



ECOLE CENTRALE DE LILLE

PROJET ORION
TRAVAIL DE L'ÉQUIPE PROJET
CODE PROJET : 21/23-14
RAPPORT

Montre connectée Orion

Élèves :

Mohammed CHADLI
Murilo DEL VECCHIO
Jules DUMEZY
Mathis GUCKERT
Paul GUILLAN
Alexis KRAFT
José Vitor MAKIYAMA
Wassim MEJJAD
Thibaud PICCINALI
Lucas SALAND
Omar SAUBRY
Paul SYLVESTRE
Roman TIÉDREZ

*Représentant du
partenaire :*

Abdelkrim TALBI

Coachs :

Cécile GHOUILA HOURI
Aurélie ROLLE

L'équipe projet atteste que l'ensemble du travail présenté est original et indique en fin de ce rapport chaque source utilisée.

 Mohammed CHADLI  Murilo DEL VECCHIO  Jules DUMEZY  Mathis GUCKERT

 Paul GUILLAN

 Alexis KRAFT

 José Vitor MAKIYAMA

 Wassim MEJJAD

 Thibaud PICCINALI

 Lucas SALAND

 Omar SAUBRY

 Paul SYLVESTRE

 Roman TIÉDREZ

Table des matières

Remerciements	6
Validation du livrable	7
Introduction	8
1 Dimensions du projet	9
1.1 Genèse	9
1.2 Equipe projet	9
1.3 Equipe encadrante	10
1.4 Partenaire	10
1.5 La technologie	11
1.5.1 Description	11
1.5.2 Calcul du champ magnétique	12
1.6 Cahier des charges	13
2 Etat de l'art	15
2.1 Le toucher	15
2.1.1 Définition	15
2.1.2 Mécanorécepteurs	15
2.1.3 La peau	16
2.1.4 Les technologies	17
2.1.5 Les domaines d'application	18
2.2 Les applications pour smartphone	21
2.2.1 Systèmes d'exploitation	21
2.2.2 Technologies utilisées	22
2.3 Les montres connectées	24
2.3.1 Définitions	24
2.3.2 Marché actuel	24
3 Management du projet	27
3.1 Organisation	27
3.1.1 Chronologie	27
3.1.2 Historique	27
3.1.3 Structure définitive	28
3.2 Communication	29
3.3 Gestion de projet	29
3.3.1 Présentation	29
3.3.2 Réunions	30
3.3.3 Documents	30
3.4 Diagrammes	32
3.4.1 Diagrammes d'analyse fonctionnelle	32
3.4.2 Diagrammes SYML	32
3.5 Gestion du budget	33
3.6 Formations et autoformations	34

4 Rapport scientifique : pôle informatique	35
4.1 Introduction	35
4.2 Navigation au sein de l'application	36
4.2.1 Contexte	36
4.2.2 Arborescence d'une application Android	36
4.2.3 Fragments	37
4.2.4 Navigation entre fragments	41
4.2.5 Architecture générale - modèle MVC	43
4.2.6 Résultats	46
4.3 Gestion des autres applications du téléphone	47
4.3.1 Contexte	47
4.3.2 Fonctionnement d'une ListView Android	47
4.3.3 Mise en place des premiers éléments	47
4.3.4 Gestion des modes de vibration	50
4.3.5 Implémentation de la liste d'applications	56
4.3.6 Ajout d'une barre de recherche	61
4.3.7 Résultats	64
4.4 Communication Bluetooth Low Energy	65
4.4.1 Contexte	65
4.4.2 Les permissions nécessaires	65
4.4.3 Détection des appareils BLE à proximité	66
4.4.4 Connexion et déconnexion de la carte à l'appareil Android	68
4.4.5 Echanges entre la carte et l'appareil Android	69
4.4.6 Résultats	71
4.5 Gestion des notifications du téléphone	72
4.5.1 Contexte	72
4.5.2 Service	72
4.5.3 Autorisations requises	72
4.5.4 Récupération et traitement des notifications	73
4.5.5 Résultats	75
4.6 Gestion de l'heure	76
4.6.1 Contexte	76
4.6.2 Le format "143"	76
4.6.3 En pratique	76
4.6.4 Résultats	77
4.7 Explication du fonctionnement de l'application	78
4.7.1 Structure de l'application	78
4.7.2 La <i>toolbar</i>	78
4.7.3 La page <i>Accueil</i>	80
4.7.4 La page <i>Vibrations</i>	80
4.7.5 La page <i>Notifications</i>	81
4.8 Conclusion	82
5 Rapport scientifique : pôle électronique	83
5.1 Introduction	83
5.2 Le circuit électrique	84
5.2.1 Les composants du circuit	84

5.2.2	Simulations et premiers tests	85
5.3	Choix des composants	87
5.4	Réalisation du circuit	88
5.5	Validation du circuit	90
5.6	La carte ESP32	91
5.6.1	Présentation	91
5.6.2	ESP32-C3-DevKitM-1	91
5.6.3	Adafruit QT Py ESP32-C3	92
5.6.4	Programmation - BLE	93
5.6.5	Programmation - Actionneurs	95
5.7	L'anneau LED NeoPixel Ring - 12 x 5050 RGB LED	96
5.7.1	Présentation	96
5.7.2	Compatibilité avec l'ESP32	96
5.7.3	Programmation	97
5.8	L'autonomie de la montre	99
5.9	Pistes d'améliorations	99
5.10	Conclusion	99
6	Rapport scientifique : pôle conception	100
6.1	Introduction	100
6.2	Conception et réalisation du boîtier	101
6.2.1	Contexte	101
6.2.2	Première conception	101
6.2.3	Première fabrication	106
6.2.4	Deuxième fabrication	107
6.2.5	Deuxième conception	108
6.2.6	Troisième fabrication	109
6.3	Conception et réalisation du bracelet	110
6.3.1	Contexte	110
6.3.2	Première conception	110
6.3.3	Première fabrication	113
6.3.4	Deuxième conception	115
6.3.5	Deuxième fabrication	116
6.4	Eléments standards	117
6.4.1	Choix des éléments standards	117
6.4.2	Implémentation	118
6.5	Conception et réalisation de la membrane	119
6.6	Test du premier assemblage	120
6.7	Test du second assemblage	121
6.8	Conclusion	122
7	Prototypage	123
7.1	Premier prototype	123
7.2	Deuxième prototype	123
7.3	Troisième prototype	124

8 Valorisation	125
8.1 Charte graphique	125
8.2 Réseaux sociaux	125
8.3 Affiche et contenu promotionnel	125
8.4 Partager une application	127
8.4.1 GitHub	127
8.4.2 APK	127
8.4.3 Google Play Store	128
9 Suite du projet et améliorations	130
9.1 Pistes d'améliorations de notre projet	130
9.2 Autres idées d'applications de la technologie	130
10 Conclusion	132
11 Bibliographie	135
11.1 Etat de l'art	135
11.2 Pôle informatique	135
11.3 Pôle électronique	136
11.4 Pôle conception	136
12 Annexes	137
12.1 Cahier des charges	137
12.2 Pôle informatique	138
12.3 Pôle électronique	138
12.4 Pôle conception	139
12.5 Management du projet	144
13 Lexique	146

Remerciements

Nous tenions tout d'abord à remercier notre partenaire l'IEMN et ses encadrants pour l'aide et le soutien apportés tout au long de ce projet.

Nous souhaitions notamment remercier Monsieur Abdelkrim TALBI pour avoir encadré et proposé ce projet. Nous voulions également remercier nos coachs : Madame Cécile GHOUILA HOURI et Madame Aurélie ROLLE pour leurs implications, leurs suivis et leurs excellents conseils sur l'ensemble de notre travail.

Nous souhaitions particulièrement remercier l'ensemble de nos consultants :

- Monsieur Yannick DUSCH : pour son aide et son suivi tout au long du projet, et particulièrement pour son implication dans les premières phases de définition du projet et de son cahier des charges.
- Monsieur Thomas BOURDEAUD'HUY : pour avoir accompagné le pôle informatique durant les premières semaines de conception de l'application en enseignant les bases de la programmation Android.
- Monsieur Simon THOMY : pour avoir guidé le pôle électronique sur la réalisation, la validation et la conception de la carte électronique.
- Monsieur Laurent PATROUX : pour avoir assisté le pôle conception sur la réalisation du bracelet et du *casing*, particulièrement pour avoir permis d'accéder à des méthodes de fabrication nouvelles.
- Monsieur Mohand Salah MOUSSA : pour son aide sur la réalisation des membranes de nos actionneurs tant sur le choix des matériaux, de la méthode de fabrication que sur la production.

Nous réservons aussi des remerciements pour toutes les personnes qui se sont impliquées de près ou de loin dans ce projet aussi bien sur son aspect technique que sur l'écriture de ce rapport.

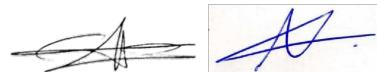
Enfin nous souhaitions remercier l'Ecole Centrale de Lille, pour avoir permis de rendre ce projet possible.

Validation du livrable

Je soussigné, Monsieur Abdelkrim TALBI responsable du projet, atteste avoir validé les différents livrables du projet *Orion*.



Nous soussignées, Madame Cécile GHOUILA HOURI et Madame Aurélie ROLLE, attestons avoir encadré l'ensemble du projet *Orion* et certifions du travail réalisé.



Je soussigné, Monsieur Yannick DUSCH, atteste que les élèves du projet *Orion* ont acquis les notions ayant permis la rédaction d'un cahier des charges et ont su les mettre en œuvre dans le cadre de leur projet.



Je soussigné, Monsieur Thomas BOURDEAUD'HUY, atteste que les élèves Mathis GUCKERT, Thibaud PICCINALI, Lucas SALAND et Roman TIEDREZ ont acquis les notions de conception sur android et ont su les mettre en œuvre dans le cadre de leur projet.



Je soussigné, Monsieur Simon THOMY, atteste que les élèves Mohammed CHADLI, Murilo DEL VECCHIO, Jules DUMEZY, Wassim MEJJAD et Omar SAUBRY ont acquis les notions d'impression d'une carte électronique et de conception d'une carte électronique sur Kicad.



Je soussigné, Monsieur Laurent PATROUX, atteste que les élèves Paul GUILLIAN, Alexis KRAFT et Paul SYLVESTRE ont acquis les notions de conception sur android et ont su les mettre en œuvre dans le cadre de leur projet.



images/signPatrouix..

Je soussigné, Monsieur Mohand Salah MOUSSA, atteste que les élèves du projet *Orion* ont acquis les notions requises pour la conception d'une membrane en élastomère et ont su les mettre en œuvre dans le cadre de leur projet.



Introduction

Fruit de la recherche et du développement, l'innovation technologique est au cœur de notre société. Cependant, toute innovation technologique n'est pas destinée au succès.

Pour qu'une innovation technologique ne tombe pas dans l'oubli, il est important que la technologie soit adoptée par les entreprises. Pour cela, il est nécessaire que les entreprises aient connaissance de l'existence de cette technologie. L'adoption d'une nouvelle technologie est également facilitée par la rédaction d'une documentation riche et complète.

Notre équipe projet a donc cherché à répondre à la problématique de mise en valeur de la technologie proposée par l'IEMN via un projet richement documenté.

Pendant un an et demi, nous avons travaillé sur une montre connectée intégrant les actionneurs de l'IEMN dans son bracelet. Cette montre transmet sous forme de vibration les notifications reçues sur le téléphone portable de l'utilisateur, permettant ainsi à ce dernier de ne pas avoir à vérifier en permanence son smartphone.

Ce rapport présente les aspects généraux, techniques, de gestion de projet et de valorisation de notre projet.

L'ensemble de l'équipe du projet *Orion* vous souhaite une bonne lecture.

1 Dimensions du projet

1.1 Genèse

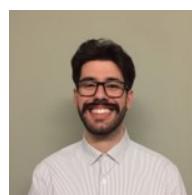
Nous avons tous découvert ce projet lors de la banque à projet organisée à CENTRALE en septembre 2021. Au fil des jours, nous nous sommes rassemblés autour de l'intérêt que nous lui portions, et c'est ainsi, assez naturellement, que s'est formée l'équipe projet. Nous étions particulièrement attirés par l'aspect pluridisciplinaire de ce projet, et la liberté que nous offrait l'IEMN (Institut d'Electronique, de Microélectronique et de Nanotechnologie), quant à l'utilisation de la technologie. Dès le début du projet, nous avons formé un groupe soudé et très motivé, et les différentes ressources provenant d'anciens projets portant sur cette même thématique (*Tact'Icones 3*, *TactiGant*) se sont avérées utiles pour le démarrer efficacement.

1.2 Équipe projet

L'équipe projet est constituée de Mohammed CHADLI, Murilo DEL VECCHIO, Jules DUMEZY, Mathis GUCKERT, Paul GUILLAN, Alexis KRAFT, José Vitor MAKIYAMA (qui nous a rejoint au début du premier semestre 2022/2023), Wassim MEJJAD, Thibaud PICCINALI, Lucas SALAND, Omar SAUBRY, Paul SYLVESTRE et Roman TIÉDREZ.



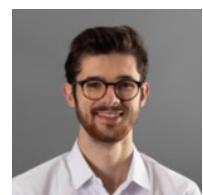
Mohammed CHADLI



Murilo DEL VECCHIO



Jules DUMEZY



Mathis GUCKERT



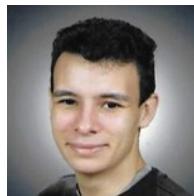
Paul GUILLAN



Alexis KRAFT



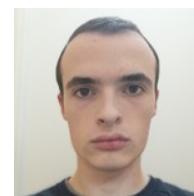
José Vitor MAKIYAMA



Wassim MEJJAD



Thibaud PICCINALI



Lucas SALAND



Omar SAUBRY



Paul SYLVESTRE



Roman TIÉDREZ

Dès le lancement, Jules a été naturellement désigné chef de projet, ayant l'expérience et les compétences nécessaires pour assurer ce rôle. Le second semestre a vu naître la mise en place sous forme de pôles (pôle électronique, pôle mécanique et pôle informatique), ayant chacun son chef de pôle, ce qui fait que nous n'avons pas vraiment ressenti le besoin d'avoir un chef de projet. Enfin, lors du troisième et dernier semestre, c'est Wassim qui a assumé le rôle de chef de projet, permettant d'envisager sa complétion et sa clôture dans les meilleures conditions. En ce qui concerne les comptes-rendus des réunions, nous avons choisi de les rédiger chacun notre tour. Voici donc la répartition globale des rôles de notre équipe projet :

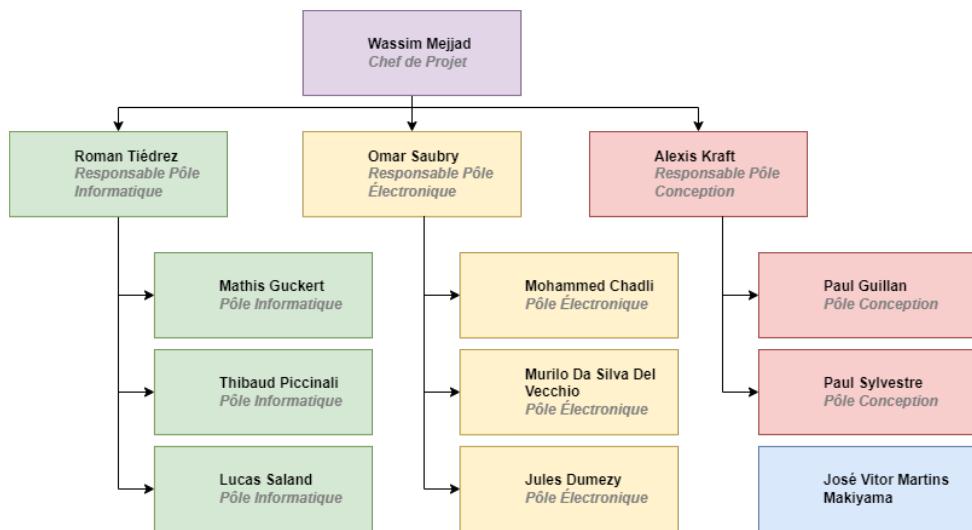


FIGURE 1 – Organigramme de l'équipe projet

1.3 Equipe encadrante

Le projet *Orion* (à l'origine nommé *TactiGant 2.0*), nous a été proposé par Monsieur Abdelkrim TALBI, enseignant-chercheur à Centrale Lille, et travaillant à l'IEMN (Institut d'Électronique, de Microélectronique et de Nanotechnologie). C'est donc à lui que nous nous sommes référés tout au long du projet, que ce soit au sujet de la direction que ce dernier prenait ou à propos de la technologie que nous avons exploitée. Afin de nous suivre et de nous aider à avancer, nous avons également eu la chance d'avoir Madame Cécile GHOUILA HOURI, maître de conférences à CENTRALE LILLE, et Madame AURÉLIE ROLLE, maître de conférences à l'ÉCOLE NATIONALE SUPÉRIEURE DE CHIMIE DE LILLE, en tant que coachs projets.

1.4 Partenaire

L'IEMN, Institut d'Electronique, de Microélectronique et de Nanotechnologie est un institut de recherche CNRS, créé en 1992, et plus précisément une Unité mixte de recherche CNRS. En effet, l'IEMN est un laboratoire partenaire de l'ÉCOLE CENTRALE DE LILLE, de deux universités de la région Nord-Pas de Calais (Lille1 et UHVC), et de l'université catholique de Lille. Son infrastructure principale se situe à Villeneuve-d'Ascq, à proximité de Lille.

Les 22 groupes de recherche de l'IEMN sont répartis dans 5 départements :

1. Matériaux et nanostructures ;
2. Micro et nano-systèmes ;
3. Micro, nano, et optoélectronique ;
4. Circuits et systèmes de communication ;
5. Acoustique.

L'IEMN collabore avec de nombreux acteurs nationaux et internationaux : industriels, laboratoires et universités, avec la volonté de développer des technologies miniaturisées innovantes à forte valeur ajoutée dans des domaines clés. C'est donc dans ce cadre que nous a été proposé ce projet. L'idée initiale était de reprendre le flambeau du projet *TactiGant* (datant de 2017) afin d'aller plus loin avec la technologie de retour tactile de l'IEMN. Celle-ci consiste en une petite matrice d'actionneurs vibrants permettant de reproduire la sensation tactile (détailée dans la suite du rapport). Les échanges avec notre partenaire étaient facilités par les liens entre CENTRALE LILLE et l'IEMN. Cela nous a particulièrement été utile lors de la phase de lancement du projet, afin de partager notre vision et nos idées avec Monsieur Talbi.



FIGURE 2 – Logo et locaux de l'IEMN.

1.5 La technologie

1.5.1 Description

Notre projet repose sur l'utilisation et la valorisation d'une technologie innovante fournie par notre partenaire, l'IEMN. Cette technologie qu'on nommera actionneur est constituée d'un système bobine-aimant-membrane.

