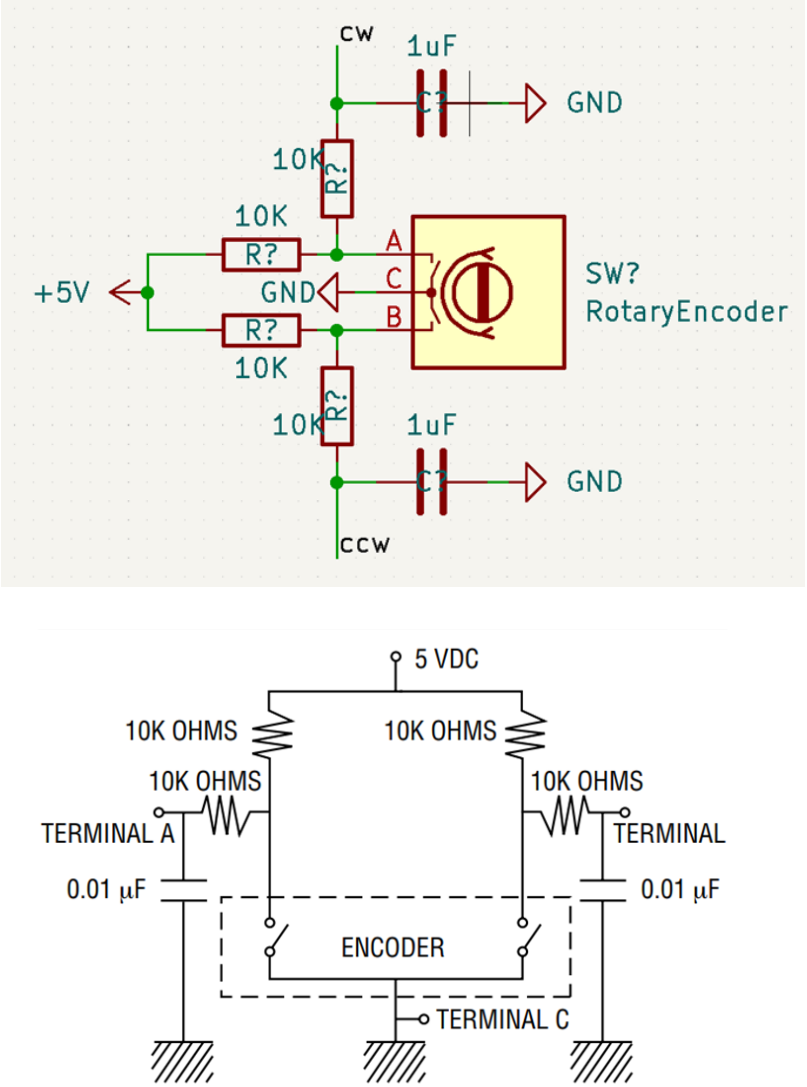


Schéma électronique de l'ESP32

Le pin CLK et SDA sont connectés à une sortie de connecteur. Nous avons prévu cela au cas où nous voudrions ajouter un écran Oled dans le futur. (CLK+SDA = bus I2C).

Pour la réalisation du schéma d'un encodeur incrémental, il faut prendre en compte que le Pin A et B sont des sorties de data. Le pin C quant à lui est une sortie commune (masse). On peut chez certain fabriquant se retrouver avec 5 pins. Les deux pins qui s'ajoutent sont généralement des pins reliés au boîtier métallique du composant. (Il faudra donc les relier à la masse).

De plus, il faut faire attention. Certains constructeurs suggérés dans la datasheet de mettre en place un système de filtrage en sortie des pins data (A et B)



Schémas électroniques d'un encodeur

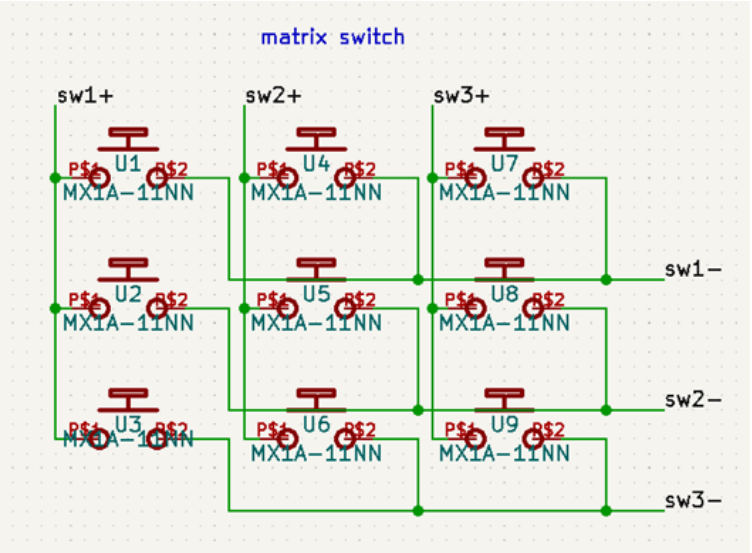


Schéma électronique du clavier matriciel

La réalisation du schématique d'un clavier matriciel est similaire au fonctionnement d'un pull up (résistance de tirage).