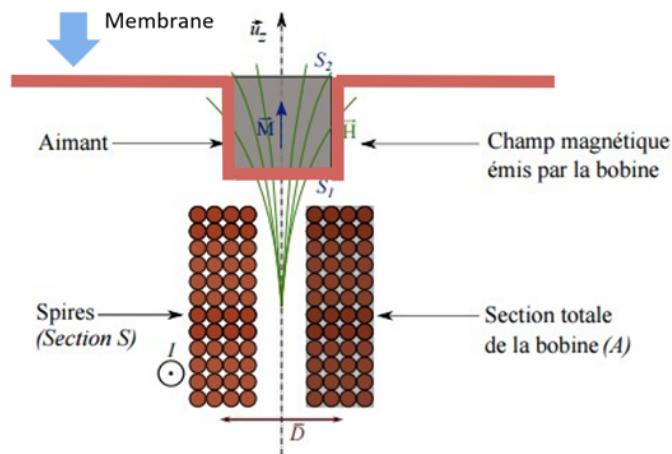


FIGURE 3 – Schéma de l'actionneur de l'IEMN

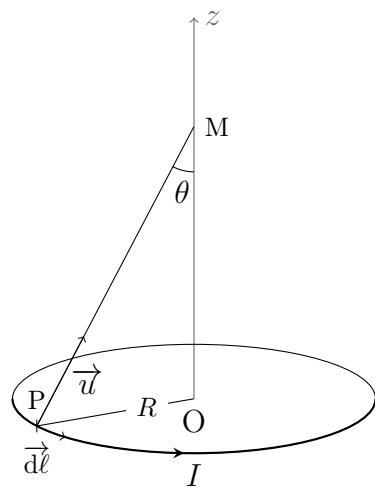
L'idée est que dès lors que la bobine reçoit un courant cette dernière va se mettre à générer un champ magnétique faisant ainsi vibrer l'aimant. En variant sur l'intensité du courant fourni à la bobine on peut faire varier l'intensité du champ magnétique et ainsi proposer des vibrations différentes. En effet, l'alternance d'un courant nul et d'un courant constant entraîne l'apparition puis la disparition du champ magnétique s'exerçant sur l'aimant, et donc une translation verticale (preuve dans la partie 1.5.2) de l'aimant.

1.5.2 Calcul du champ magnétique

Nous allons démontrer dans cette partie que le champ magnétique créé par la bobine est bien vertical et qu'il fait donc bouger l'aimant verticalement.

Pour déterminer le champ magnétique créé par la bobine on considère que la bobine est une superposition de n spires. Ainsi le champ magnétique total sera la superposition de chaque champ de ces n spires.

Notre problème se simplifie donc à l'étude du champ magnétique d'une spire. On adoptera les notations suivantes :



D'après la loi de Biot et Savart, on sait que lorsque la perméabilité μ_0 d'un milieu est proche de 1, on peut exprimer le champ magnétique exercé par tout point P d'un fil en tout point M de l'espace de la manière suivante :

$$d\vec{B}_P(M) = \frac{\mu_0 I}{4\pi} \frac{d\vec{l} \wedge \vec{u}_{P \rightarrow M}}{PM^2}$$

Ainsi on a pour toute la spire :

$$\begin{aligned} \Rightarrow \vec{B} &= \frac{\mu_0}{4\pi} \oint_{spire} \frac{I d\vec{l} \wedge \vec{PM}}{PM^3} \\ \Leftrightarrow \vec{B} &= \frac{\mu_0 I}{4\pi} \oint_{spire} \frac{d\vec{l} \wedge (\vec{PO} + \vec{OM})}{PM^3} \\ \Leftrightarrow \vec{B} &= \frac{\mu_0 I}{4\pi} \left(\oint_{spire} \frac{d\vec{l} \wedge \vec{PO}}{PM^3} + \oint_{spire} \frac{d\vec{l} \wedge \vec{OM}}{PM^3} \right) \\ \Leftrightarrow \vec{B} &= \frac{\mu_0 I}{4\pi} \left(\oint_{spire} \frac{d\vec{l} \wedge \vec{PO}}{PM^3} - \frac{\vec{OM}}{PM^3} \wedge \oint_{spire} d\vec{l} \right) \\ \Leftrightarrow \vec{B} &= \frac{\mu_0 I}{4\pi} \left(\oint_{spire} \frac{d\vec{l} \wedge \vec{PO}}{PM^3} - \frac{\vec{OM}}{PM^3} \wedge \int_0^{2\pi} R d\theta \vec{e}_\theta \right) \\ \Leftrightarrow \vec{B} &= \frac{\mu_0 I}{4\pi} \left(\oint_{spire} \frac{d\vec{l} \wedge \vec{PO}}{PM^3} - \frac{\vec{OM}}{PM^3} \wedge R \int_0^{2\pi} (\cos \theta \vec{e}_x + \sin \theta \vec{e}_y) d\theta \right) \\ \Leftrightarrow \vec{B} &= \frac{\mu_0 I}{4\pi} \oint_{spire} \frac{d\vec{l} \wedge \vec{PO}}{PM^3} \text{ par } 2\pi\text{-périodicité des fonctions cos et sin} \\ \Leftrightarrow \vec{B} &= \frac{\mu_0 I}{4\pi} \oint_{spire} \frac{d\theta \vec{e}_\theta \wedge (-R \vec{e}_r)}{PM^3} \\ \Leftrightarrow \vec{B} &= \frac{\mu_0 I}{4\pi} \oint_{spire} \frac{d\theta R^2 \vec{e}_z}{PM^3} \\ \Leftrightarrow \vec{B} &= \frac{\mu_0 I R^2}{4\pi PM^3} \left(\int_0^{2\pi} d\theta \right) \vec{e}_z \\ \Leftrightarrow \vec{B} &= \frac{\mu_0 I R^2}{2PM^3} \vec{e}_z \\ \Leftrightarrow \vec{B} &= \frac{\mu_0 I R^3}{2RPM^3} \vec{e}_z \\ \Leftrightarrow \vec{B} &= \frac{\mu_0 I}{2R} \sin^3 \theta \vec{e}_z \text{ avec } \sin \theta = \frac{R}{PM} \end{aligned}$$

D'où le résultat :
$$\boxed{\vec{B} = \frac{\mu_0 I}{2R} \sin^3 \theta \vec{e}_z}$$

Nous venons ainsi de déterminer l'expression du champ magnétique créé par la bobine. On constate qu'il est bien vertical.

1.6 Cahier des charges

Dès le début du projet, nous avons rapidement commencé à rédiger un cahier des charges fonctionnel, d'une part, afin de concrétiser nos idées, et d'autre part, afin que notre partenaire soit en mesure de comprendre et de valider la direction que le projet prenait. Cette partie décrit donc les fonctions principales du projet, d'après le cahier des charges situé en annexe. Ce cahier des charges est composé de deux types de fonctions : les fonctions principales (FP) et les fonctions contraintes (FC). Un critère et des niveaux décrivent précisément ce à quoi chaque fonction fait référence, et l'indice de flexibilité qualifie notre capacité à ne pas respecter strictement la fonction concernée.

Le projet Orion consiste en la création d'une montre connecté utilisant la technologie fournie par l'IEMN, décrite dans la partie précédente, sous formes de 4 actionneurs réparties uniformément sur le bracelet. Cette montre doit être capable de se connecter de manière sans fil au téléphone de l'utilisateur grâce à une application et grâce au protocole Bluetooth Low Energy (BLE). La montre doit vibrer instantanément lorsqu'une notification est reçue sur le smartphone de l'utilisateur. Ce retour haptique pourra être personnalisable pour chaque application (plusieurs modes de vibration doivent être définis), directement via l'application. Ces modes doivent être facilement distinguables (par exemple avec des variations spatiales des vibrations, mais aussi les changements de fréquence et d'intensité) tout cela sans déranger l'utilisateur. Concernant le bracelet en lui-même, celui-ci doit être moderne, ergonomique, peu coûteux et respectueux de l'environnement, notamment en ce qui concerne les matériaux utilisés et sa faible consommation d'énergie. Il doit également avoir la possibilité d'afficher l'heure, mais également l'état de connexion du bracelet. Enfin, un bouton On/Off permettra à l'utilisateur d'éteindre et d'allumer le bracelet manuellement.

Enfin, pour ce qui est de l'application en elle-même, celle-ci doit être compatible avec tout appareils, mais également posséder une interface utilisateur efficace et design qui permettra d'exploiter au mieux le montre et sa technologie. Elle communiquera directement grâce au BLE à la carte électronique du bracelet, et sa programmation optimisée devra permettre de réduire au mieux la latence avec cette dernière.

D'autres fonctions ont été intégrées au cahier des charges mais n'ont pas été réalisées (par manque de temps ou de moyens), comme par exemple, la capacité du bracelet à produire différentes vibrations selon les fréquences et le tempo d'un morceau joué sur le téléphone. D'autres ont été intégrées mais pas complètement (par exemple la compatibilité pour tout type de smartphone aura été réduite à une compatibilité pour ton type d'appareil Android).

Carte de commande et énergie				
FP6	Transmettre aux actionneurs un signal délivré par l'application de manière instantanée	Temps de réponse	t5% < 200ms entre l'application et les actionneurs	1 ▼
FP7	Communication avec les actionneurs conçus par l'IEMN	Puissance et nombre de sorties suffisantes	- Puissance de 0,3125W par actionneur (à discuter) - au moins N sorties (N à définir)	1 ▼
FP8	Communication avec l'application dans des environnements variés	Utiliser une technologie sans-fil (Bluetooth/WiFi)	Puissance suffisante pour fonctionner à travers le tissu ou une cloison fine	1 ▼
FP9	Recharge facile	Utilisation d'une batterie et d'un port de chargement classique (USB)		▼
FP10	Comfort et simplicité			▼
FP11	Ne pas être trop complexe	Simplicité de développement (outils adaptés à des étudiants sans expérience)		▼
FP12	Respect de l'environnement	Matériaux et consommation d'énergie		▼
FC10.1	L'ensemble carte électronique + batterie ne doit pas être trop volumineux	Dimensions	Volume de moins de 15cm3	2 ▼
FC10.2	Le système doit avoir une autonomie suffisante	Autonomie	Autonomie d'au moins 6h	3 ▼
FC10.3	Le système ne doit pas être dangereux	Intensité du courant et force des actionneurs	Intensité de moins de 100mA et contact limité à l'électronique	3 ▼
FC11.1	Le système doit pouvoir être facilement débogué	Interface de débogage présente		▼
FC11.2	Le système ne doit pas être trop coûteux	Prix	Prix d'une unité à moins de 50€ en matières premières	3 ▼
FC11.3	Le système doit être rapidement démontable	Temps de démontage	Démontage en moins d'une minute	3 ▼
FC11.4	Le système doit avoir une durée de vie suffisante	Obsolescence	Système fonctionnant pendant au moins 2 ans	1 ▼

FIGURE 4 – Extrait du Cahier des Charges Fonctionnel du projet. Retrouvez le cahier des charges complet en annexes

2 Etat de l'art

L'état de l'art est une étape cruciale dans la réalisation d'un projet. Sa réalisation permet de comprendre les avancées et les limites de la recherche existante et ainsi de définir les objectifs et les hypothèses du projet de manière plus précise. En somme, l'état de l'art est un prérequis essentiel pour assurer la qualité et la pertinence d'un projet.

Nous allons donc dans cette partie faire un état de l'art sur les technologies tactiles ; en détaillant leur domaines d'applications ; sur les applications pour smartphone et enfin, sur les montres connectées.

2.1 Le toucher

2.1.1 Définition

Le **sens du toucher** appartient à la **somesthésie** : l'ensemble des sensations (température, douleur, contact physique, étirement d'un muscle...) qui ne sont apparentées ni au goût, ni à l'odorat, ni à l'ouïe, ni à la vue. Le toucher désigne alors la fraction des sensations somesthésiques acquises par l'intermédiaire de la peau. La discipline qui étudie ce sens se nomme **l'haptique**.

On répartit les sensations tactiles selon trois catégories :

- **contact de pression** (ou "grossier") qui intervient quand on serre quelque chose, quand on pousse quelque chose...
- **contact léger** (ou encore "de déchiffrage" ou "fin"), qui intervient dans la lecture du braille, l'appréciation d'une matière...
- **vibrations** : l'organisme humain est capable de ressentir des variations de pressions lorsque la fréquence de ces variations est comprise entre 30 et 1500 Hz.

Le projet *Orion* s'intéresse essentiellement à cette dernière catégorie.

2.1.2 Mécanorécepteurs

Il existe six grands groupes de récepteurs sensoriels (neurones) du toucher, appelés **mécanorécepteurs**. On y distingue les **récepteurs toniques**, qui évaluent les variations de pression, des **récepteurs phasiques**, à "vitesse de réaction" plus lente et qui mesurent la durée du contact (le ressenti des vibrations est donc entièrement dû aux récepteurs toniques) :

- **les corpuscules (ou disques) de Merkel** (toniques), spécialisés dans la perception de la forme et de la texture
- **les corpuscules de Meissner** (phasiques), spécialisés dans la détection du mouvement et de la saisie
- **les plaques du toucher**, aussi appelés "disques tactiles" (toniques)
- **les récepteurs du follicule pileux** (phasiques)
- **les corpuscules de Paccini** (phasiques), qui permettent d'être conscients de la forme des membres et guident ainsi les déplacements
- **les corpuscules (ou terminaisons) de Ruffini** (toniques), spécialisés dans la détection des vibrations rapides

Des **nocicepteurs** et **thermorécepteurs** existent aussi et servent respectivement à évaluer la douleur et la température.

"Un **champ récepteur** est une partie anatomique d'une surface sensorielle (rétine, peau)

qui, lorsqu'elle est soumise à un stimulus, modifie le potentiel de membrane d'un neurone." (vetopsy.fr)

Plus les champs récepteurs d'une partie du corps donnée sont étroits, meilleure est la "précision" des sensations. L'homme a ainsi des champs récepteurs de 2mm environ sur les doigts et de 42mm sur les jambes où la discrimination entre différents stimuli est beaucoup plus faible.

2.1.3 La peau

La peau humaine est composée de deux couches principales :

- **L'épiderme** est la couche externe, la plus mince. Les disques de Merkel et tactiles, ainsi que les récepteurs des follicules pileux se situent dans l'épiderme.
- **Le derme** est responsable de la texture de la peau car riche en vaisseaux sanguins et en terminaisons nerveuses. Les corpuscules de Meissner et Ruffini s'y trouvent.

Les corpuscules de Paccini sont situés dans l'hypoderme, soit immédiatement en-dessous du derme.

On peut alors distinguer deux grands types de peau, en fonction de leur pilosité et des mécanorécepteurs présents :

- **La peau glabre** (dépourvue de poils), présente principalement sur la paume des mains et la plante des pied, et dont les récepteurs principaux sont les disques de Merkel et les corpuscules de Meissner qui possèdent des champs récepteurs restreints.
- **La peau velue**, dont les récepteurs principaux sont ceux des follicules pileux et les plaques de toucher.

Les corpuscules de Ruffini et de Pacini, aux champs récepteurs étendus, sont présents dans les deux types de peau.

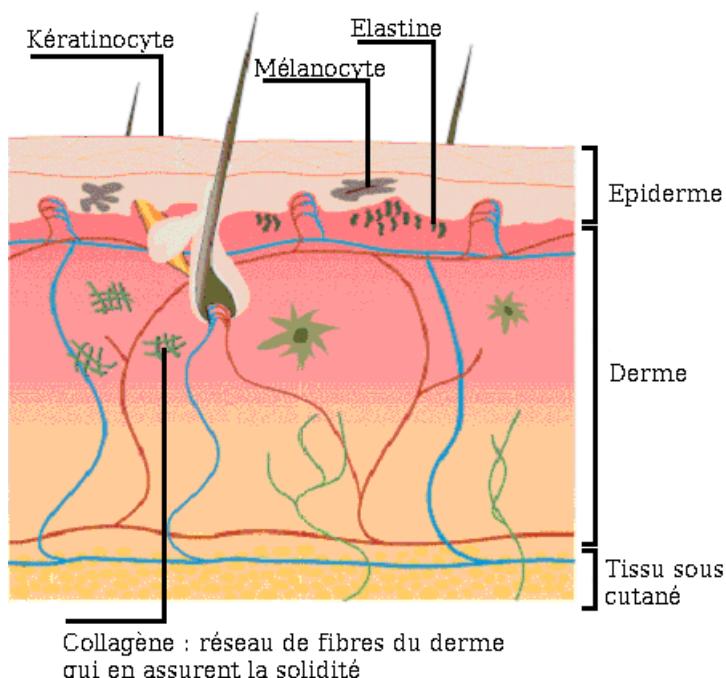
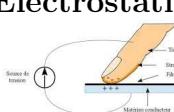
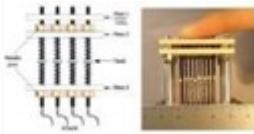
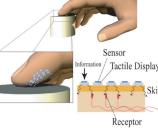


FIGURE 5 – Schéma de la peau

2.1.4 Les technologies

Les retours tactiles, haptique, ou de force sont des mécanismes qui permettent de restituer à l'utilisateur une sensation de force ou de texture lorsqu'il interagit avec un dispositif ou une interface. Ils peuvent être utilisés pour renforcer le réalisme des interactions virtuelles, pour aider à la navigation, pour renforcer la sécurité de certaines actions...

Plusieurs technologies permettent aujourd'hui d'offrir ces types de retour. Détailons dans cette partie un comparatif des différentes technologies existantes :

Technologie	Principe	Avantages	Inconvénients
Électrostatique 	Le doigt repose sur un plateau mobile qui bouge grâce à des électrodes	- Peut procurer des sensations de toucher légères	- Peut nécessiter des champs électrostatiques élevés
Magnétostatique 	Vibrations créées à partir d'un aimant et d'une bobine	- Bonne fréquence - Facile à alimenter	- Intensité peu importante
Polymères thermoactifs 	Contraction et décontraction de fils sous l'influence de la température	- Intensité importante	- Commande par température peu pratique - Faible fréquence
Électrocutané 	Stimulation directe des différents récepteurs sensoriels de la peau par le biais d'électrodes placées au contact	- Permet de simuler des sensations de toucher plus réalistes	- Coût important
Piézoélectrique 	Changement du champ électrique provoquant la contraction et l'extension de cristaux	- Force importante - Bonne fréquence - Grand déplacement	- Tension élevée - Encombrement important
Pneumatique 	Changement de pression d'un gaz provoquant le gonflement et dégonflement d'un matériau	- Grand déplacement	- Faible fréquence - Encombrement important

2.1.5 Les domaines d'application

Les domaines d'application des technologies de retour tactile, de force ou haptique sont nombreux. Nous allons dans cette partie vous en présenter quatre d'entre eux.

→ Application dans le domaine médical :

Les technologies de retour haptique sont particulièrement utilisées dans le domaine de la médecine, notamment en ce qui concerne la simulation médicale. On pense par exemple au bras *Mediseus Surgical Drilling Simulator* qui offre un ressenti très proche de la réalité à son utilisateur lui permettant ainsi d'effectuer, par exemple, des opérations à distance.



FIGURE 6 – Le *Mediseus Surgical Drilling Simulator*.

Plus récemment, les technologies de retour haptique apportent également de nouvelles approches dans l'enseignement. Grâce à un dispositif comme le *Desktop 6D*, il est possible de réaliser une opération pratique sans utiliser de mannequin ou de cadavre.



FIGURE 7 – Le *Desktop 6D*.

→ Application dans le domaine du handicap :

L'ensemble de ces technologies permet aussi d'apporter des solutions à des personnes atteintes d'un handicap. Les afficheurs brailles par exemple, sont une solution pour les personnes aveugles ou malvoyantes. Dotées d'un retour tactile, ces solutions peuvent permettre de traduire n'importe quel livre en braille :



FIGURE 8 – L'*ADVANCED DISPLAYS for the BLIND*.

D'autres périphériques, tel que des souris à retour tactile, permettent de rendre plus accessible l'utilisation d'un ordinateur :



FIGURE 9 – Exemple de couplage des dispositifs de pointage avec le mode tactile

→ Application dans le domaine militaire :

Le domaine militaire est également un important champ d'application. De manière similaire au domaine médical, les technologies de retour tactile, haptique ou de force permettent de concevoir des équipements utiles pour la formation des soldats. Par exemple, le projet *TRACER*, qui comprend des armes factices avec retour de force couplées à des casques à réalité augmentée, permettent aux marines d'avoir accès à des scénarios d'entraînement plus réalistes et dynamiques.