Nous allons détailler :

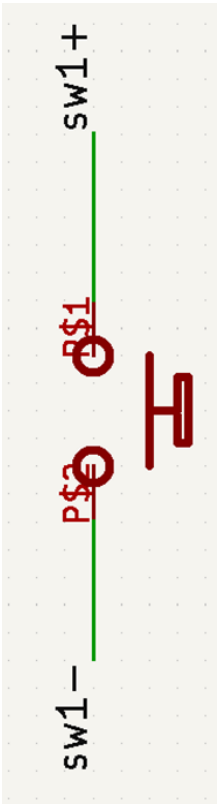


Schéma d'un switch

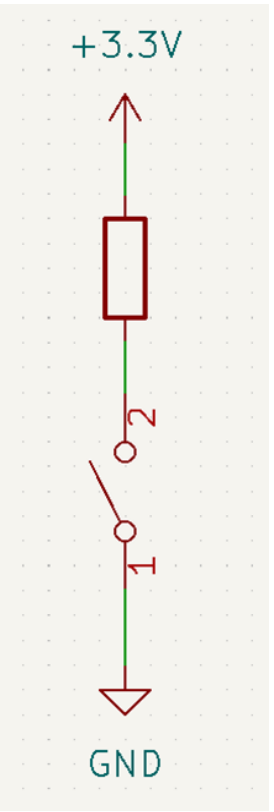
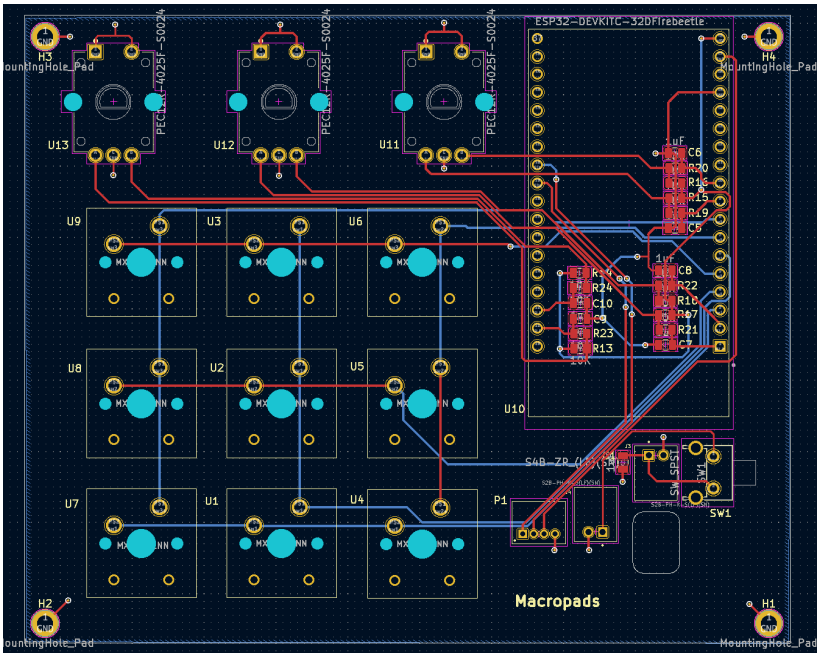


Schéma d'une pull-up

Les deux schémas présentés ci-dessus, ont le même fonctionnement. Sw1+ est le 3.3V (relié sur un pin du microcontrôleur). Et le sw1- lui est le GND (relié sur un autre pin du microcontrôleur). Et quand nous allons appuyer sur le bouton poussoir (switch du clavier), le courant va pouvoir passer d'un bout à l'autre.

Le clavier matriciel est donc une architecture optimisée pour occuper moins de pin du microcontrôleur. Et fonctionne exactement comme un pull up. (sw1+ et sw1- peuvent être inversés comme bon vous semble).

2. Le PCB



Routage du PCB

Pour le routage du PCB, nous avons opté pour une technologie à double couche, car nous disposions d'un espace suffisant et cela nous a permis de réduire les coûts de fabrication.