FIGURE 10 – Illustration du projet *TRACER*

Les robots militaires récemment développés par l'armée britannique permettent aux soldats de ressentir des sensations tactiles lors de leur utilisation. Ces sensations peuvent être utiles lors d'actions telles que la désactivation de bombes :



FIGURE 11 – Robot désamorceur de bombes

→ Application dans le domaine ludique :

L'ensemble de ces technologies peut également être utilisé à titre plus "récréatif". Il est par exemple possible de modéliser des objets virtuels. Ainsi le stylet *PHANTOM* couplé au logiciel *inTouch* permettent de sculpter des objets grâce à un retour de force.



FIGURE 12 – Utilisation du stylet *PHANTOM* avec le logiciel *inTouch*

Enfin, les technologies tactiles sont aussi utilisées dans le domaine du jeu vidéo. Il existe des projets de gants tactiles (comme *HaptX*, ou comme nos prédecesseurs : *TactiGant*) couplés avec des casques à réalité virtuelle (à l'instar du projet *TRACER*) qui proposent une meilleure expérience de jeu en offrant une meilleure immersion.



FIGURE 13 – Illustration du projet *TactiGant*

2.2 Les applications pour smartphone

Les smartphones ont pris une place importante dans nos sociétés. Un des intérêts du smartphone est l'accès aux applications mobiles. Avant qu'une application soit accessible aux utilisateurs, il faut que celle-ci soit conçue et développée. Il est donc intéressant de faire un tour d'horizon du développement des applications mobiles.

2.2.1 Systèmes d'exploitation

Il existe de nombreux systèmes d'exploitation pour les smartphones mais seulement deux d'entre eux dominent le marché actuel des téléphones mobiles : Android développé par Google et iOS développé par Apple. Android est le leader en matière de système d'exploitation mobile depuis plus de 10 ans.



FIGURE 14 – Parts de marché des systèmes d'exploitation sur mobile en 2023

De 2009 à 2018, Android a connu une forte croissance en parts de marché avant de se stabiliser tandis qu'iOS a évolué de manière plus stable au cours des années.

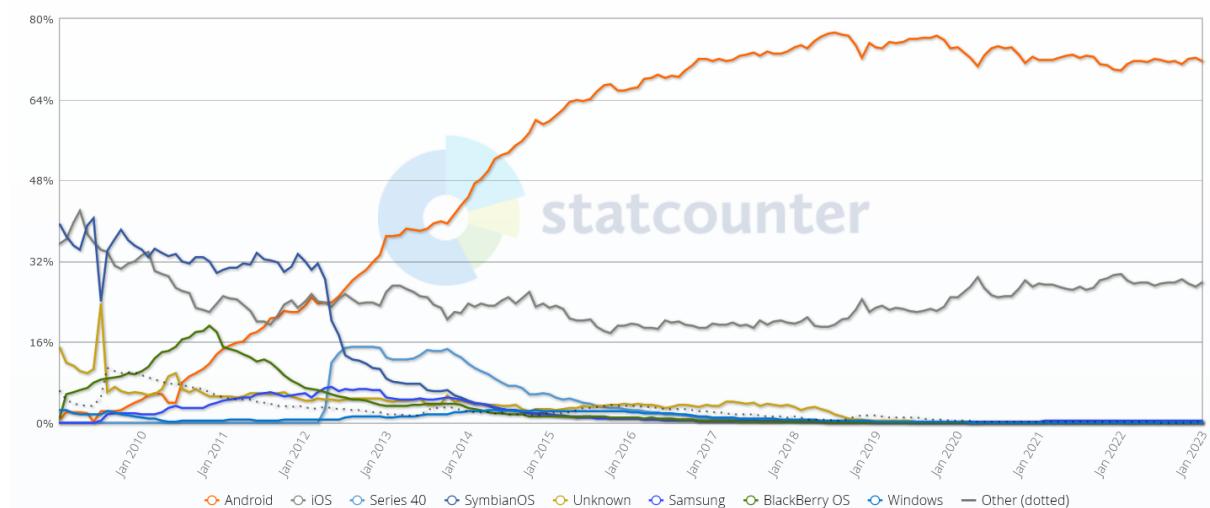


FIGURE 15 – Évolution des systèmes d'exploitation mobile à travers le monde

Les deux systèmes d'exploitation ont deux philosophies différentes. En effet, iOS est développé par Apple pour être utilisé sur les iPhone également produit par Apple. Leur système d'exploitation est conçu pour s'intégrer à l'écosystème créé par Apple. Des fonctionnalités comme airdrop permettant le partage de fichier via bluetooth entre appareils Apple illustre cette volonté de créer un environnement propre à Apple. Tandis qu'Android est prévu pour fonctionner sur de nombreux smartphones différents. Android étant *open source*, les constructeurs peuvent créer leur propre version d'Android appelée surcouche. Il est ainsi fréquent de voir l'apparence de l'interface varier entre deux téléphones Android.

issus de deux constructeurs différents.

Voici un tableau comparatif entre Android et iOS :

	Android	iOS
Accès au code source	<i>open source</i> , les constructeurs créent leur propre version d'Android	Logiciel propriétaire, certaines parties de l'OS sont <i>open source</i> sous la licence Apple Public Source License
Interface utilisateur	Change en fonction du constructeur	Simple et propre
Personnalisation	L'accès au code source permet de personnaliser en fonction des besoins	Limitée
Magasins d'applications	Google Play Store - plus de 3 500 000 applications - possibilité de télécharger des applications Android via d'autres stores comme Amazon et Aptoide	App Store - plus de 1 600 000 applications - Apple vérifie chaque application avant de la publier dans l'App Store
Sécurité	Moins sécurisé car le code source est accessible à tous	Plus sécurisé car l'accès au code source d'iOS est bloqué

2.2.2 Technologies utilisées

Il existe trois principaux types d'applications :

- Natif
- Cross-platform
- Hybride

Les applications natives utilisent les outils de développement de leur système d'exploitation. Ainsi, les applications Android sont développées sur Android Studio en Java ou en Kotlin tandis que les applications iOS sont développées sur xCode en Swift ou Objective-C.

Les applications cross-platform sont issues d'un même code source qui une fois compilé produit deux applications natives : une pour Android et une pour iOS. Les applications cross-platform sont développées avec des *frameworks* comme Xamarin ou .NET MAUI (successeur de Xamarin) en C#, React Native en Javascript ou encore Flutter en Dart.

Enfin, les applications hybrides sont similaires aux applications cross-platform. Un seul code source permet de générer deux applications pour Android et iOS. Ces applications sont développées en utilisant les langages web telles que Javascript, HTML et CSS. Ces applications sont exécutées dans un "conteneur natif" : une webview (une page internet).

Les spécificités de chaque type d'application sont résumées dans le tableau ci-dessous :

Type	Natif	Cross-platform	Hybride
Debugging	Utilise les outils de débogage natifs (log cat sous Android studio par exemple)	Dépend du framework utilisé	Utilisation des outils de débogage natifs et de développement web
Codebase	Un code source par plateforme (un pour Android et un pour iOS)	Un seul code source pour différentes plateformes	Un seul code source pour différentes plateformes
Avantages	Bonnes performances, Accès total aux fonctionnalités et aux équipements, l'UI de l'application est mise à jour avec les mises à jour du système d'exploitation	L'apparence et le feeling de l'UI très proche du natif	Support de plusieurs plateformes
Inconvénients	Le code source produit une application pour une seule plateforme, Le code n'est pas réutilisable.	Performances inférieures aux applications natives	Performances inférieures aux applications natives
Dépendance	Dépend moins sur les librairie open-source	Repose beaucoup sur des librairies et outils	Repose beaucoup sur des librairies et outils
Moteur de rendu	Natif	Natif	Browser
Outils (logiciels et frameworks)	xCode, Android Studio	React Native, Flutter, Xamarin, .NET MAUI	Apache Cordova, Visual Studio
Langages de programmation	Java, Kotlin, Objective-C, Swift	C#, Dart, JavaScript, HTML, CSS	JavaScript, HTML, CSS
Coûts	Coûts de développement plus élevés	Coûts de développement réduits	Coûts de développement réduits
Temps avant mise sur marché	Long car il faut développer deux applications	Gain de temps car le code est commun à Android et iOS	Gain de temps car le code est commun à Android et iOS

2.3 Les montres connectées

2.3.1 Définitions

Une **montre intelligente**, *smartwatch* ou encore **montre connectée**, est un dispositif porté en bracelet dont les fonctionnalités sont supérieures à celles d'une montre "classique" grâce à l'utilisation de l'informatique. Les termes "montre connectée" font référence à un type de montre intelligente particulier utilisant une technologie sans-fil, bien que les trois dénominations soient généralement utilisées de manière interchangeable. L'archétype actuel de la montre intelligente est une montre connectée à écran tactile. Elles peuvent être réparties en deux catégories principales :

- les montres de sport
- les "smartphones au poignet"

Les **montres de sport** sont des moniteurs d'activité physique intelligents. Elles sont pour cela généralement munies de cardiofréquencemètres et d'accéléromètres, et éventuellement d'un dispositif GPS. En plus de la mesure du rythme cardiaque, les fonctionnalités les plus classiques sont la podométrie, la quantification de l'effort physique en nombre de calories brûlées et une évaluation de la qualité du sommeil. Les modèles récents fonctionnent de paire avec une application pour smartphone qui permet de visualiser un résumé des mesures effectuées. Cette application reçoit en règle générale les données par Bluetooth. Les modèles plus évolués peuvent proposer des programmes d'exercices et afficher en temps réel leur progression.

Les "**smartphones au poignet**" permettent à l'utilisateur d'utiliser les fonctions essentielles de son smartphone (appels, SMS, alarmes, ...) sans avoir les mains encombrées par ledit smartphone. Un intérêt tout particulier de ces appareils est de pouvoir être immédiatement averti de ses notifications sans avoir à regarder son téléphone et sans nécessairement utiliser de technique "bruyante" comme un vibreur ou une sonnerie. En outre, plusieurs constructeurs y intègrent des services permettant le paiement sans contact en présentant simplement la montre. De nombreux "smartphones au poignet" intègrent les fonctionnalités caractéristiques des montres de sport et encourager les utilisateurs à partager leurs performances fait partie intégrante de la stratégie publicitaire des constructeurs.

2.3.2 Marché actuel

Le marché des montres connectées est en hausse significative. Par exemple, les ventes ont augmenté de 13% entre le premier trimestre 2021 et le premier trimestre 2022. Les paragraphes suivants présentent quatre de ses plus grands acteurs.

Ce marché est dominé par **Apple** et son *Apple Watch* (36% de part de marché au premier trimestre 2022). Conformément à la stratégie principale de la firme, cette montre s'insère de manière naturelle dans "l'écosystème Apple" et se veut aussi bien une extension qu'une version poignet de l'iPhone. La marque revendique un dispositif léger et le bracelet "respirant" fait partie intégrante du produit. Certains modèles remplacent intégralement le smartphone et fonctionnent sans avoir besoin de l'avoir à proximité. Outre les fonctionnalités propres à tout téléphone, Apple met en avant la mesure du taux d'oxygène dans le sang, un électrocardiogramme, un thermomètre ou encore le suivi du cycle menstruel.



FIGURE 16 – *Apple Watch Series 8* ©Apple

Le premier rival d’Apple est **Samsung** et sa *Galaxy Watch*. S’il existe un grand nombre de modèles différents, Samsung présente surtout ses montres comme des accessoires de sport. Par exemple, la *Galaxy Watch5 Pro* est capable de mesurer l’impédance bioélectrique, la fréquence cardiaque, la tension artérielle. Elle résiste à l’eau et son autonomie et sa robustesse sont particulièrement mises en avant. Enfin, la quantité d’applications utilisables directement depuis la montre est présentée comme un atout important.



FIGURE 17 – *Galaxy Watch5 Pro* ©Samsung

D’autres fabricants de smartphones comme Xiaomi, Huawei développent leurs propres montres connectées de ce type.

Si le marché des montres de sport est plus équilibré, il admet deux acteurs principaux.

Fitbit (rachetée par Google en 2019) se consacre au développement de "coachs électroniques". Outre une mesure "extrêmement précise" de la fréquence cardiaque est vendue une aide à la gestion du stress. L’aspect "smartphone au poignet" existe dans une certaine mesure sur certains modèles qui permettent de lire les SMS par l’intermédiaire d’une connexion 4G, entre autres. Un argument de vente de poids est le catalogue d’applications développées par Google pour ses montres.



FIGURE 18 – *Google Pixel Watch* ©Google

Garmin produit des montres intelligentes majoritairement axées sur le sport. De nombreux modèles sont disponibles selon les fonctionnalités recherchées et le budget. La marque accorde une grande importance au design de ses montres. Par exemple, la *Forerunner® 955* présente toutes les fonctionnalités mentionnées dans le paragraphe de présentation des montres de sport, avec des systèmes exhaustifs d'analyse des performances, un coach virtuel et des cartes détaillées. La lecture de musique ou bien de notifications sont possibles une fois la montre synchronisée avec un service dédié.



FIGURE 19 – *Garmin Forerunner® 955* ©Garmin

3 Management du projet

3.1 Organisation

3.1.1 Chronologie

Le déroulement du projet peut être réparti en trois grandes phases :

- *Définition (Lancement-Printemps 2022)*
- *Travail technique (Printemps 2022-Automne 2022)*
- *Finalisation (Automne 2022-Clôture)*

La phase de *Définition* correspond à la recherche d'objectifs formels pour le projet. Les tâches sont principalement de la gestion de projet et la constitution d'un état de l'art.

La phase de *Travail technique* représente toute la phase de formation puis de développement concret du produit. Les réunions sont beaucoup plus rares et chaque pôle travaille de manière indépendante des deux autres.

Enfin, la phase de *Finalisation* équivaut à la réunion des travaux de chaque pôle en vue d'obtenir des prototypes fonctionnels puis un produit final à la hauteur des attentes du client.

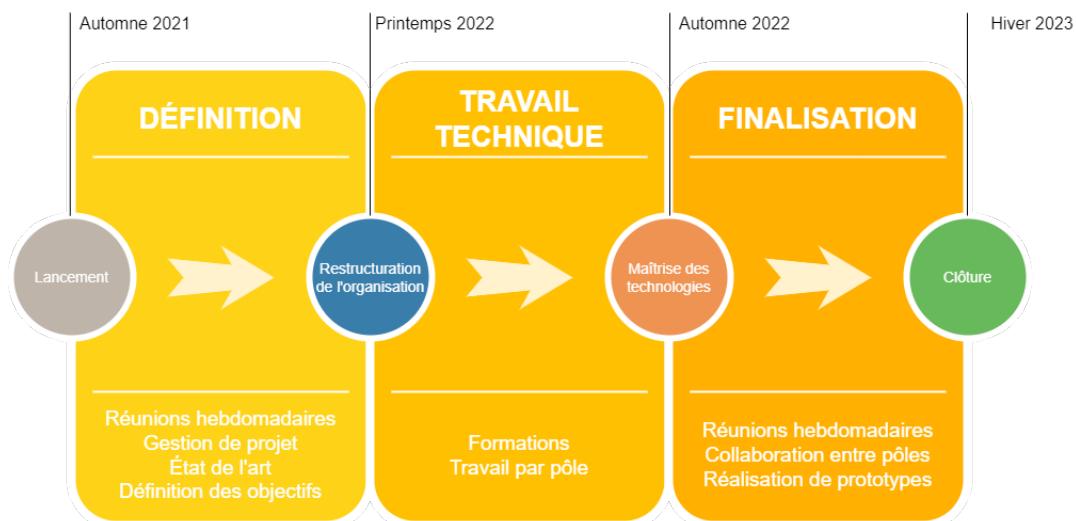


FIGURE 20 – Chronologie du déroulé du projet

3.1.2 Historique

Au début du projet, lorsque l'objectif de celui-ci était encore vaguement défini, il n'existe quasiment aucune organisation formelle. Jules Dumezy avait été désigné chef de projet et à ce titre se chargeait d'organiser les réunions et était plus ou moins seule force de décision lors des réunions. Les comptes-rendus de réunion étaient rédigés tour-à-tour par les membres de l'équipe. Mathis Guckert se proposa pour devenir responsable administratif et gérer la communication avec les encadrants au début de l'hiver 2022. Les tâches étaient attribuées sur la base du volontariat et leurs avancées directement rapportées au chef de projet.

Cette organisation fut entièrement mise à niveau au printemps 2022. Lorsque l'objectif du projet fut clairement défini, une répartition en trois pôles -*Informatique*, *Électronique*

et *Conception-* parut optimale. La répartition s'est faite en fonction des compétences de chacun et de leurs envies. Un responsable fut élu par pôle. Wassim Mejjad devint le nouveau chef de projet. La position de responsable administratif fut absorbée par le chef de projet.

Certaines positions plus informelles ont existé de manière temporaire du début à la fin du projet. Par exemple, un "Pôle Membrane" avec un "Responsable Membrane", ou encore une "Équipe communication" ont chacun existé pendant une période de quelques semaines à quelques mois et réunissaient des collaborateurs en avance sur les tâches de leurs pôles respectifs et désignés par le chef de projet pour se concentrer sur une tâche annexe.

3.1.3 Structure définitive

Dans la structure finale, les membres de l'équipe sont répartis au sein d'une hiérarchie pyramidale classique qui reflète la multidisciplinarité du projet :

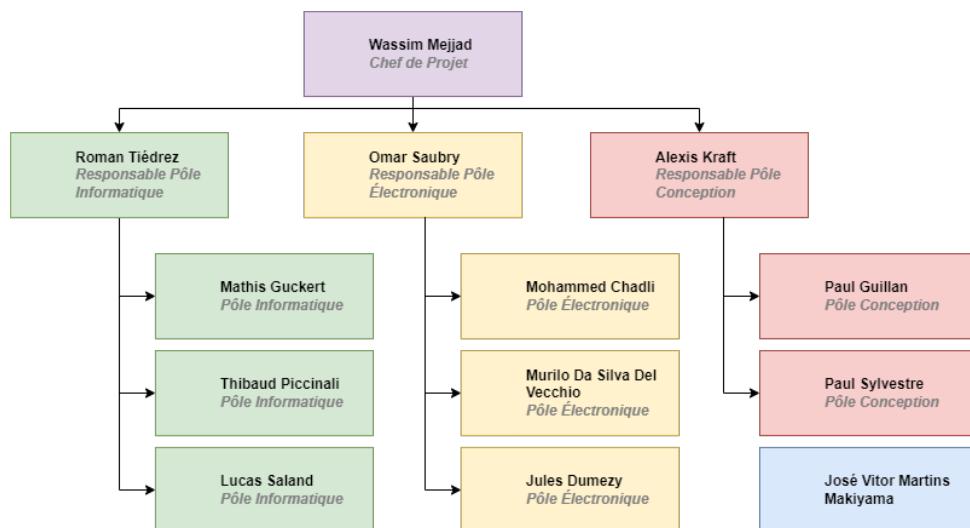


FIGURE 21 – Organigramme de l'équipe projet

Le **chef de projet** assume plusieurs rôles. Il est d'abord celui qui se charge d'organiser et d'animer les réunions rassemblant toute l'équipe, ainsi que les audits. Il rédige les comptes-rendus de réunion et se charge aussi de la communication entre l'équipe et les encadrants du projet. Il assure enfin la mise à jour des documents de gestion de projet. Il n'intervient quasiment pas sur l'aspect technique du projet.

Les responsabilités au niveau de l'aspect technique du projet incombent aux **responsables de pôle**. Ce sont eux qui prennent les grandes décisions et fixent les objectifs quant aux moyens de créer concrètement le produit final. Le choix des outils de travail et des procédés utilisés leur revient. Ils déterminent les jalons et sont libres d'organiser des réunions avec les membres de leur pôle pour assurer le bon déroulement du projet. Le responsable de pôle est un membre à part entière de son propre pôle et à ce titre intervient autant que les autres sur les tâches techniques. Il rapporte ses avancées sous forme vulgarisée au chef de projet.