En ce qui concerne le placement des composants, nous avons privilégié les interrupteurs en premier plan, suivis des codeurs. Nous avons inclus un trou dans la carte pour permettre la connexion de la batterie. De plus, nous avons positionné l'interrupteur sur l'un des côtés pour faciliter la prise en main de l'appareil.

3. La programmation de l'ESP32

Dans cette partie, nous allons nous attarder sur la programmation de l'ESP32 et notamment sur la communication Bluetooth, la réception des trames JSON pour la configuration et la gestion du clavier matriciel. Un clavier matriciel est un ensemble de

boutons disposés en une matrice rectangulaire. Lorsqu'un bouton est enfoncé, il crée une connexion entre une rangée et une colonne spécifiques, ce qui permet de déterminer quelle touche a été pressée.

Tout d'abord, la classe KeyConfig est définie pour gérer la configuration des touches. Elle a pour but de détecter, avec l'aide de la fonction ToucheAppuyee, les appuis sur les différentes touches et les mouvements des encodeurs. Elle a également pour rôle d'effectuer les actions en sur l'appareil cible en Bluetooth avec l'aide de la librairie BleKeyboard.

Ensuite, la fonction ReceivedKeyConfiguration a pour mission de recevoir et de décoder les trames JSON venant du logiciel Electron. Ces trames contiennent les informations de configuration telles que la combinaison des touches pour effectuer l'action, le type de touche (classique, media...) ou encore le numéro de la touche auquel l'action doit être affectée.

Après, on peut observer les fonctions Bluetooth et Configuration. Ces deux fonctions vont chacune tournée sur un cœur de l'ESP32 différent. Ainsi les processus de configuration et d'envoi d'action sont totalement indépendants ce qui signifie que la configuration ne va pas venir déranger l'envoi des actions et inversement.

Enfin, les fonctions EEPROMSave et EEPROMLoad vont permettre la sauvegarde de la configuration du clavier au sein de l'ESP32. Cela évite de perdre les paramètres de la dernière utilisation. Ainsi, pas besoin de le configurer à chaque démarrage, les actions vont se charger automatiquement.

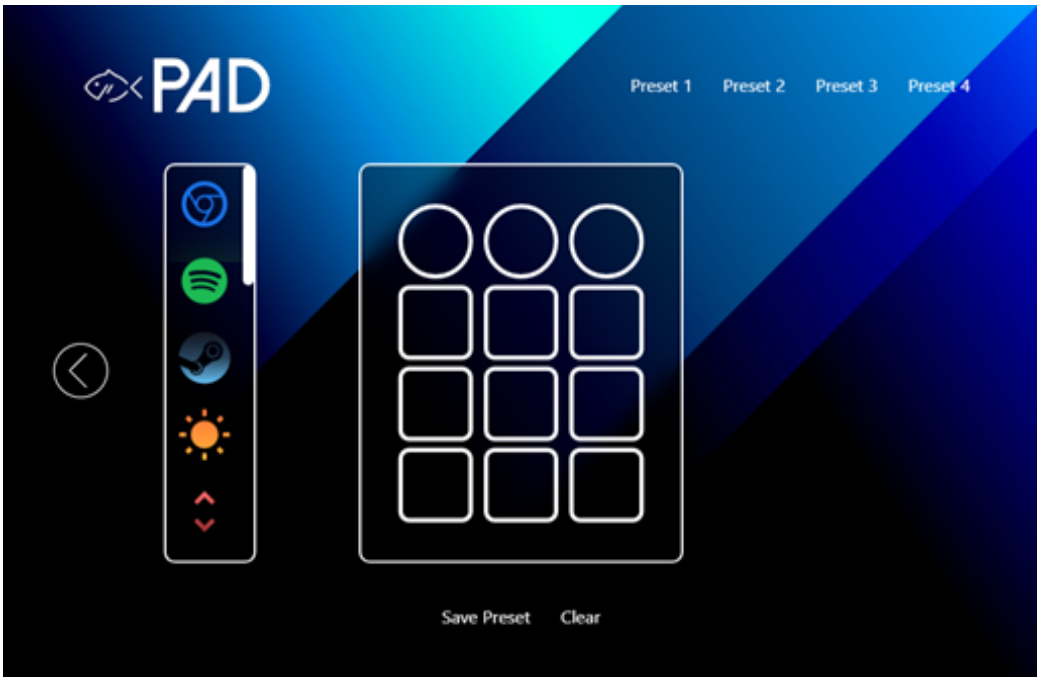
En résumé, ce programme permet de contrôler un clavier matriciel, de recevoir et sauvegarder une configuration de touches et d'envoyer des entrées clavier ou média via une connexion Bluetooth. Cela rend notre clavier personnalisable et permet de contrôler d'autres appareils compatibles avec Bluetooth, tels que des ordinateurs ou des smartphones.

4. La réalisation du logiciel Electron

Le design de notre application répond aux mêmes objectifs que les fonctionnalités énoncées dans le cahier des charges, mettant l'accent sur la simplicité et l'intuitivité. Le front end a été codé en HTML, CSS ainsi que du javascript pour des fonctions basiques. Le code complet se situe dans le dossier archive.

L'interface de l'application est conçue de manière à ce que les utilisateurs puissent configurer rapidement et facilement les touches du clavier. Nous avons créé

un menu clair et organisé qui répertorie tous les raccourcis disponibles, permettant aux utilisateurs de trouver rapidement les fonctions qui les intéressent. De plus, pour configurer une touche rapidement, nous avons choisi la fonctionnalité "drag and drop" (glisser-déposer) offrant une expérience fluide.



Visuel de l'IHM Electron

Afin de faciliter davantage la configuration du clavier, nous avons inclus des presets personnalisables dans l'application. Ces presets sont des configurations qui peuvent être sauvegardées. Les utilisateurs peuvent ensuite choisir le preset correspondant à leurs besoins et l'appliquer rapidement à leur clavier, ce qui leur fait gagner du temps et éviter d'avoir à configurer chaque touche individuellement.

L'application comprend une représentation visuelle du clavier au centre de l'interface, offrant une vue d'ensemble intuitive. Les touches sont représentées graphiquement, permettant aux utilisateurs de visualiser facilement la configuration actuelle et les changements effectués.

En plus de l'application nous avons conçu un logo et une identité de marque pour accompagner la création du clavier. Concernant le logo, nous voulions un design minimaliste, qui suit la tendance prise par les grandes marques. Le logo utilise un symbole graphique distinctif, ce choix est destiné à renforcer la reconnaissance et l'attrait visuel du logiciel.

Visuel de l'IHM Electron pour mobile

En matière de design visuel, nous avons adopté une approche qui met en valeur la simplicité, la clarté et l'accessibilité. L'application est conçue de manière « responsive », s'adaptant à différents types d'écrans et de résolutions, pour offrir une expérience cohérente selon les besoins.

C. Les difficultées rencontrées

1. Un projet (trop) ambitieux

Nous avons commencé les deux semaines de projets avec des idées plein la tête et de très grandes ambitions comme une application super design, super responsive et un joli boîtier pour la carte de notre clavier. La réalité nous a très vite rattrapée et nous nous sommes rendus compte que nous n'aurions jamais le temps de tout réaliser. En effet, nous pensions pouvoir avancer chacun de notre côté durant les périodes d'entreprises mais pour des raisons matérielles, de fatigue et de motivation, nous n'avons pas réussi à obtenir les résultats escomptés. En outre, nous nous sommes certainement trop éparpillés sur des détails au lieu de nous concentrer sur le développement des fonctions essentielles.

En plus de cela, c'est ajouté à cela un retard de livraison qui nous a empêché d'avancer correctement sur le projet.

2. Des retards de livraison

Lors de la réception de notre PCB, nous avons rencontré un problème logistique. En effet, notre carte a été perdue au sein même de l'IUT. Nous n'étions malheureusement pas présents lors de la livraison de celle-ci au bureau de Mme. Denécheau (nous sommes tous les trois alternants). Le temps de se rendre compte que la carte était introuvable et de relancer une production, nous avons perdu presque trois semaines sur notre planning.

3. Des erreurs de conception

L'utilisation d'un ESP32 pour ce projet était un choix réfléchi et approprié en fonction de nos besoins. Cependant, opter pour un DEVkit d'une marque autre que le

distributeur officiel n'était pas une bonne idée. Nous avons utilisé un ESP32 D Wroom de la marque Firebeetle, distribué par le site DFrobot. Cela nous a posé des problèmes lors de la réalisation du PCB. Ce modèle de DEVkit a été peu vendu car la marque officielle "Espressif" propose ses propres DEVKIT, ce qui a entraîné des incompatibilités au niveau des empreintes (disposition des broches) et a rendu le processus plus long que prévu. De plus, en cas de panne de notre ESP32, il sera plus difficile de se procurer un DEVkit peu courant. Il aurait été plus judicieux d'utiliser le DEVkit de la marque officielle pour éviter ces problèmes de compatibilité et d'approvisionnement.