Les **autres membres** de l'équipe sont chargés d'effectuer les tâches déléguées par le responsable de leur pôle. En pratique, une grande autonomie leur est laissée au niveau du

planning et leur opinion compte beaucoup dans les décisions prises par le responsable de pôle.

Notons que la multidisciplinarité du projet est responsable d'une grande transversalité dans les tâches, et chacun est régulièrement amené à collaborer étroitement avec des membres d'un autre pôle. Dans ce cas, la tâche en question est associée à un des pôles concernés et placée sous la responsabilité du chef de ce pôle.

3.2 Communication

Avec une telle transversalité, se tenir au courant des avancées de chacun est primordial.

L'application *Messenger* est l'outil principal utilisé par le chef de projet pour communiquer les dates de réunion à l'équipe. De manière plus informelle, l'application est parfois utilisée par tout membre de l'équipe qui souhaite attirer l'attention de tous les autres membres sur un point précis et de manière immédiate.

Discord est fortement utilisé pour le travail sur l'aspect technique du projet. Le *chat* permet d'échanger facilement et rapidement sur des détails techniques, images à l'appui si nécessaire. La répartition en *salons* permet à la fois à chaque pôle d'avoir son espace de travail virtuel dédié et de se tenir au courant des avancées des autres pôles. Les *salons* permettent aussi de hiérarchiser les messages, alors qu'avec *Messenger* un message urgent risque d'être manqué s'il est envoyé entre plusieurs messages moins importants. L'intérêt de *Discord* est manifeste de paire avec des méthodes de travail itératives puisque l'application permet de soumettre instantanément une version intermédiaire d'un livrable au reste de l'équipe et de recevoir des retours sur celle-ci.

Enfin, l'ensemble des documents et travaux réalisés par l'équipe est publié sur son *Drive*. Si *Discord* permet de se tenir au courant des sujets traités à un moment donné, le *Drive* joue un rôle d'archive accessible à chacune des parties prenantes du projet.



FIGURE 22 – Logos des applications *Messenger*, *Discord* et *Google Drive*(lien vers le drive)

L'outil utilisé pour les réunions en distanciel est laissé à la discréction des organisateurs. *Google Meets* et *Teams* sont privilégiés pour les réunions réunissant toute l'équipe, alors que *discord* est préféré pour les échanges techniques souvent informels entre membres d'un même pôle.

3.3 Gestion de projet

3.3.1 Présentation

La gestion de projet est essentielle lorsque l'on travaille en groupe sur un projet commun. Elle permet de définir les objectifs du projet, de planifier et de suivre les tâches à accomplir, de gérer les ressources et de respecter les délais. Elle permet également de

communiquer efficacement avec tous les membres du groupe et de résoudre les problèmes de manière organisée.

Le projet suivait essentiellement une méthode agile. Pendant la phase de *Travail technique* et celle de *Finalisation*, chaque tâche correspondait à une fonctionnalité concrète du produit, et les tâches suivantes contenait bien souvent une amélioration des fonctionnalités déjà développées.

3.3.2 Réunions

Les réunions d'avancement réunissant toute l'équipe organisées de manière hebdomadaire par le chef de projet suivent toutes le schéma suivant :

1. Annonce de l'ordre du jour
2. Rappel des tâches en cours et résumé des avancées depuis la dernière réunion
3. Rappel des jalons
4. Discussion sur les objectifs à atteindre et mise au point de tâches adéquates
5. Mise au point d'une To-do list

Ce schéma est par ailleurs souvent utilisé dans les réunions par pôle.

3.3.3 Documents

Pendant la phase de *Définition* du projet, plusieurs supports de gestion de projet ont été rédigés, puis mis à jour au fur et à mesure de l'évolution du projet :

Le *Cahier des Charges Fonctionnels* (CdCF), qui décrit l'ensemble du produit visé, sans forcément rentrer dans les détails techniques. Le projet est réussi, du point de vue de l'équipe, lorsque chaque ligne du cahier des charges est satisfaite. L'équipe est poussée à dépasser ces objectifs et de ce fait, ce document décrit davantage un prototype qu'un produit commercialisable.

Carte de commande et énergie				
FP6	Transmettre aux actionneurs un signal délivré par l'application de manière instantanée	Temps de réponse	t5% < 200ms entre l'application et les actionneurs	1 ▼
FP7	Communication avec les actionneurs conçus par l'IEMN	Puissance et nombre de sorties suffisantes	- Puissance de 0,3125W par actionneur (à discuter) - au moins N sorties (N à définir)	1 ▼
FP8	Communication avec l'application dans des environnements variés	Utiliser une technologie sans-fil (Bluetooth/WiFi)	Puissance suffisante pour fonctionner à travers le tissu ou une cloison fine	1 ▼
FP9	Recharge facile	Utilisation d'une batterie et d'un port de chargement classique (USB)		▼
FP10	Comfort et simplicité			▼
FP11	Ne pas être trop complexe	Simplicité de développement (outils adaptés à des étudiants sans expérience)		▼
FP12	Respect de l'environnement	Matériaux et consommation d'énergie		▼
FC10.1	L'ensemble carte électronique + batterie ne doit pas être trop volumineux	Dimensions	Volume de moins de 15cm3	2 ▼
FC10.2	Le système doit avoir une autonomie suffisante	Autonomie	Autonomie d'au moins 6h	3 ▼
FC10.3	Le système ne doit pas être dangereux	Intensité du courant et force des actionneurs	Intensité de moins de 100mA et contact limité à l'électronique	3 ▼
FC11.1	Le système doit pouvoir être facilement débogué	Interface de débogage présente		▼
FC11.2	Le système ne doit pas être trop coûteux	Prix	Prix d'une unité à moins de 50€ en matières premières	3 ▼
FC11.3	Le système doit être rapidement démontable	Temps de démontage	Démontage en moins d'une minute	3 ▼
FC11.4	Le système doit avoir une durée de vie suffisante	Obsolescence	Système fonctionnant pendant au moins 2 ans	1 ▼

FIGURE 23 – Extrait du Cahier des Charges Fonctionnel du projet. Retrouvez le cahier des charges complet en annexes

Le *WBS*, qui est une liste de chaque tâche à effectuer pour le bien du projet.

6 Déroulement de la partie électronique										
6.1 Vers un premier prototype V0										
6.11	Autoformation sur l'utilisation de l'ESP32 et du BLE	Jules, Amine			15/02/22	15/03/22	30		100 %	
6.12	Etablissement d'une connexion bluetooth entre un téléphone et l'ESP32	Jules, Amine			04/04/22	19/09/22	165		100 %	
6.13	Pilotage des actionneurs avec l'ESP32	Jules, Amine			04/04/22	19/09/22	165		100 %	
6.14	Intégration des différentes parties du code	Jules, Amine			19/09/22	24/10/22	35		100 %	
6.15	Etude du circuit électrique en effectuant des simulations sur LtSpice	Omar, Murillo, Wassim			15/02/22	01/03/22	16		100 %	
6.16	Réalisation d'un premier circuit sur breadboard	Omar, Murillo, Wassim			01/03/22	11/04/22	40		100 %	
6.17	Pilotage des actionneurs et du circuit avec un GBF	Omar, Murillo, Wassim			08/03/22	18/04/22	40		100 %	
6.18	Autoformation sur l'utilisation de Kicad	Omar, Murillo			04/04/22	04/05/22	30		100 %	
6.19	Conception d'une carte électronique sur Kicad	Omar			04/05/22	04/06/22	30		100 %	

FIGURE 24 – Extrait du WBS du projet

La *matrice RACI*, qui décrit chaque tâche mentionnée dans le WBS en y ajoutant les parties prenantes associées et leurs rôles. Elle est surtout utile aux différents responsables qui s'en servent pour déléguer le travail de manière équilibrée.

	Kraft Alexis	Saland Lucas	Guckert Mathis	Chadli Mohammed	Da Silva Del Vecchio	Saubry Omar	Guillan Paul	Sylvestre Paul	Piccinini Thibaud	Tiedrez Roman	Mejjad Wassim	Dumezy Jules	José Vitor Martins Makiyama
Lot 1	R	R	R	R	R	R	R	R	R	R	R	A/R	I
	R	R	R	R	R	R	R	R	R	R	R	A/R	I
	R	R	R	R	R	R	R	R	R	R	R	A/R	I
	I	I	I	I	I	I	I	I	I	I	I	A/R	I
	I	I	I	I	I	I	I	I	I	I	I	A/R	I
	R	R	R	R	R	R	R	R	R	R	R	A/R	I
	R	R	R	R	R	R	R	R	R	R	R	A/R	I
Lot 2	R	R	R	R	R	R	R	R	R	R	R	A/R	I
	I	I	I	I	I	I	I	I	I	I	I	A/R	I
	R	R	R	R	R	R	R	R	R	R	R	A/R	I
	R	R	R	R	R	R	R	R	R	R	R	A/R	I
	R	R	R	R	R	R	R	R	R	R	R	A/R	I
	R	R	R	R	R	R	R	R	R	R	R	A/R	I
	R	R	R	R	R	R	R	R	R	R	R	A/R	I

FIGURE 25 – Extrait de la matrice RACI du projet

Le *diagramme de Gantt*, qui est une version schématisée du WBS qui représente le déroulé chronologique d'un projet. Il est principalement utilisé par le chef de projet pour suivre les délais mais tous les membres de l'équipe sont encouragés à en prendre connaissance pour savoir de quelles manières leurs tâches s'inscrivent dans la réalisation du projet. Le diagramme de Gantt prend toute son importance en clôture du projet comme support de réflexion pour dresser un bilan.



FIGURE 26 – Extrait du diagramme de Gantt du projet

Un *plan de gestion des risques*, qui permet à chaque acteur du projet d'être conscient des écueils susceptibles d'entraver son bon déroulement.

3. Les risques concernant le pôle mécanique						
Les machines de fabrication/techniciens sont indisponibles au moment voulu	3,5	2	7	Alexis	Anticiper clairement les étapes de réalisation et planifier les séances en atelier	Reporter les séances de travail ou replanifier les rendez-vous
Blessures lors de l'utilisation des machines /ors de la fabrication	1	1	1	Alexis	Anticiper clairement les étapes de réalisation qui pourraient causer de telles blessures	Rester toujours vigilant, porter des EPI si nécessaire
4. Les risques concernant le pôle électronique						
Le matériel est indisponible au moment voulu	3,5	2	7	Omar	Anticiper clairement les étapes de réalisation et commander en avance le matériel (potentiellement) nécessaire	Vérifier le planning du fablab à l'avance. Passer commande au plus tôt
Blessures lors de l'utilisation du courant (soudures ...)	1	1	1	Omar	Anticiper clairement les étapes de réalisation qui pourraient causer de telles blessures	Rester toujours vigilant, porter des EPI si nécessaire

FIGURE 27 – Extrait du plan de gestion des risques du projet

3.4 Diagrammes

3.4.1 Diagrammes d'analyse fonctionnelle

Les diagrammes d'analyse fonctionnelle sont un outil essentiel dans tout projet, puisqu'ils permettent d'assurer la compréhension, la communication et la planification efficaces des différentes parties du projet.

Nous avons ainsi choisi de nous concentrer sur la réalisation des diagrammes suivants :

- Diagramme d'expression du besoin : il permet de décrire les besoins et les exigences spécifiques auxquels le système doit répondre
- Diagramme pieuvre fonctions principales : il montre les fonctions principales du système et les relations entre elles.
- Diagramme pieuvre fonctions contraintes : il permet de comprendre les limites du système et les besoins non fonctionnels.

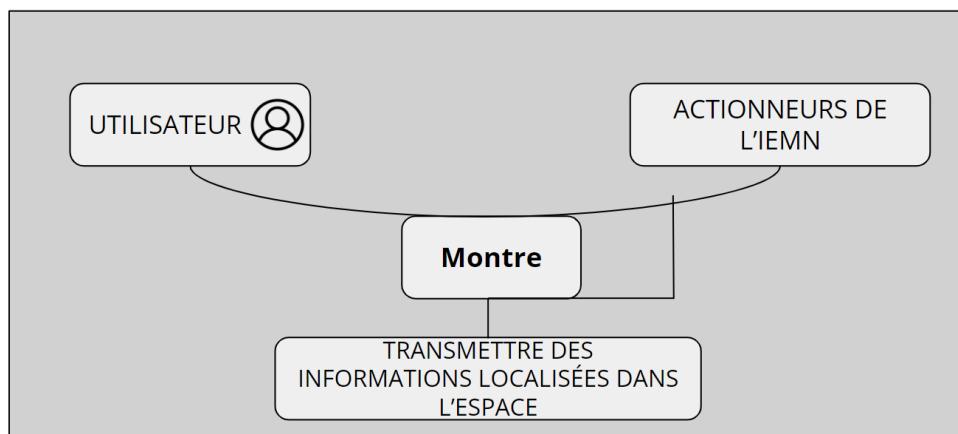


FIGURE 28 – Diagramme d'expression du besoin

Vous pouvez retrouver l'ensemble des autres diagrammes en Annexes.

3.4.2 Diagrammes SYML

Les diagrammes *SYSML* (SYstem Modeling Language) sont des outils très importants dans tout projet puisqu'ils modélisent le système de manière formelle et rigoureuse permettant ainsi de comprendre ses exigences, ainsi que les relations entre les différents éléments qui le composent.

Nous avons ainsi choisi de nous concentrer sur la réalisation des diagrammes suivants :

- Diagramme de cas d'utilisation : il permet de comprendre les fonctionnalités du système et les besoins des utilisateurs.
- Diagramme de contexte : il permet de comprendre l'environnement dans lequel le système opère.

- Diagramme de séquence : il permet de comprendre comment le système répond aux scénarios d'utilisation.
- Diagramme d'exigence : il permet de s'assurer que le système répond aux besoins des utilisateurs et aux exigences du projet.

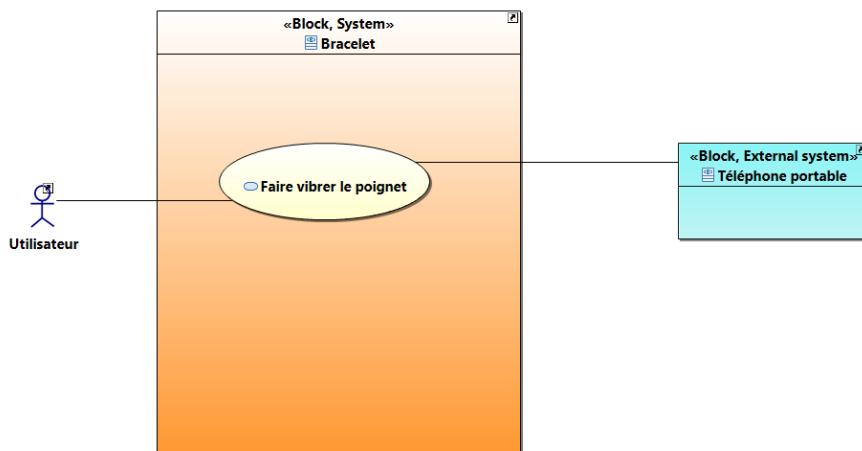


FIGURE 29 – Diagramme de cas d'utilisation

Vous pouvez retrouver l'ensemble des autres diagrammes SYSML en Annexes.

3.5 Gestion du budget

La rédaction d'une convention avec le client fut étudiée et il apparut que ce n'était pas nécessaire. En effet, le seul point qui mérite de l'attention au niveau de la propriété intellectuelle est l'utilisation d'actionneurs développés par l'IEMN. Or, ceux-ci ayant fait l'objet de publications scientifiques qui précèdent le projet *Orion*, il a été convenu qu'il était inutile de prendre des mesures particulières.

Développer une application comme *Orion* est certes sans coût, mais la publier sur un *store* engendre des frais. Les matériaux utilisés pour fabriquer le boîtier et le bracelet sont disponibles en libre-accès au sein des infrastructures de Centrale Lille mais représentent quand même un coût. Également, l'achat du matériel électronique est à prendre en compte. Les dépenses liées au projet sont résumées dans le tableau suivant :

Pôle	Pièce	Prix	Quantité	Total
Mecanique	Neopixel Ring 12	9,64 €	1	9,64 €
Mecanique	Barrette ressort de fixation du bracelet	3,90 €	2	7,80 €
Mecanique	Boucle fermoir bracelet	7,40 €	1	7,40 €
Mecanique	Pointeau de pose	2,70 €	1	2,70 €
Electronique	Impression de la carte (PCB)	5,00 €	1	5,00 €
Informatique	Compte Publisher Google	25,00 €	1	25,00 €
Electronique	Transistor : ZVN3306F (pour 5)	0,67 €	1	0,67 €
Electronique	Diode : MBR230LSFT1G (pour 5)	0,48 €	1	0,48 €
Electronique	Résistances	0,20 €	1	0,20 €
Electronique	ESP32	13,90 €	1	13,90 €
Electronique	Batterie	1,20 €	1	1,20 €
Mecanique	Matériaux caisning	3,00 €	1	3,00 €
Mecanique	Matériaux bracelet	3,00 €	1	3,00 €
				79,99 €

FIGURE 30 – Budget du projet (cliquer ici pour voir en plus grand)

3.6 Formations et autoformations

La quasi-totalité des outils et méthodes utilisés pour mettre au point puis fabriquer le produit étaient inconnus de l'équipe projet lorsque ce dernier fut lancé. Il a été nécessaire de s'y former. Les responsables de chaque pôle eurent non seulement la responsabilité de choisir ces outils et méthodes, mais aussi celle de désigner par quels moyens les connaissances seraient acquises. En particulier, leur jugement était attendu pour savoir si les formations seraient encadrées par un expert ou pouvaient être faites de manière autonome.

La phase de *Travail technique* a ainsi démarré par un temps de formation aux outils que nous avions décidé d'utiliser, pour un total de 35.5h de formations résumées dans le tableau suivant, extrait du budget d'expertise du projet :

Numéro	Type d'expertise	Points abordés	Intervenant(s)	Durée (en heures)
1	Consulting	Aide pour le début du projet : mettre en place des	Yannick Dusch	1
2	Consulting	Validation du cahier des charges + points sur l'au	Yannick Dusch	1
3	Autoformation	Formation JAVA	Diego Cattaruzza, Pierre Hosteins, Matteo Pe	7
4	Autoformation	Base de la conception sur Android	Thomas Bourdeaud'huy	7
5	Consulting	Bilan sur la première version de l'application. Mor	Thomas Bourdeaud'huy	0.5
6	Consulting	Points de gestion de projet + conseils pour l'audit	Yannick Dusch	1
7	Autoformation	Fonctionnement d'un transistor NMOS	Simon Thomy et Abdelkrim Talbi	3
8	Autoformation	Impression d'une carte électronique	Simon Thomy	
9	Consulting	Aide pour la conception de la membrane en PDM	Salah Moussa	2
10	Consulting	Aide à la conception du bracelet et de son moule	Laurent Patrouix	7
11	Autoformation	Conception de la carte électronique sur Kicad	Simon thomy	4
12	Consulting	Impression d'une carte électronique / soudage	Simon thomy	2
Total				35.5

FIGURE 31 – Extrait du budget d'expertise du projet

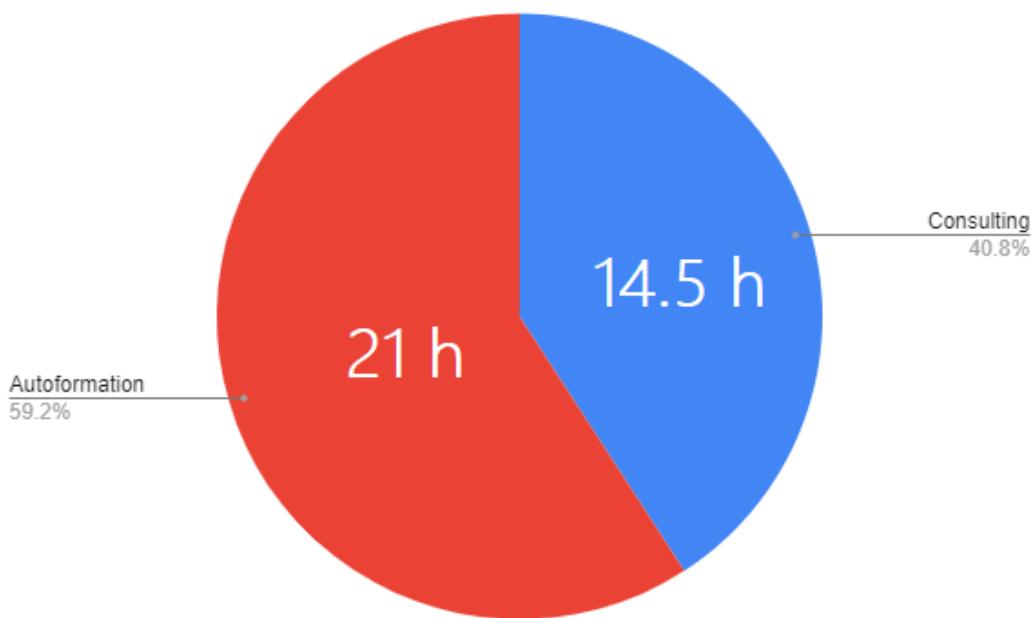


FIGURE 32 – Répartition du budget d'expertise du projet

4 Rapport scientifique : pôle informatique

4.1 Introduction

Aujourd’hui la grande majorité du marché de la téléphonie mobile est dominé par le système d’exploitation d’Android. En effet, en 2021, 84 % des téléphones en circulation sont estampillés Android contre seulement 16 % pour Apple. De nombreuses raisons peuvent expliquer cette différence comme notamment le prix, mais aussi le fait d’être *open source*. Effectivement, le code source qui permet à un appareil Android de fonctionner est libre et accessible à tous. Ainsi, chaque constructeur (comme Samsung, Huawei, OnePlus...) est libre de proposer sa propre version de smartphone sans pour autant repartir de zéro en concevant son propre système d’exploitation.



FIGURE 33 – Le duopole d’Android et d'iOS.

Dans le cadre de notre projet, nous avons été amené à concevoir une application pour smartphone. Son but est de pouvoir communiquer avec une carte électronique en utilisant la technologie Bluetooth LE pour échanger des informations sur des notifications reçues par le téléphone. Le paragraphe précédent explique en partie notre choix de se tourner vers une solution pour appareils Android. En effet, si il est plus simple pour un constructeur de travailler sur Android, il en va de même pour un développeur.

Dans cette partie, nous allons expliquer comment nous avons réussi à mettre en place une telle solution. Cependant, nous ne développerons pas la réalisation de notion que nous jugeons comme "classiques" dans la conception d'un application Android (comme par exemple : comment insérer un bouton, comment compiler son code...), mais nous nous attarderons sur les spécificités propres de notre solution.

Nous avons décidé de travailler avec le logiciel Android Studio qui s'impose comme étant la référence pour développer des applications natives Android. Comme expliqué précédemment, nous utiliserons des termes techniques relatifs à ce logiciel et nous ne détaillerons pas son fonctionnement.

Nous organiserons cette partie en cinq parties principales :

1. Navigation au sein de l’application
2. Gestion des autres applications du téléphone
3. Communication Bluetooth Low Energy
4. Gestion des notifications du téléphone
5. Gestion de l’heure

4.2 Navigation au sein de l'application

4.2.1 Contexte

Si le smartphone fut un temps vu comme un gadget, il fait maintenant partie de notre panoplie quotidienne, au côté de notre portefeuille et de notre trousseau de clés. 77% de la population en possède un (INSEE, 2021) et les nouvelles générations ont grandi avec un smartphone dans les mains. Ces nouvelles générations ont ainsi acquis un ensemble de réflexes qui leur permettent de prendre en main un nouveau smartphone ou une nouvelle application de manière automatique. On conçoit aisément que l'utilisateur "moyen" de smartphone n'a jamais eu besoin de lire le mode d'emploi de son appareil, ni même d'aucune de ses applications.

Ce dernier point n'est pas anodin. Si les modes d'emplois semblent appartenir au passé, c'est que les nouveaux produits suivent un ensemble de normes souvent tacites que les utilisateurs ont internalisées. Par exemple, l'utilisateur saura qu'appuyer sur une partie du texte le redirigera vers une page web parce que cette partie du texte est colorée en bleu et soulignée. Ce même utilisateur aura instantanément repéré que la roue dentée dans le coin de son écran permet d'ouvrir le menu des paramètres de l'application. Etc.

Pour le développeur, cela s'apparente à une liste de contraintes qu'il doit obligatoirement respecter, sous peine de perturber ses utilisateurs potentiels. Si elles définissent aussi des lignes directrices, des "questions qu'il n'aura pas à se poser" (sur le choix d'une icône pour un bouton, par exemple), elles restreignent aussi la part laissée à l'imagination.

Parmi ces attentes en matière d'ergonomie se trouve la manière de naviguer au sein de l'application. Dans cette partie, nous présenterons les principales décisions prises pour mettre au point l'architecture de notre application en abordant certains détails techniques de la navigation au sein de celle-ci.

4.2.2 Arborescence d'une application Android

Android Studio propose d'afficher l'arborescence des fichiers utilisés pour construire l'application de la manière suivante :

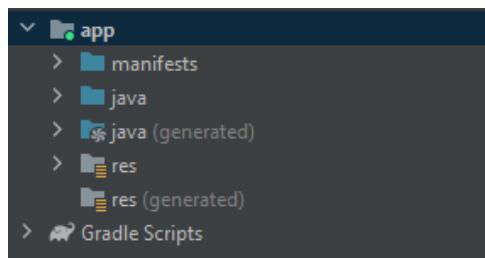


FIGURE 34 – Arborescence standard d'une application Android en cours de développement

De haut en bas :

- Le dossier **manifests** ne contient généralement que le fichier **AndroidManifest.xml** qui contient l'icône de l'application, son nom, la liste de ses Activités, son icône, et, le cas échéant, les permissions qui lui sont accordées ou encore les Services employés par l'application.

- Le dossier `java` contient l'essentiel du code de l'application : Activités, Fragments, Services et Classes écrites en *Java*, *Kotlin* voire *C++* éventuellement répartis dans différents *packages*. Ce dossier contient aussi par défaut des classes dédiées aux tests unitaires et instrumentés.
- Le dossier `java (generated)` est généré automatiquement par Android Studio et sert à la compilation. Il n'est pas censé être modifié
- Le dossier `res` contient l'ensemble des éléments de l'UI : images, widgets, couleurs, chaînes de caractères...
- Le dossier `res (generated)` est généré automatiquement par Android Studio et sert à la compilation. Il est généralement vide.
- Le dossier `Gradle Scripts` contient les informations permettant à Android Studio de compiler le code du développeur pour en faire une application. Il contient des informations sur la version de l'application, ses dépendances... Il peut par exemple être modifié pour demander au compilateur de réduire la taille des noms de variables avant de compiler afin de rendre l'application plus légère.

Le développeur passe 99% de son temps à travailler dans les dossiers `java` et `res`. Nous avons uniquement modifié le dossier `manifests` afin d'indiquer les permissions requises par l'application et modifier son icône, le reste étant généré automatiquement.

4.2.3 Fragments

Commençons par rappeler ce qu'est un fragment. Un fragment peut être représenté comme une partie réutilisable de notre interface graphique. Ainsi, en pratique, une activité va être découpée en un ou plusieurs fragments afin de créer une interface utilisateur modulable et surtout flexible visuellement parlant. Chaque fragment sera contenu dans son activité parente (souvent c'est *MainActivity*) et héritera de sa classe.

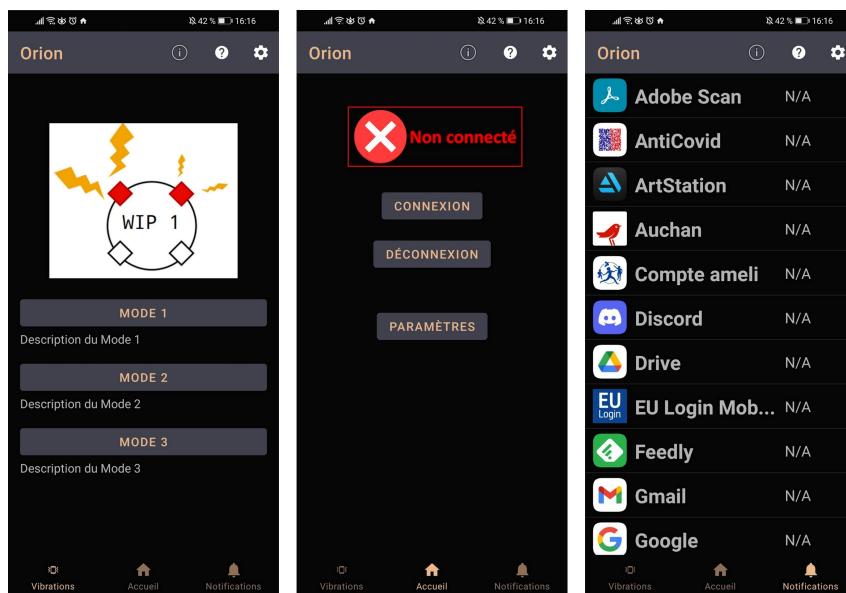


FIGURE 35 – Exemple de fragments

On peut commencer à mettre en place notre fragment. Ainsi un code comme celui qui suit est capable de créer un fragment (ici nommé *VibrationsFragment*) :

```

1  public class VibrationsFragment extends Fragment {
2      @Override
3      public void onCreate(Bundle savedInstanceState) {
4          super.onCreate(savedInstanceState);
5      }
6      public View onCreateView(@NonNull LayoutInflater inflater, ViewGroup container,
7          Bundle savedInstanceState) {
8          com.example.tactigant20.databinding.FragmentVibrationsBinding binding =
9              FragmentVibrationsBinding.inflate(inflater, container, false);
10         View root = binding.getRoot();
11         [1]
12         return root;
13     }
14 }
```

Il suffit d'ajouter (au niveau de la balise [1]) les définitions des éléments du fragment (boutons, images...) que l'on souhaite. Chaque fragment a un fichier XML qui lui est propre et qu'il convient de créer. C'est dans ce dernier que l'on ajoute les boutons, images, textes...

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     android:layout_width="match_parent" android:layout_height="match_parent"
4     android:orientation="vertical" >
5
6     <TextView
7         android:id="@+id/textView1"
8         android:layout_width="wrap_content"
9         android:layout_height="wrap_content"
10        android:text="TextView" />
11
12     <Button
13         android:id="@+id/button1"
14         android:layout_width="wrap_content"
15         android:layout_height="wrap_content"
16         android:text="Button" />
17
18 </LinearLayout>
```

Pour terminer, il suffit d'ajouter le fragment au fichier XML parent. De manière statique on peut procéder de la sorte :

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     android:layout_width="match_parent"
4     android:layout_height="match_parent"
5     android:orientation="vertical" >
6
7     <fragment
8         android:name="com.example.android.FooFragment"
9         android:id="@+id/fooFragment"
10        android:layout_width="match_parent"
11        android:layout_height="match_parent" />
12
13 </LinearLayout>
```

Si vous souhaitez travailler avec des fragments, au lieu de partir de zéro comme présenté précédemment, nous vous conseillons de travailler à l'aide de l'exemple proposé par Android studio qui permet d'initialiser beaucoup plus simplement des fragments. L'application *Orion* est composée de quatre activités :

- **MainActivity**, qui est elle même composée des fragments **VibrationsFragment**, **HomeFragment** et **NotificationsFragment**
- **HelpActivity**
- **InfoActivity**
- **SettingsActivity**

Ces trois dernières activités sont uniquement accessibles depuis **MainActivity**. Cette subsection présente la solution adoptée pour y accéder, à savoir la barre d'outil, qui s'est imposée comme un standard. Elle permet entre autres d'accéder au paramètres de l'application en cliquant sur un bouton situé en haut à droite de l'écran, ce qui est une convention bien entérinée.



FIGURE 36 – La *Toolbar* de l'application *Orion*

Pour commencer, il suffit de créer un fichier de layout décrivant un menu dans lequel on liste chacune des pages avec leurs icônes respectives :

```

1 <menu xmlns:android="http://schemas.android.com/apk/res/android"
2   xmlns:app="http://schemas.android.com/apk/res-auto"
3   xmlns:tools="http://schemas.android.com/tools"
4   tools:context="com.example.tactigant20.MainActivity">
5
6   <item
7     android:id="@+id/action_info"
8     android:icon="@android:drawable/ic_menu_info_details"
9     android:title="@string/infos"
10    app:showAsAction="ifRoom" />
11
12   <item
13     android:id="@+id/action_help"
14     android:icon="@mipmap/ic_aideblanc_foreground"
15     android:title="@string/help"
16     app:showAsAction="ifRoom" />
17
18   <item
19     android:id="@+id/action_settings"
20     android:icon="@drawable/ic_settings_white"
21     android:title="@string/settings"
22     app:showAsAction="ifRoom" />
23
24 </menu>
```

Ensuite, il convient d'indiquer à l'activité qui contient la barre d'outils qu'elle doit être munie de ce menu, et que ce menu doit apparaître sous forme de barre d'outils. Ainsi, on ajoute à **activity_main.xml** le code suivant :

```

1 <com.google.android.material.appbar.MaterialToolbar
2     android:id="@+id/topAppBar"
3     android:layout_width="match_parent"
4     android:layout_height="?attr/actionBarSize"
5     app:menu="@menu/toolbar"
6     style="@style/Widget.MaterialComponents.Toolbar.Primary" />

```

Il faut ensuite instancier cette barre dans `MainActivity.java`...

```

1 Toolbar topAppBar = findViewById(R.id.topAppBar);
2 setSupportActionBar(topAppBar);

```

...et l'associer aux boutons définis plus tôt :

```

1 @Override
2 public boolean onCreateOptionsMenu(Menu menu) {
3     getMenuInflater().inflate(R.menu.toolbar, menu);
4     return true;
5 }

```

Il ne reste plus qu'à définir ce que fait chaque bouton, à savoir lancer l'activité associée :

```

1 @Override
2 public boolean onOptionsItemSelected(MenuItem item) {
3     int id = item.getItemId();
4
5     switch (id) {
6         case (R.id.action_help):
7             launchActivity(HelpActivity.class);
8             return true;
9         case (R.id.action_info):
10            launchActivity(InfoActivity.class);
11            return true;
12         case (R.id.action_settings):
13             launchActivity(SettingsActivity.class);
14             return true;
15         default:
16             return super.onOptionsItemSelected(item);
17     }
18
19 }
20
21 private void launchActivity(Class<? extends Activity> activity) {
22     Intent intent = new Intent(this, activity);
23     startActivity(intent);
24 }

```

Pour retourner à `MainActivity` depuis une des autres activités, il suffit d'appuyer sur le bouton de retour présent sur les appareils Android. Cette fonction est assurée de manière automatique et n'a pas besoin d'être programmée.

4.2.4 Navigation entre fragments

Cette subsection traite des différentes manières de naviguer entre les trois pages principales de notre application.

→ **Barre de navigation :**

Notre application comportant trois menus principaux d'importance égale, il est nécessaire de pouvoir passer de l'un à l'autre très rapidement. La solution naturelle pour se faire est une barre de navigation placée en bas de l'écran.



FIGURE 37 – La *Bottom Navigation Bar* de l'application *Orion*

La création d'une barre de navigation commence de manière identique à la barre d'outils :

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <menu xmlns:android="http://schemas.android.com/apk/res/android">
3
4     <item
5         android:id="@+id/navigation_vibrations"
6         android:icon="@mipmap/ic_vibration_foreground"
7         android:title="@string/title_vibrations" />
8     <item
9         android:id="@+id/navigation_home"
10        android:icon="@drawable/ic_home_black_24dp"
11        android:title="@string/title_home" />
12     <item
13         android:id="@+id/navigation_notifications"
14         android:icon="@drawable/ic_notifications_black_24dp"
15         android:title="@string/title_notifications" />
16
17 </menu>
```

Et dans `activity_main.xml` :

```

1 <FrameLayout
2     android:id="@+id/fragment_container"
3     android:layout_width="match_parent"
4     android:layout_height="match_parent"
5     android:layout_above="@+id/nav_view" />
6
7 <com.google.android.material.bottomnavigation.BottomNavigationView
8     android:id="@+id/nav_view"
9     android:layout_width="0dp"
10    android:layout_height="wrap_content"
11    android:background="?android:attr/windowBackground"
12    app:layout_constraintBottom_toBottomOf="parent"
13    app:layout_constraintLeft_toLeftOf="parent"
14    app:layout_constraintRight_toRightOf="parent"
15    app:menu="@menu/bottom_nav_menu" />
```

Reste à associer chaque `item` de la barre de navigation au fragment correspondant. Nous y reviendrons après la sous-section suivante.

→ **Swipe :**

L'utilisateur de smartphone moyen en 2022 peut légitimement s'attendre à pouvoir balayer l'écran ("Swipe") d'une page à l'autre quand il voit une barre de navigation en bas de son écran. Ceci peut être réalisé grâce à l'objet `ViewPager2` proposé par Android.

Tout comme la barre de navigation, le `ViewPager2` doit être déclaré dans le layout de l'activité correspondante :

```

1 <androidx.viewpager2.widget.ViewPager2
2     android:id="@+id/vpPager"
3     android:layout_width="match_parent"
4     android:layout_height="match_parent">
5 </androidx.viewpager2.widget.ViewPager2>
```

Au-delà de ce point, il n'existe *a priori* pas de manière unique et formelle d'implémenter cette fonctionnalité. Dans une logique orientée objet, il a été décidé de placer les trois fragments dans une liste. Nous créons une classe dédiée à la manipulation de cette liste :

```

1 public class SwipeAdapter extends FragmentStateAdapter {
2
3     private final ArrayList<Fragment> fragmentList = new ArrayList<>();
4
5     public SwipeAdapter(@NonNull FragmentManager fragmentManager, @NonNull Lifecycle
6                         lifecycle) {
7         super(fragmentManager, lifecycle);
8     }
9
10    @NonNull
11    @Override
12    public Fragment createFragment(int position) {
13        return fragmentList.get(position);
14    }
15
16    public void addFragment(Fragment fragment) {
17        fragmentList.add(fragment);
18    }
19
20    @Override
21    public int getItemCount() {
22        return fragmentList.size();
23    }
}
```

On peut alors placer chaque fragment dans la liste en rentrant le code suivant dans `MainActivity` (le `ViewPager2` est ici un attribut de `MainActivity`) :

```

1 private ViewPager2 myViewPager2;
2 [...]
3 myViewPager2 = findViewById(R.id.vpPager);
4 SwipeAdapter mySwipeAdapter = new SwipeAdapter(getSupportFragmentManager(),
5     getLifecycle());
5 mySwipeAdapter.addFragment(vibrationsFragment);
6 mySwipeAdapter.addFragment(homeFragment);
7 mySwipeAdapter.addFragment(notificationsFragment);
8 myViewPager2.setAdapter(mySwipeAdapter);

```

On précise aussi que le balayage doit être horizontal et que `HomeFragment` doit être affiché en premier :