De plus, nous avons rencontré des problèmes avec le routage, notamment dans le choix des pins : Nous avions relié un encodeur sur les pins Rx et Tx de l'ESP32 ce qui rendait la liaison série impossible. En plus de cela, nous n'avons pas disposé les touches au bon endroit lors du routage ce qui signifie que nous avons dû ajuster l'ordre des touches au sein même du code. Cette modification rend le code plus difficile à lire et à modifier.

IV. La conclusion et les perspectives d'amélioration

A. Ce que l'on retient de ce projet

Grâce à ce projet, on peut maintenant affirmer sans aucune hésitation, que le travail d'équipe ainsi que l'organisation au sein d'un groupe de travail sont primordiaux. Cela nous a permis de réaliser certains de nos objectifs plus rapidement avec une charge de travail individuelle moindre.

Nous avons aussi appris de nos erreurs. En effet, nous avons certainement été trop ambitieux. Nous avions l'espoir en début de projet de réaliser plus de fonctionnalités. Le temps a joué contre nous, et la non prise en compte d'éventuels aléas logistiques nous a fait perdre un temps précieux.

Pour la répartition des tâches, nous avons décidé de répartir le travail en fonction des points forts de chacun. Cela nous a permis de gagner du temps en termes d'étude logicielle/hardware mais aussi sur nos commandes chez les différents fournisseurs. La réalisation des différents rapports du projet a donc aussi été plus rapide grâce à cette méthode.

B. Ce que l'on peut améliorer

1. La gestion du projet

Pour la gestion de projet, la principale amélioration que l'on pourrait faire à l'avenir serait l'ajout de réunions afin de "brainstormer" sur la fin de chaque étape du projet. Cela permettrait de ce fait de discuter des exigences de la prochaine étape à effectuer et de garder en tête l'objectif principal du projet.

J'ajouterais que l'utilisation du logiciel ClickUp n'est pas le meilleur choix pour la gestion de projet. En effet l'utilisation du calendrier pour la planification des objectifs, des dates de livraisons ainsi que des deadlines, n'est pas très ergonomique et difficile à prendre en main.

2. Les choix techniques et la réalisation

Dans un premier temps, nous n'avons pas eu le temps de réaliser un boîtier pour habiller notre PCB ni d'acheter des keycaps pour couvrir les touches et les encodeurs. Ces ajouts d'ordre esthétique "finiraient" le produit et sont essentiels pour rendre le produit utilisable au quotidien.

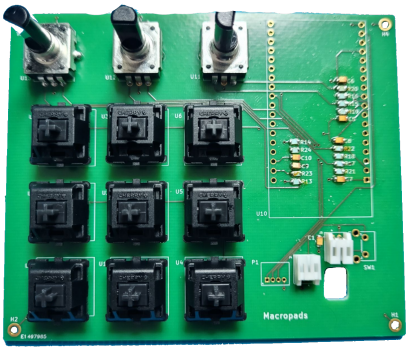
De même, il est important d'ajouter une page html auto-hébergée sur l'ESP32 pour la configuration du réseau Wifi pour le fonctionnement du serveur WebSocket. Actuellement, il faut entrer le SSID et le mot de passe du réseau utilisé directement dans le code et ensuite le téléverser sur l'ESP32.

Par ailleurs, il serait judicieux de faire une deuxième version du PCB afin de corriger les erreurs et peut-être aussi gagner en place sur la carte. Egalement, il faudrait mesurer la consommation totale de l'ensemble et dimensionner une batterie pour pouvoir le rendre autonome en énergie.

Enfin, il faut reprendre le code de l'application Electron pour le rendre fonctionnel. En effet, actuellement il fait juste figure de modèle esthétique mais ne permet pas réellement de configurer notre clavier contrairement à l'application de démonstration qui elle, n'est pas esthétique mais fonctionnelle. L'application de démonstration qui fonctionne avec le clavier se trouve dans le dossier "*Electron-appFonctionelle*" et l'application développée par Mathis et qui est à améliorer se trouve dans le dossier "*Electron-appDesign*" dans le pack livré avec le rapport.

Le résumé

Le projet *Macro Pad* a été réalisé par des étudiants de deuxième année de BUT GEII en alternance. Il a pour objectif de rendre l'utilisation des raccourcis ergonomiques pour tous types d'utilisateurs. Ainsi avec notre clavier *Maquereau Pad*, il est possible de réaliser un *Copier-Coller* en appuyant seulement sur deux touches. Il fonctionne à 100% sans fil et est 100% configurable.



Abstract

The Macro Pad project was carried out by second-year BUT GEII students on a work-study basis. Its objective is to make the use of ergonomic shortcuts for all kinds of users. Thus with our Macropad keyboard, it is possible to perform a Copy-Paste by pressing only two keys. It works 100% wireless and it is 100% configurable.