```

1 myViewPager2.setOrientation(ViewPager2.ORIENTATION_HORIZONTAL);
2 myViewPager2.setCurrentItem(1, false);

```

→ Retour sur la barre de navigation :

L'intérêt de cette méthode est qu'elle permet de coupler l'emploi d'une barre de navigation et d'un balayage de l'écran, ce qui est loin d'être anodin. En effet, le second a besoin de "savoir" quel fragment est actuellement affiché pour "savoir où aller" lorsque l'utilisateur balaye son écran.

On utilise ainsi le code suivant, qui couple les deux fonctionnalités en indiquant au `ViewPager2` quel fragment est sélectionné à chaque fois qu'on clique sur un élément de la barre de navigation :

```

1 NavigationBarView.OnItemSelectedListener navListener = item -> {
2
3     int itemId = item.getItemId();
4     if (itemId == R.id.navigation_vibrations) {
5         myViewPager2.setCurrentItem(0);
6     } else if (itemId == R.id.navigation_home) {
7         myViewPager2.setCurrentItem(1);
8     } else if (itemId == R.id.navigation_notifications) {
9         myViewPager2.setCurrentItem(2);
10    }
11
12    return true;
13 };
14
15 BottomNavigationView bottomNav = findViewById(R.id.nav_view);
16 bottomNav.setOnItemSelectedListener(navListener);

```

Sur le plan technique, on instancie un `OnItemSelectedListener` : un objet qui réagit à chaque fois que l'utilisateur appuie sur un élément de la barre du bas. Les deux dernières lignes correspondent à l'instanciation effective de la barre.

4.2.5 Architecture générale - modèle MVC

Un point majeur dans l'évolution de notre projet est l'intégration du concept de *séparation des préoccupations*, c'est-à-dire le fait, dans la mesure du possible, d'isoler chaque aspect de l'application dans des fichiers indépendants. Cela évite d'avoir des fichiers longs

et illisibles parce qu'ils gèrent des fonctionnalités diverses les unes à la suite des autres. Cela évite aussi de perdre du temps à modifier une multitude de fichiers parce qu'une fonctionnalité est référencée à de nombreux endroits différents, puisque la fonctionnalité est alors décrite dans un unique fichier dédié.

Pour donner un exemple concret, la gestion du BLE (*Bluetooth Low Energy*) dans notre application était auparavant étalée sur l'ensemble des fichiers ayant à voir avec la communication avec la carte électronique. En particulier `HomeActivity` contenait l'ensemble des instanciations des objets liées au BLE (`gatt`, `UUID`, `MAC...`) simplement parce que le bouton permettant de lancer l'appairage faisait partie de ce fragment ! Cela signifiait d'abord que `HomeFragment` faisait plusieurs centaines de lignes. Ensuite, l'UI et le BLE étaient étroitement liés, si bien qu'une modification de l'un entraînait un dysfonctionnement de l'autre. Toute la logique de connexion plantait dès qu'on touchait au bouton `CONNEXION`, avec des ramifications dans `NotificationsFragment` parce que ce dernier participait aussi à configurer les informations envoyées à la carte... La lecture des *Logs* devenait fastidieuse et le déboguage prenait des heures pour un rien.

Pour y remédier, nous avons adopté le motif d'architecture *Modèle-Vue-Présentateur* (MVP) :

- Le *Modèle* désigne l'ensemble des fichiers qui concernent la logique métier de l'application, c'est-à-dire l'ensemble des manières possibles de modifier les données. Il est réutilisable par toute application qui manipule les mêmes types de données.
- La *Vue* est l'ensemble des éléments susceptibles d'apparaître à l'écran ainsi que leur agencement par défaut. Il est réutilisable par toute application dont les pages sont agencées de manière similaire.
- Le *Présentateur* désigne l'ensemble des fichiers qui font le lien entre les deux autres. Il met à jour l'interface en fonction de la valeur des données et met à jour les données en fonction des interactions de l'utilisateur avec l'interface.

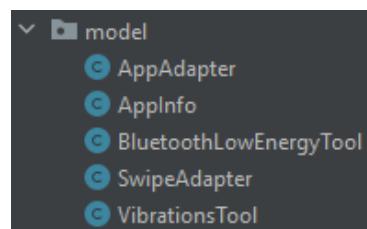


FIGURE 38 – Le Modèle de notre application est contenu dans un dossier

Pour reprendre l'exemple précédent, décrivons les composants qui permettent à l'utilisateur de lancer l'appairage BLE :

La Vue contient le fichier `fragment_home.xml` qui décrit l'UI de `HomeFragment`. Le code suivant détermine l'emplacement d'un bouton, *sans en décrire le fonctionnement* :

```

1 <Button
2     android:id="@+id/connectionButton"
3     android:layout_width="wrap_content"
4     android:layout_height="wrap_content"
5     android:layout_marginTop="8dp"
6     android:text="@string/connection"
7     app:layout_constraintEnd_toEndOf="parent"
8     app:layout_constraintStart_toStartOf="parent"

```

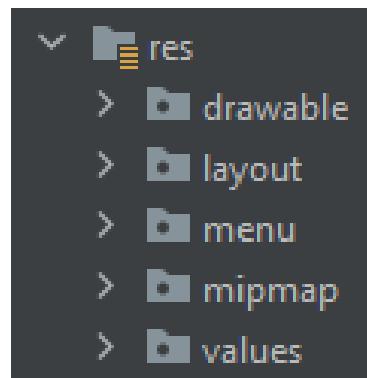


FIGURE 39 – La Vue de notre application correspond concrètement au contenu du dossier *res*

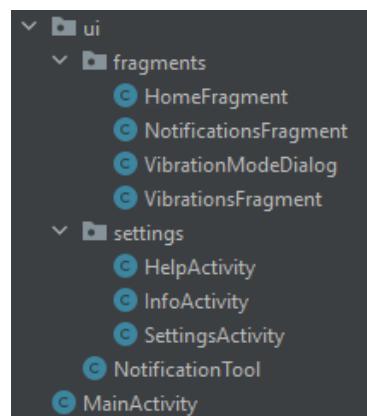


FIGURE 40 – Les activités et fragments de l'application constituent notre Présentateur

```
9     app:layout_constraintTop_toBottomOf="@+id/connexionValide" />
```

Notons (première ligne) que ce bouton est associé à un ID. La ligne suivante indique que ce bouton doit se trouver dans **HomeFragment** :

```
1 tools:context=".ui.fragments.HomeFragment"
```

Le fichier **BluetoothLowEnergyTool.java** appartient au *Modèle* de notre application. Un **BluetoothLowEnergyTool** est un objet qui possède comme attribut l'adresse MAC de l'appareil auquel on souhaite se connecter (**mAdresseMAC**), entre autres. Il contient les fonctions **scan()** et **disconnect()** qui gèrent respectivement l'appairage et la déconnexion. Le développeur n'a pas nécessairement besoin de connaître en détail les processus invoqués par ces fonctions, qui peuvent être employées dans tout contexte où l'on souhaite s'appairer à un appareil BLE. Au sein de **MainActivity**, qui fait partie du *Présentateur*, on instancie un objet de la classe **BluetoothLowEnergyTool** en lui indiquant l'adresse MAC de l'objet auquel on souhaite s'appairer.

```
1 private static BluetoothLowEnergyTool myBLET;
2 [...]
3 myBLET = new BluetoothLowEnergyTool(AdresseMAC, this);
```

Le fichier `HomeFragment.java` appartient aussi au *Présentateur* de notre application. Il contient la fonction suivante :

```

1 private void cConnectionButton(View v) {
2     if (connectionButton.getText().equals(requireContext().getResources().getString(R.
3         string.connection))) {
4         MainActivity.getMyBLET().scan();
5     } else if (connectionButton.getText().equals(requireContext().getResources().getString(R.string.disconnection))) {
6         MainActivity.getMyBLET().disconnect();
7     } else {
8         Log.e(TAG_HOME, "Etat de connexion inconnu");
9     }
10 }
```

Cette fonction se charge d'appeler `scan()` ou bien `disconnect()` en fonction de l'état actuel de l'appairage. Enfin, on indique dans `HomeFragment` que la fonction `cConnectionButton` est appelée quand on appuie sur le bouton. C'est ici que l'ID mentionné plus haut intervient.

```

1 private Button connectionButton;
2 [...]
3 connectionButton = root.findViewById(R.id.connectionButton);
4 connectionButton.setOnClickListener(this::cConnectionButton);
```

Suivre un motif d'architecture prend tout son sens dans le cadre d'un travail collaboratif. Peu importe le placement et l'apparence du bouton, tant que le *Présentateur* n'est pas modifié, le bouton aura toujours la même fonctionnalité. Ainsi, les différentes équipes ne peuvent pas se "marcher dessus".

4.2.6 Résultats

Une fois tout ces éléments implémentés, nous nous retrouvons avec une solution qui, est certes fonctionnelle, mais surtout, qui respecte les méthodes et "bonnes pratiques" d'un développeur Android tout en proposant une solution ergonomique et intuitive pour l'utilisateur.

4.3 Gestion des autres applications du téléphone

4.3.1 Contexte

Un des aspects primordiaux de notre application est la capacité de l'utilisateur d'affecter les différents modes de vibrations existants à chacune des applications de son smartphone, permettant ainsi d'adapter le comportement du montre en fonction de la notification reçue. Pour cela, nous avons implémenté, dans le fragment `NotificationsFragment`, et sous forme de `ListView` Android, une liste de l'ensemble des applications du téléphone susceptibles d'émettre des notifications.

4.3.2 Fonctionnement d'une ListView Android

Une `ListView` est un élément de l'interface utilisateur Android qui permet de présenter une liste de données sous forme de liste verticale. Elle est généralement utilisée pour afficher une liste de choix à l'utilisateur, comme une liste de contacts ou de messages.

L'implémentation d'une `ListView` se fait en plusieurs étapes. Tout d'abord, il est nécessaire de définir un `layout` qui sera utilisé pour afficher chaque élément de la liste. Ce `layout` doit contenir les éléments de l'interface utilisateur que vous souhaitez afficher pour chaque élément de la liste, comme par exemple le nom de l'objet, un logo, une description...

Un second élément clé au fonctionnement d'une `ListView` est un adaptateur. L'adaptateur a pour rôle le remplissage de la `ListView` avec les données à afficher. L'adaptateur utilise le `layout` défini précédemment et instancie une vue pour chaque élément de la liste en utilisant les données fournies. Il existe plusieurs types d'adaptateurs dans Android, tels que `ArrayAdapter` et `CursorAdapter`, qui sont adaptés à différents types de données.

Enfin, si la `ListView` a pour but de permettre à l'utilisateur d'interagir avec ses éléments, il est nécessaire de définir un écouteur d'événement qui sera appelé chaque fois qu'un élément de la liste est sélectionné par l'utilisateur. Cet écouteur peut être utilisé pour afficher des informations supplémentaires sur l'élément sélectionné ou pour effectuer une action spécifique. Le plus souvent, c'est l'écouteur d'événement `onItemClickListener` qui sera utilisé, puisqu'il suffit de cliquer sur un des objets de la liste pour le déclencher.

4.3.3 Mise en place des premiers éléments

Cette partie traite de l'implémentation d'une liste des applications installées sur le smartphone de l'utilisateur, sous forme d'une `ListView`.

→ Type de données :

Avant même de réfléchir à la manière dont nous allons afficher les données, il est nécessaire d'établir précisément le type de données que nous allons traiter. Les données manipulées ici étant des "applications" auxquelles sera affecté un mode de vibration, il était nécessaire de définir une nouvelle classe avec les attributs suivant :

- `mLabel`, une chaîne de caractères contenant le nom de l'application,
- `mVibrationMode`, le mode de vibration associé à l'application (instance de `VibrationMode`, classe détaillée dans la prochaine partie),

- `mInfo`, instance de la classe `ApplicationInfo` contenant des informations sur l’application (nom du *package*, version...), ainsi que de nombreux *flags*, par exemple `FLAG_INSTALLED`, qui indique si l’application est installée ou non, ou `FLAG_SYSTEM`, qui indique si l’application fait partie du système du smartphone.

Une fois ces attributs correctement définis, nous avons pu créer classe `AppInfo`, avec les différents *getters* et *setters* (fonctions permettant d’accéder aux attributs d’une instance de cette classe et de les modifier) comme suit :

```

1 public class AppInfo {
2     private ApplicationInfo mInfo;
3     private String mLabel;
4     private VibrationMode mVibrationMode;
5     public AppInfo() {
6         this.mInfo = null;
7         this.mLabel = "Default Label";
8         this.mVibrationMode = VibrationMode.getDefaultVibrationMode();
9     }
10    public ApplicationInfo getInfo() {
11        return this.mInfo;
12    }
13    public void setInfo(ApplicationInfo mInfo) {
14        this.mInfo = mInfo;
15    }
16    public String getLabel() {
17        return this.mLabel;
18    }
19    public void setLabel(String mLabel) {
20        this.mLabel = mLabel;
21    }
22    public VibrationMode getVibrationMode() {
23        return this.mVibrationMode;
24    }
25    public void setVibrationMode(VibrationMode mVibrationMode) {
26        this.mVibrationMode = mVibrationMode;
27    }
28 }
```

→ Affichage d’un élément de la liste :

Le type de données maintenant choisi, il est essentiel de programmer l’affichage d’un objet `AppInfo` au sein de notre liste. Pour cela, on crée un nouveau layout nommé `app_item_layout.xml`, dans lequel on vient placer les différents éléments que l’on souhaite afficher. Dans notre cas, on souhaite avoir le nom de l’application, son logo, ainsi que le mode de vibration associé. En insérant deux *TextView* et un *ImageView*, on obtient la disposition suivante (pour une application) :

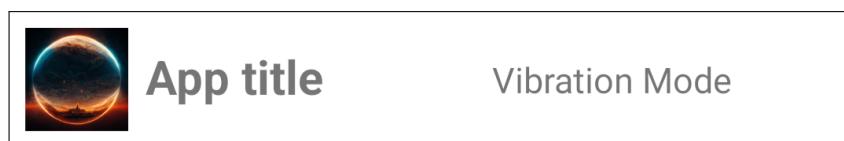


FIGURE 41 – Mise en page de chaque élément de la liste d’applications

→ **Implémentation de l'adaptateur :**

Comme expliqué dans le préambule, l'adaptateur fait le lien entre les données et l'interface utilisateur, en convertissant celles-ci en objets pouvant être affichés, en l'occurrence dans une `ListView`. Nous avons donc créé une nouvelle classe `AppAdapter`, qui hérite de la classe `ArrayAdapter` d'Android et qui permet de créer une liste d'applications à partir d'une liste d'objets de type `AppInfo` et du layout d'un élément (`app_item_layout`). Le choix d'hériter la classe `ArrayAdapter` est simplement lié au fait que cette classe est la plus couramment utilisée, et la plus adaptée aux données stockées dans des tableaux ou listes, ce qui est notre cas (les objets `AppInfo` étant stockés dans une `ArrayList`).

```

1 public class AppAdapter extends ArrayAdapter<AppInfo> {
2
3     private final LayoutInflater mLayoutInflater;
4     private final PackageManager mPackageManager;
5     private final List<AppInfo> mApps;
6
7     public AppAdapter(Context context, List<AppInfo> mApps) {
8         super(context, R.layout.app_item_layout, mApps);
9         this.mLayoutInflater = LayoutInflater.from(context);
10        this.mPackageManager = context.getPackageManager();
11        this.mApps = mApps;
12    }

```

Le constructeur de l'adaptateur prend en paramètres le contexte de l'application et la liste d'objets `AppInfo` à afficher. Il initialise également un `LayoutInflater` et un `PackageManager` qui seront utilisés respectivement pour créer les vues des éléments de la liste et pour récupérer les icônes des applications.

La classe `AppAdapter`, héritant de la classe `ArrayAdapter` nécessite également l'implémentation de la méthode `getView`. Celle-ci est utilisée par l'adaptateur pour créer une vue pour chaque élément de la liste. Elle prend en paramètres la position de l'élément dans la liste, une vue à réutiliser si disponible et le parent de la vue à créer. Si aucune vue n'est passée en paramètre , une nouvelle est créée à partir du fichier `app_item_layout.xml`. Les champs correspondant au titre, au logo et au mode de vibration sont alors actualisés avec les données de l'objet `AppInfo` à la position spécifiée dans la liste. Le code est le suivant :

```

1 @NonNull
2 @Override
3 public View getView(int position, @Nullable View convertView, @NonNull ViewGroup
4 parent) {
5     AppInfo current = this.mApps.get(position);
6     View v = convertView;
7
8     if (v == null) {
9         v = this.mLayoutInflater.inflate(R.layout.app_item_layout, parent, false);
10    }
11
12    TextView title = v.findViewById(R.id.titleTextView);
13    String titre = current.getLabel();
14    if (titre.length() > 12) {
15        titre = titre.substring(0, 12) + "...";
16    }

```

```

16     title.setText(titre);
17     TextView vibrationModeTextView = v.findViewById(R.id.vibrationModeTextView);
18     if (current.getVibrationMode() == null)
19         vibrationModeTextView.setText("N/A");
20     else
21         vibrationModeTextView.setText(current.getVibrationMode().getName());
22
23     ImageView iconImageView = v.findViewById(R.id.iconImage);
24     Drawable icon = current.getInfo().loadIcon(this.mPackageManager);
25     iconImageView.setImageDrawable(icon);
26
27     return v;
28 }
```

4.3.4 Gestion des modes de vibration

L'objectif étant de permettre à l'utilisateur de notre application de modifier le mode de vibration associé à chaque application via la liste d'applications, il est nécessaire de définir une classe "mode de vibration", et de mettre en place d'une part un système de sauvegarde de données (et donc de chargement), et un moyen de permettre à l'utilisateur de choisir lui-même le mode qu'il souhaite.

→ La classe VibrationMode :

Afin de représenter un mode de vibration, nous avons défini la classe `VibrationMode` suivante (avec ses constructeurs) :

```

1 public class VibrationMode {
2
3     private String name;
4     private String id;
5
6     public VibrationMode(String name, String id) {
7         this.name = name;
8         this.id = id;
9     }
10    public VibrationMode(JSONObject json) {
11        try {
12            this.id = json.getString("id");
13            this.name = json.getString("name");
14        } catch (JSONException e) {
15            e.printStackTrace();
16        }
17    }
}
```

La classe possède deux chaînes de caractères comme attributs : `name`, qui est le nom du mode de vibration, et `id`, qui correspond à l'identifiant de ce dernier. C'est grâce à cet identifiant qu'on va pouvoir associer une application à un mode de vibration, et communiquer avec la carte électronique. A ce stade de développement, il n'existe que trois modes de vibration différents, appelés `Mode 1`, `Mode 2` et `Mode 3`, et ayant respectivement pour identifiant 1, 2 et 3.

On définit également un mode de vibration par défaut, appelé N/A, qui est affecté à une application lorsque aucun autre mode n'y est associé. Il correspond en réalité à aucune vibration du montre, et est créé grâce à cette méthode :

```

1 public static VibrationMode getDefaultVibrationMode() {
2     return new VibrationMode("N/A", "N");
3 }
```

Afin de garder en mémoire les modes de vibration prédéfinis, on utilise le format json et le fichier `SharedPreferences` :

```

1 public JSONObject toJson() {
2     JSONObject json = new JSONObject();
3     try {
4         json.put("id", id);
5         json.put("name", name);
6     } catch (JSONException e) {
7         e.printStackTrace();
8     }
9     return json;
10 }
11 public static VibrationMode fromJson(JSONObject json) {
12     return new VibrationMode(json);
13 }
14 public void saveVibrationMode(Context context) {
15     SharedPreferences sharedpreferences = context.getSharedPreferences("vibration_modes", Context.MODE_PRIVATE);
16     JSONObject json = this.toJson();
17     sharedpreferences.edit().putString(this.getId(), json.toString()).apply();
18 }
```

Pour les charger, il suffit alors de les récupérer et de reconvertis les objets json en instances `VibrationMode`. C'est ce que fait la méthode `getSavedVibrationModes` :

```

1 public static List<VibrationMode> getSavedVibrationModes(Context context) {
2     List<VibrationMode> vibrationModes = new ArrayList<>();
3     SharedPreferences sharedpreferences = context.getSharedPreferences("vibration_modes", Context.MODE_PRIVATE);
4     Set<String> keys = sharedpreferences.getAll().keySet();
5
6     for (String key : keys) {
7         String jsonString = sharedpreferences.getString(key, null);
8         if (jsonString != null) {
9             try {
10                 JSONObject json = new JSONObject(jsonString);
11                 VibrationMode vibrationMode = VibrationMode.fromJson(json);
12                 vibrationModes.add(vibrationMode);
13             } catch (JSONException e) {
14                 e.printStackTrace();
15             }
16         }
17     }
18     return vibrationModes;
19 }
```

La méthode `getSavedVibrationModeFromId` appelle la méthode précédente et permet de récupérer un mode de vibration sauvegardé à partir de son identifiant `id` :

```

1 public static VibrationMode getSavedVibrationModeFromId(Context context, String id) {
2     List<VibrationMode> vibrationModes = getSavedVibrationModes(context);
3     Map<String, VibrationMode> vibrationModeMap = new HashMap<>();
4
5     for (VibrationMode vibrationMode : vibrationModes) {
6         vibrationModeMap.put(vibrationMode.getId(), vibrationMode);
7     }
8
9     return vibrationModeMap.get(id);
10 }
```

→ Affectation des modes de vibration aux applications et enregistrement des données :

Notre première idée, que nous avons décidé de garder et de mettre en place, est l'enregistrement des affectations des modes de vibration aux applications dans un fichier situé dans le dossier `data` de l'application. Nous avions au début choisi d'utiliser un fichier texte, ne connaissant pas particulièrement les autres possibilités, mais nous avons finalement choisi d'utiliser un fichier json (nommé `vibration_modes_data.json`) car il s'agit d'un format simple à manipuler et optimisé (en tout cas pour notre utilisation).

La sauvegarde des données se fait grâce à la méthode

`public void saveAppVibrationMode(String packageName, String vibrationMode)` de la classe `VibrationsTool`. Cette méthode permet de stocker dans le fichier `vibration_modes_data.json` l'association du mode de vibration ayant pour identifiant `vibrationModeId` à l'application ayant pour nom de package `packageName`. Si un mode de vibration est déjà associé à l'application, alors il est remplacé. Voici comment cette méthode s'articule :

```

1 public void saveAppVibrationModeId(String packageName, String vibrationMode) throws
2     JSONException {
3     File file = new File(mContext.get().getFilesDir(), "vibration_modes_data.json");
```

On ouvre le fichier `vibration_modes_data.json`, et on vérifie bien qu'il existe (sinon on le crée) :

```

1     if (!file.exists()) {
2         try {
3             file.createNewFile();
4         } catch (IOException e) {
5             e.printStackTrace();
6         }
7     }
```

On utilise ensuite un `Scanner` et un `StringBuilder` pour transformer le fichier (sous réserve qu'il ne soit pas vide), en objet `JSONObject` qu'on pourra modifier :

```

1  JSONObject root = new JSONObject();
2  try {
3      Scanner sc = new Scanner(file);
4      StringBuilder builder = new StringBuilder();
5      while (sc.hasNextLine()) {
6          builder.append(sc.nextLine());
7      }
8      String fileContents = builder.toString();
9      if (!fileContents.isEmpty()) {
10         root = new JSONObject(fileContents);
11     }
12 } catch (IOException | JSONException e) {
13     e.printStackTrace();
14 }
```

Enfin, on affecte la nouvelle valeur au package packageName dans l'objet JSON, et on met à jour le fichier :

```

1  root.put(packageName, vibrationModeId);
2
3  try {
4      FileWriter writer = new FileWriter(file);
5      writer.write(root.toString());
6      writer.close();
7  } catch (IOException e) {
8      e.printStackTrace();
9  }
10 }
```

Le chargement des données se fait de manière analogue, les mêmes précautions étant prises lors de l'ouverture du fichier. La plupart du temps, lorsque l'application devra charger les données de `vibration_modes_data.json`, elle devra le faire pour toutes les applications : nous implémentons donc une première méthode renvoyant le `JSONObject` construit à partir du fichier `vibration_modes_data.json` contenant toutes les données nécessaires. Cela permet d'optimiser l'exécution en n'ouvrant le fichier qu'une seule fois :

```

1 public JSONObject loadAppsVibrationModes(String notifName, Context context) {
2     File file = new File(context.getFilesDir(), "vibration_modes_data.json");
3
4     if (!file.exists()) {
5         try {
6             file.createNewFile();
7         } catch (IOException e) {
8             e.printStackTrace();
9         }
10    }
11    JSONObject root = new JSONObject();
12    try {
13        Scanner sc = new Scanner(file);
14        StringBuilder builder = new StringBuilder();
15        while (sc.hasNextLine()) {
16            builder.append(sc.nextLine());
17        }
18        root = new JSONObject(builder.toString());
19    }
```

```

19     } catch (IOException | JSONException e) {
20         e.printStackTrace();
21     }
22     return root;
23 }
```

A partir du `JSONObject` obtenu, on utilise simplement la méthode `optString` de la classe `JSONObject` pour récupérer la valeur du mode de vibration associé au nom de package `packageName` (on renvoie "UNKNOWN" si aucune n'est affectée). En voici un exemple :

```

1 public String loadAppVibrationMode(String packageName, Context context) {
2     JSONObject root = loadAppsVibrationModes(context);
3     return root.optString(packageName, "UNKNOWN");
4 }
```

→ Choix des modes de vibration :

Maintenant que le système de sauvegarde des modes de vibration est mis en place, attardons-nous sur le code permettant à l'utilisateur de choisir pour chaque application, le mode de vibration qu'il souhaite. Il est intéressant de rappeler que dans la version actuelle de l'application, il n'existe que 3 modes de vibrations différents (appelés "Mode 1", "Mode 2", et "Mode 3"). Ceux-ci sont initialisés (si pas déjà sauvegardés) lors de l'ouverture de l'application :

```

1 Context context = getApplicationContext();
2 List<VibrationMode> vibrationModes = VibrationMode.getSavedVibrationModes(context);
3 if (vibrationModes.isEmpty()) {
4     VibrationMode mode1 = new VibrationMode("Mode 1", "1");
5     VibrationMode mode2 = new VibrationMode("Mode 2", "2");
6     VibrationMode mode3 = new VibrationMode("Mode 3", "3");
7     mode1.saveVibrationMode(context);
8     mode2.saveVibrationMode(context);
9     mode3.saveVibrationMode(context);
10 }
```

Le choix de l'utilisateur va se faire au travers d'un clic sur n'importe laquelle des applications de la liste d'applications. Il conviendra de revenir en détail sur l'implémentation du `onItemClickListener` dans la partie suivante, mais, lors du clic, une fenêtre de dialogue permettant à l'utilisateur de choisir un mode de vibration s'ouvrira au milieu de l'écran. Cette fenêtre est en réalité une instance de la classe `VibrationModeDialog`, qui hérite de la classe `Dialog` d'Android :

```

1 public class VibrationModeDialog extends Dialog {
2
3     private final AppInfo mAppInfo;
4     private VibrationMode selectedVibrationMode;
5
6     public VibrationModeDialog(Context context, AppInfo mAppInfo) {
7         super(context);
8         this.mAppInfo = mAppInfo;
9     }
```

Cette classe a un attribut `mAppInfo`, qui correspond à l'application sur laquelle l'utilisateur a cliqué, et un attribut `selectedVibrationMode`, qui correspond au mode de vibration choisi par l'utilisateur dans le `Spinner` (que nous allons détailler juste après).

La classe `VibrationModeDialog` nécessite également un `layout`. Afin de préparer d'éventuelles améliorations (notamment celles de rajouter de futurs modes de vibrations), nous mettons en page la fenêtre de dialogue en utilisant un `Spinner`. Un `Spinner` est un widget qui permet à l'utilisateur de choisir une option parmi une liste de choix. L'interface utilisateur affiche un menu déroulant avec une liste d'options, et l'utilisateur peut sélectionner une option en faisant défiler la liste ou en tapant sur l'option désirée. Voici l'aperçu de cette fenêtre et du menu déroulant :

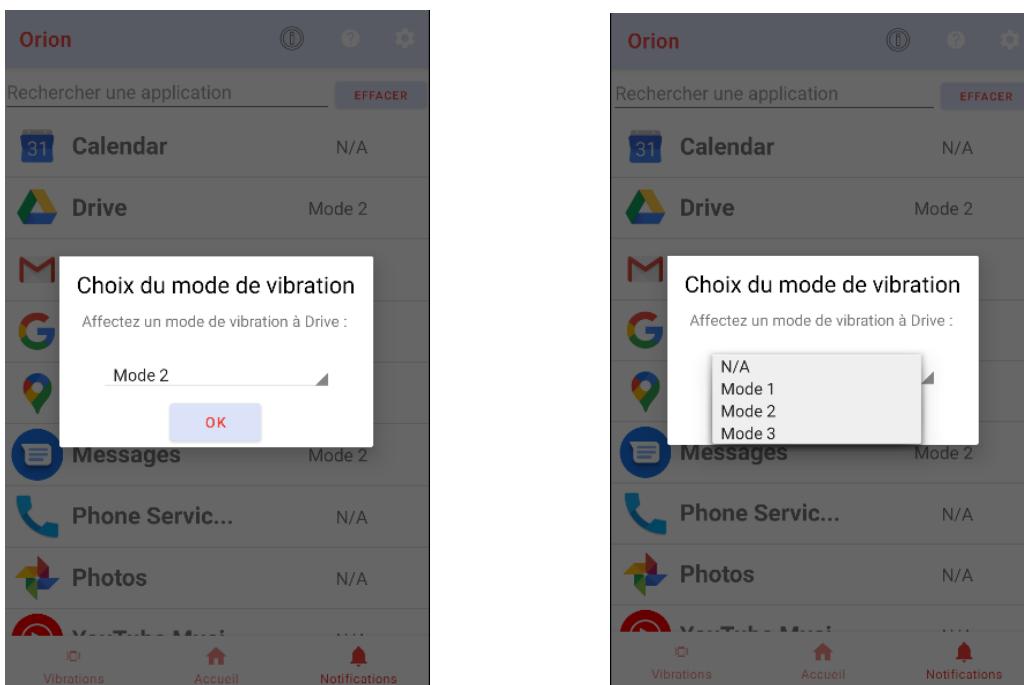


FIGURE 42 – Aperçu de la fenêtre de dialogue et du menu déroulant

Une fois le fichier `.xml` complété, nous pouvons implémenter la méthode `onCreate` de la classe `VibrationModeDialog` :

```

1 @Override
2 protected void onCreate(Bundle savedInstanceState) {
3     super.onCreate(savedInstanceState);
4     requestWindowFeature(Window.FEATURE_NO_TITLE);
5     setContentView(R.layout.dialog_vibration_mode);
6     TextView vibrationModeChoiceApp = findViewById(R.id.vibration_mode_choice_app);
7     Spinner vibrationModeSpinner = findViewById(R.id.vibrationModeSpinner);
8     List<VibrationMode> vibrationModes = VibrationMode.getSavedVibrationModes(
9     getApplicationContext());
10    vibrationModes.add(0,new VibrationMode("N/A",""));
11    ArrayAdapter<VibrationMode> adapter = new ArrayAdapter<>(getContext(), android.R.
12    layout.simple_spinner_item, vibrationModes);
13    adapter.setDropDownViewResource(android.R.layout.simple_spinner_item);
14    vibrationModeSpinner.setAdapter(adapter);
15    VibrationMode currentVibrationMode = mAppInfo.getVibrationMode();

```

```

14     vibrationModeSpinner.setSelection(adapter.getPosition(currentVibrationMode));
15     selectedVibrationMode = currentVibrationMode;

```

Cette méthode permet de générer la vue utilisateur, de créer le **Spinner** via un **ArrayAdapter** et la liste des modes de vibration sauvegardées récupérée via la méthode **getSavedVibrationModes**. Elle fait également en sorte que le mode de vibration sélectionné par défaut soit bien celui associé à l'application (N/A si aucun n'est associé).

On ajoute également un écouteur d'évènement au **Spinner** afin de pouvoir actualiser l'attribut **selectedVibrationMode** dès qu'il est modifié :

```

1 vibrationModeSpinner.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener
2     () {
3         @Override
4         public void onItemSelected(AdapterView<?> adapterView, View view, int i, long l) {
5             selectedVibrationMode = vibrationModes.get(i);
6         }
7         @Override
8         public void onNothingSelected(AdapterView<?> adapterView) {
9             selectedVibrationMode = VibrationMode.getDefaultVibrationMode();
10        }
11    });

```

Enfin, la méthode récupère la référence du bouton "OK" et lorsqu'il est cliqué, affecte le mode de vibration sélectionné par l'utilisateur à l'objet **AppInfo** associé, puis met à jour la liste des applications en appliquant ce changement (à la ligne 25). Le mode de vibration est sauvegardé grâce à la méthode **saveVibrationMode**, et il convient d'indiquer à l'adaptateur de la liste d'applications que les données doivent être rechargés, ce que fait la méthode **notifyDataSetChanged()** de la classe **Adapter** :

```

1 Button vibrationDialogOKButton = findViewById(R.id.vibrationDialogOKButton);
2 vibrationDialogOKButton.setOnClickListener(view -> {
3     mAppInfo.setVibrationMode(selectedVibrationMode);
4     NotificationsFragment.updateItem(mAppInfo);
5     try {
6         MainActivity.getMyVibrationsTool().saveAppVibrationModeId(mAppInfo.getInfo() .
7             packageName, mAppInfo.getVibrationMode().getId());
8     } catch (JSONException e) {
9         e.printStackTrace();
10    }
11    NotificationsFragment.getAdapter().notifyDataSetChanged();
12    dismiss();
13 });

```

4.3.5 Implémentation de la liste d'applications

Les données étant maintenant traitées grâce à la classe **AppAdapter**, et le choix de l'utilisateur étant rendu possible grâce au code de la partie précédente, il ne reste plus qu'à programmer la **ListView** dans le fragment **NotificationsFragment**.

→ Mise en page de la liste :

On commence par ajouter la `ListView`, qu'on appellera par la suite `appListView`, dans le `layout` correspondant :

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     xmlns:tools="http://schemas.android.com/tools"
4     android:layout_width="fill_parent"
5     android:layout_height="fill_parent"
6     android:orientation="vertical"
7     tools:context=".ui.fragments.NotificationsFragment">
8
9
10    <FrameLayout
11        android:layout_width="match_parent"
12        android:layout_height="match_parent"
13        android:layout_marginTop="103dp"
14        android:layout_marginBottom="?attr actionBarSize">
15
16        <ListView
17            android:id="@+id/appList"
18            android:layout_width="match_parent"
19            android:layout_height="match_parent" />

```

La `ListView` est placé au sein d'un `FrameLayout` car ce dernier est utilisé pour la barre de recherche (son fonctionnement sera expliqué par la suite).

→ Génération de la liste d'applications :

Afin de générer la liste d'applications, il est nécessaire de remplir la liste `appList` contenant l'ensemble des données qui sont affichées par la `ListView`. Pour cela, on utilise la classe `LoadAppInfoTask` qui va permettre de créer un thread d'exécution auxiliaire (en arrière-plan).

- Choix des applications :

Un smartphone contient, par défaut, de nombreuses applications systèmes, qui n'agissent pas de manière visible sur le fonctionnement du téléphone, et qui donc, ne sont pas pertinentes dans notre cas. Nous avons donc réduit notre liste d'applications en considérant uniquement les applications installées par l'utilisateur, et celles faisant partie du système et susceptibles d'être utilisés par l'utilisateur. En voici une liste non exhaustive : applications de la suite Google (Drive, Calendar, Youtube, Maps, Gmail...), Messages, Appels... Ce tri est effectué grâce à la méthode `filter(ApplicationInfo appInfo)` :

```

1 protected boolean filter(ApplicationInfo appInfo) {
2     Set<String> allowedApps = new HashSet<>(Arrays.asList(
3         "com.google.android.apps.docs",
4         "com.google.android.gm",
5         "com.google.android.googlequicksearchbox",
6         "com.google.android.calendar",
7         "com.google.android.chrome",
8         "com.google.android.apps.deskclock",
9         "com.google.android.apps.maps",
10        "com.google.android.apps.messaging",
11        "com.android.phone",

```

```

12     "com.google.android.apps.photos",
13     "com.google.android.apps.youtube",
14     "com.google.android.apps.youtube.music"
15 );
16
17     if (appInfo.packageName.equals("com.example.tactigant20")) {
18         return false;
19     } else if (allowedApps.contains(appInfo.packageName)) {
20         return true;
21     } else {
22         return ((appInfo.flags & ApplicationInfo.FLAG_SYSTEM) != ApplicationInfo.
23 FLAG_SYSTEM);
24 }

```

- Exécution du thread auxiliaire :

La classe `LoadAppInfoTask` permettant de générer la liste d'applications se base sur trois méthodes : `OnPreExecute()`, `OnExecute()` et `OnPostExecute()`. Comme leurs noms l'indiquent, les trois méthodes sont respectivement appelées avant, pendant et après l'exécution du thread. La méthode `onPreExecute()` ne nous intéresse pas, nous la laissons donc vide :

```

1 public class LoadAppInfoTask extends AsyncTask<Integer, Integer, List<AppInfo>> {
2
3     @Override
4     protected void onPreExecute() {
5         super.onPreExecute();
6     }

```

Le tri et la récupération des modes de vibration des applications se fait donc naturellement dans la méthode `onExecute`. Pour cela, on récupère tout d'abord le `PackageManager` afin d'obtenir une liste d'objets `ApplicationInfo` représentant l'ensemble des applications installées sur le téléphone. On charge ensuite les données sauvegardées à l'aide de la méthode `loadVibrationModes()`, et on itère sur chaque élément de la liste. Si la liste correspond aux critères décrits précédemment, alors on crée une instance de la classe `AppInfo`, pour laquelle on récupère le mode de vibration et on l'ajoute à la liste `appList`. Voici le code de cette méthode :

```

1 @RequiresApi(api = Build.VERSION_CODES.O)
2 @Override
3 protected List<AppInfo> doInBackground(Integer... params) {
4     PackageManager packageManager = requireContext().getPackageManager();
5     List<ApplicationInfo> infos = packageManager.getInstalledApplications(
6         PackageManager.GET_META_DATA);
7     JSONObject root = MainActivity.getMyVibrationsTool().loadAppsVibrationModes(
8         requireContext());
9     for (ApplicationInfo info : infos) {
10         if (filter(info)) {
11             AppInfo app = new AppInfo();
12             app.setInfo(info);
13             app.setLabel((String) info.loadLabel(packageManager));
14             String vibrationModeId = root.optString(info.packageName, "N");
15             app.setVibrationModeId(vibrationModeId);
16             appList.add(app);
17         }
18     }
19 }

```

```

13         VibrationMode vibrationMode = VibrationMode.getSavedVibrationModeFromId(
14             getContext(), vibrationModeId);
15             app.setVibrationMode(vibrationMode);
16             appList.add(app);
17         }
18     }
19
20     Collections.sort(appList, Comparator.comparing(AppInfo::getLabel));
21
22     return appList;
23 }
```

Une fois la liste créée avec les applications que l'on souhaite, il nous reste plus qu'à affecter un **AppAdapter** à la **ListView** **appListView** :

```

1 @Override
2 protected void onPostExecute(List<AppInfo> appInfos) {
3     super.onPostExecute(appInfos);
4     searchedAppsList = appList;
5     adapter = new AppAdapter(getContext(), searchedAppsList);
6     appListView.setAdapter(adapter);
7 }
```

→ La classe **NotificationsFragment** :

Le code suivant définit la classe **NotificationsFragment**, qui va donc contenir la liste des applications ainsi que la barre de recherche :

```

1 public class NotificationsFragment extends Fragment {
2
3     private static List<AppInfo> appList;
4     private static List<AppInfo> searchedAppList;
5     private static AppAdapter adapter;
6     private static boolean dataReinitialised;
7     private ListView appListView;
```

Elle possède un certain nombre d'attributs, qu'il convient de détailler :

- **List<AppInfo> appList** est la liste de l'ensemble des applications filtrées du téléphone, triée par ordre de *label* alphabétique,
- **searchedAppList** est la liste auxiliaire utilisée par la **ListView** pour afficher les applications selon la recherche utilisateur (si aucune recherche n'est faite, on a **searchedAppList = appList**).
- **AppAdapter adapter** est l'adaptateur associé à la **ListView**,
- **dataReinitialised** est un booléen valant **true** lorsque les données ont été réinitialisées dans les paramètres mais pas encore mises à jour,
- **ListView appListView** est la **ListView** affichant l'ensemble des applications de **appList**.

La méthode **updateItem(AppInfo appInfo)** permet de mettre à jour dans **appList** l'objet ayant le même **label** que le paramètre **appInfo** en le remplaçant par ce dernier :

```

1 public static void updateItem(AppInfo appInfo) {
2     String appLabel = appInfo.getLabel();
3     int i = 0;
4     if (!appList.isEmpty()) {
5         while (i < appList.size() && !appList.get(i).getLabel().equals(appLabel))
6             i++;
7     }
8     appList.set(i, appInfo);
9 }
```

Une autre méthode utilisée dans la fonction principale du fragment `NotificationsFragment` est la suivante :

```

1 public void createNewVibrationModeDialog(AppInfo appInfo) {
2     final VibrationModeDialog dialog = new VibrationModeDialog(this.getContext(),
3         appInfo);
4     dialog.show();
5 }
```

Celle-ci crée et affiche simplement la fenêtre de dialogue permettant de faire le choix du mode de vibration.

La méthode `onCreateView` est appelée lors de la création de la vue du fragment. Nous l'utilisons afin d'initialiser la liste `appList`, et de la remplir en exécutant une tâche, instance de `LoadAppInfoTask` :

```

1 public View onCreateView(@NonNull LayoutInflater inflater, ViewGroup container, Bundle
2     savedInstanceState) {
3
4     com.example.tactigant20.databinding.FragmentNotificationsBinding binding =
5     FragmentNotificationsBinding.inflate(inflater, container, false);
6     View root = binding.getRoot();
7
8     appListView = root.findViewById(R.id.appList);
9     appListView.setTextFilterEnabled(true);
10    TextView noResultsFound = root.findViewById(R.id.no_results_text);
11    appListView.setEmptyView(noResultsFound);
12    appList = new ArrayList<>();
13    LoadAppInfoTask task = new LoadAppInfoTask();
14    task.execute();
```

Enfin, on initialise l'écouteur d'évènement `onItemClickListener` pour `appListView` :

```

1 appListView.setOnItemClickListener((adapterView, view, i, l) -> {
2     AppInfo currentItem = (AppInfo) appListView.getItemAtPosition(i);
3     createNewVibrationModeDialog(currentItem);
4     getAdapter().notifyDataSetChanged();
5 });
```

Lorsqu'un élément de `appListView` est cliqué, on appelle naturellement la méthode `createNewVibrationModeDialog` pour l'élément sélectionné (`currentItem`), ce qui permet de modifier le mode de vibration associé.

Finalement, on obtient bien la liste d'applications souhaitées, triée par ordre alphabétique avec leurs modes de vibration, et la capacité de les modifier :

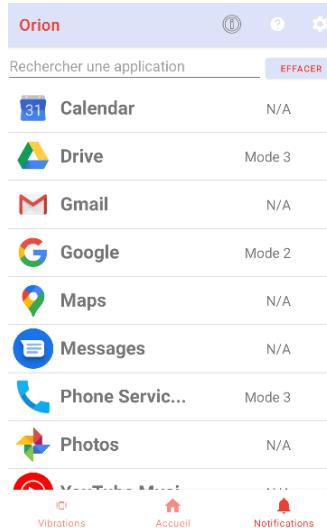


FIGURE 43 – Aperçu du fragment *Notifications*

4.3.6 Ajout d'une barre de recherche

Dans l'optique de rendre la navigation dans la liste d'applications plus ergonomique et plus facile, nous avons eu l'idée d'ajouter une barre de recherche au-dessus de la liste dans le fragment `NotificationsFragment`. L'utilisateur peut alors rechercher et accéder facilement à une application spécifique.

→ Modification de la mise en page :

Afin de créer cette barre de recherche, nous allons insérer un widget `EditText` dans le fichier `fragment_notifications.xml`, ainsi qu'un bouton permettant d'effacer la recherche :

```

1 <EditText
2     android:id="@+id/app_search_bar"
3     android:layout_width="317dp"
4     android:layout_height="wrap_content"
5     android:layout_alignParentTop="true"
6     android:layout_marginTop="57dp"
7     android:hint="@string/search_app"
8     android:autofillHints=""
9     android:inputType="text"/>
10 <Button
11     android:id="@+id/delete_button"
12     android:layout_width="wrap_content"
13     android:layout_height="37dp"
14     android:layout_alignTop="@+id/app_search_bar"
15     android:layout_marginStart="3dp"
16     android:layout_marginTop="7dp"
17     android:layout_toEndOf="@+id/app_search_bar"
18     android:text="@string/delete"
19     android:textSize="11sp" />
```

Voici le rendu de la barre de recherche :



FIGURE 44 – Aperçu de la mise en page de la barre de recherche

Nous avons également rajouté dans le `FrameLayout` contenant la `ListView` un `TextView` qui correspond à ce qu'affichera la liste dans le cas où la recherche de l'utilisateur ne donne aucun résultat :

```

1 <TextView
2     android:id="@+id/no_results_text"
3     android:layout_width="wrap_content"
4     android:layout_height="wrap_content"
5     android:layout_gravity="center"
6     android:text="@string/no_results_app"
7     android:textAlignment="center"
8     android:textSize="17sp"
9     android:visibility="gone" />

```

→ Adaptation du code de NotificationsFragment :

Afin de mettre en place la fonctionnalité de la barre de recherche, il faut rajouter l'attribut `searchedAppsList`, qui correspond à la liste des applications recherchées. Celle-ci est égale à `appList` si la barre de recherche est vide, et contient l'ensemble des fonctions commençant par la chaîne de caractères inscrite dans la barre de recherche (majuscules non prises en compte). Ceci est implémenté au sein de la méthode `onCreateView` :

```

1 EditText editText = root.findViewById(R.id.app_search_bar);
2 editText.setInputType(InputType.TYPE_TEXT_FLAG_MULTI_LINE | InputType.TYPE_CLASS_TEXT)
3 ;
4 editText.setInputType(editText.getInputType() & ~InputType.TYPE_TEXT_FLAG_MULTI_LINE);
5 editText.addTextChangedListener(new TextWatcher() {
6     @Override
7     public void beforeTextChanged(CharSequence charSequence, int i, int i1, int i2) {}
8     public void filterListviewItems(String s) {
9         searchedAppsList = new ArrayList<>();
10        s = s.toLowerCase();
11        int n = s.length();
12        for (AppInfo app : appList) {
13            String name = app.getLabel().toLowerCase();
14            if (n < name.length() && name.substring(0,n).equals(s))
15                searchedAppsList.add(app);
16        }
17    }

```

```

17     @Override
18     public void onTextChanged(CharSequence charSequence, int i, int i1, int i2) {
19         if (charSequence == null) {
20             searchedAppsList = appList;
21         }
22         else {
23             filterListviewItems(charSequence.toString());
24         }
25         adapter = new AppAdapter(getContext(), searchedAppsList);
26         appListView.setAdapter(adapter);
27         getAdapter().notifyDataSetChanged();
28     }
29     @Override
30     public void afterTextChanged(Editable editable) {}
31 });

```

On affecte à l'`EditText` représentant la barre de recherche un `onTextChangedListener`. Quand le texte est modifié (et le contenu non vide), on appelle la méthode `filterListviewItems` qui va garder uniquement les applications satisfaisant le critère précédemment cité. Enfin, on affecte à `appListView` un nouvel adaptateur associé à la liste `searchedAppsList`, et on met à jour la `ListView`.

On ajoute également à l'aide de la méthode `ListView.setEmptyView` la vue de `appListView` si celle-ci est vide (juste après l'initialisation de `appListView`) :

```

1 TextView noResultsFound = root.findViewById(R.id.no_results_text);
2 appListView.setEmptyView(noResultsFound);

```

Voici un aperçu de l'affichage lorsque la liste est vide :



Aucune application ne correspond à votre recherche.



FIGURE 45 – Aperçu de l'affichage d'une recherche sans résultat

Enfin, on configure le bouton "Effacer" très simplement :

```
1 Button deleteButton = root.findViewById(R.id.delete_button);  
2 deleteButton.setOnClickListener(view -> editText.setText(""));
```

4.3.7 Résultats

Dans cette partie, nous avons mis en lumière notre manière d'implémenter d'une part une liste d'applications sélectionnées selon nos critères (facilement modifiables), et d'autre part, un système de sauvegarde des données de l'application (grâce à des fichiers json). Il existe sans aucun doute de nombreuses autres façons de programmer ceci, et l'expérience de développeur est sûrement la clé pour choisir la plus appropriée. Néanmoins, notre solution reste assez optimisée et suffisamment "simple" pour permettre de futures améliorations, telles que le choix entre un nombre indéfini de modes de vibration, ou des recherches plus avancées (selon d'autres filtres).

4.4 Communication Bluetooth Low Energy

4.4.1 Contexte

La Bluetooth Low Energy (abrégée BLE), est une technique de transmission sans fil créée en 2006 par Nokia. Cette nouvelle technologie se base sur celle existante du Bluetooth et la complète sans pour autant la remplacer. La principale différence avec cette dernière est sa consommation d'énergie qui est 10 fois inférieure. Cette différence permet d'intégrer la technologie Bluetooth à de tout nouveaux types d'équipements tels que des bracelet et autres **montres connectées**. La technologie BLE est intégrée aux normes Bluetooth depuis la version 4.0, elle accompagne ainsi aujourd'hui la quasi-totalité du marché Android.

Dans le cadre de notre projet nous avons utilisé une Arduino ESP32 (une carte électronique dotée de BLE) ainsi qu'un téléphone Android Samsung J3 2017 (tournant sous Android 9) pour faire nos tests. Tout ce qui est présenté ici est commun à tout type de support similaire.

4.4.2 Les permissions nécessaires

Avant de rentrer dans le vif du sujet, il est nécessaire d'aborder le thème des permissions Android. En effet, si vous voulez que tout fonctionne correctement chez vous il est nécessaire de faire quelques manipulations.

Premièrement sur Android Studio, dans votre **manifest** il est nécessaire de rajouter les permissions de Bluetooth et de localisation suivantes :

```

1 <uses-permission android:name="android.permission.BLUETOOTH" />
2 <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
3 <uses-permission android:name="android.permission.BLUETOOTH_CONNECT" />
4 <uses-permission android:name="android.permission.BLUETOOTH_SCAN" />
5 <uses-permission android:name="android.permission.BLUETOOTH_ADVERTISE" />
6 <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
```

Notre application fonctionne sans la permission suivante. Celle-ci est néanmoins souvent mentionnée dans les documentations :

```
1 <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

Également lorsque l'application sera installée sur l'appareil Android, il faudra l'autoriser à utiliser la localisation ainsi qu'à se connecter à des appareils à proximité. Normalement l'application demande si elle peut activer par elle-même ces permissions. Dans le cas contraire, vous pouvez le faire manuellement en vous rendant dans les paramètres de l'application (cf figure 46.).

Une application n'a en aucun cas besoin de connaître la position de l'appareil pour utiliser le BLE. Accorder cette dernière permission est imposé par Android afin de représenter le fait qu'il est en théorie possible de retrouver la position dudit appareil en analysant les trames BLE.



FIGURE 46 – Capture d'écran illustrant les autorisations à activer.

4.4.3 Détection des appareils BLE à proximité

Passons désormais à notre première partie concernant le scan des appareils BLE proches. Lorsqu'on parle de scan, on parle en réalité de la recherche d'appareils BLE aux alentours. Attention ici on ne parle pas de connexion à un appareil mais bien uniquement de recherche.

Vous aurez alors besoin d'un objet `scanner`; qui va venir contrôler la recherche de votre appareil Android sur les services Bluetooth à proximité; et qui se forme à la l'aide d'un `adapter` :

```
1 private BluetoothAdapter adapter = BluetoothAdapter.getDefaultAdapter();
2 private BluetoothLeScanner scanner = adapter.getBluetoothLeScanner();
```

Vous pouvez (et devez !) personnaliser votre scan de deux manières différentes :

1. A l'aide de filtres qui vont venir présélectionner le genre d'appareils que vous souhaitez rechercher. Vous pouvez filtrer par nom, adresse MAC ou UUID¹. Vous pouvez aussi très bien fournir la valeur `null` à votre filtre si vous souhaitez étudier tous les services Bluetooth aux alentours. Dans notre cas, nous avons choisi de filtrer par adresse MAC :

```
1 if (scanner != null) {
2     String[] peripheralAddresses = new String[]{"94:3C:C6:06:CC:1E"};
3     List<ScanFilter> filters = null;
4     if (peripheralAddresses != null) {
5         filters = new ArrayList<>();
6         for (String address : peripheralAddresses) {
7             ScanFilter filter = new ScanFilter.Builder()
8                 .setDeviceAddress(address)
9                 .build();
10            filters.add(filter);
11        }
12    }
```

Vous trouverez en annexe du code (non testé) pour les autres types de filtres.

1. UUID : Universally Unique IDentifier, un nombre codé sur 128 bits qui sert à identifier un service ou une caractéristique.

2. Mais aussi à l'aide de paramètres qui permettent d'expliciter le type de comportement que vous souhaitez avoir pour votre scan. Ces paramètres sont nombreux : mode de connexion, consommation d'énergie... Nous recommandons les paramètres suivants qui fonctionneront pour la majorité des projets :

```

1 ScanSettings scanSettings = new ScanSettings.Builder()
2     .setScanMode(ScanSettings.SCAN_MODE_LOW_POWER)
3     .setCallbackType(ScanSettings.CALLBACK_TYPE_ALL_MATCHES)
4     .setMatchMode(ScanSettings.MATCH_MODE_AGGRESSIVE)
5     .setNumOfMatches(ScanSettings.MATCH_NUM_ONE_ADVERTISEMENT)
6     .setReportDelay(0L)
7     .build();

```

Une fois ces deux points effectués il ne reste plus qu'à lancer le scan :

```
1 scanner.startScan(filters, scanSettings, scanCallback);
```

Si le scan réussit (c'est-à-dire qu'il ne rencontre aucune erreur due au `scanner`) , le code appelle automatiquement l'objet `scanCallback`. Ce `scanCallback` permet de traiter différents cas en fonction de comment le scan s'est comporté après son lancement. Chaque cas est caractérisé par une fonction associée qui sera appelée si le scan correspond à cette configuration. Si le scan réussit c'est la fonction `onScanResult()` qui sera appelée, s'il échoue ce sera la fonction `onScanFailed()`.

Le code s'organise ainsi de la manière suivante :

```

1 private final ScanCallback scanCallback = new ScanCallback() {
2     @RequiresApi(api = Build.VERSION_CODES.M)
3     @SuppressLint("MissingPermission")
4     @Override
5     public void onScanResult(int callbackType, ScanResult result) {
6         // Votre code en cas de succès
7     }
8     @Override
9     public void onScanFailed(int errorCode) {
10        // Votre code en cas d'échec
11    }
12 };

```

Pour la fonction `onScanFailed()`, un simple message d'erreur dans le logcat peut suffire :

```

1     @Override
2     public void onScanFailed(int errorCode) {
3         Log.e(TAG_HOME_BLE, "ERREUR : scan (onScanFailed)");
4     }

```

Par contre pour la fonction `onScanResult()`, le travail est plus compliqué : c'est cette fonction qui va vous servir pour connecter votre carte à votre appareil Android. C'est ce que nous allons développer dans la prochaine partie.

4.4.4 Connexion et déconnexion de la carte à l'appareil Android

Abordons les sujets de la connexion (qui permettra plus tard de faire communiquer carte et appareil Android) et de la déconnexion.

Comme expliqué dans la partie précédente, nous allons dans un premier temps détailler la fonction `onScanResult()`. Pour cela il faut instancier un objet `gatt`, qui définit la connexion entre deux appareils :

```
1 BluetoothGatt gatt = device.connectGatt(getContext(), true, bluetoothGattCallback,
    TRANSPORT_LE);
```

Le premier paramètre de cette fonction est le contexte dont Android a besoin pour faire la connexion (`getContext()` suffit). Le second paramètre indique si l'on souhaite une connexion immédiate ou non : en choisissant `true` Android se connectera dès l'instant où il aura détecté l'appareil Bluetooth, en choisissant `false` Android se connectera après un certain délai (30 secondes pour la plupart des appareils). Le troisième paramètre est un objet de rappel (fonctionne de manière similaire à l'objet `scanCallback`). Le dernier concerne le type de transport de données que l'on souhaite dans notre connexion. Cependant la manière dont fonctionne ce paramètre n'est pas très claire et dépend du support Android utilisé. Il est donc fortement recommandé de toujours utiliser le paramètre `TRANSPORT_LE` qui fonctionne peu importe la situation.

Le code s'organise de la manière suivante :

```
1 @Override
2 public void onScanResult(int callbackType, ScanResult result) {
3     BluetoothDevice device = result.getDevice();
4     gatt = device.connectGatt(getContext(), true, bluetoothGattCallback, TRANSPORT_LE)
5         ;
```

Si l'objet `gatt` réussit, on appelle l'objet `bluetoothGattCallback`, qui vient lui-même appeler une fonction `onConnectionStateChange()`. Cette dernière permet de détecter chaque changement de connexion (et sera appelée automatiquement à chaque changement).

Dans le cas de base ou vous essayez de connecter vos deux appareils (à l'aide de la fonction `connectGatt()` détaillée dans le code précédent) et que ces derniers ne sont pas appairés il y a deux cas possibles : soit une erreur quelconque, soit votre état de connexion change et passe à *connecté*. Cependant puisque la fonction `onConnectionStateChange()` est appelée à chaque changement de connexion. Il se peut aussi que vos deux appareils soient appairés et que vous demandiez une déconnexion (fonctionnement détaillé ci-après), il y a là aussi deux cas : soit une erreur quelconque, soit votre état de connexion change et passe à *déconnecté*.

Le tout s'organise de la manière suivante :

```
1 private final BluetoothGattCallback bluetoothGattCallback = new BluetoothGattCallback
() {
2     @SuppressLint("MissingPermission")
3     @Override
4     public void onConnectionStateChange(final BluetoothGatt gatt, final int status
5         , final int newState) {
6         if(status == GATT_SUCCESS) { // On s'est connecté à un appareil
```

```

6         if (newState == BluetoothProfile.STATE_CONNECTED) {
7             // Connexion reussie, vous pouvez utiliser le service
8             Log.d(TAG_HOME_BLE, "CONNECTE");
9         } else if (newState == BluetoothProfile.STATE_DISCONNECTED) {
10            // Deconnection reussie
11            gatt.close();
12            Log.d(TAG_HOME_BLE, "DECONNECTE");
13        }
14    } else {
15        Log.e(TAG_HOME_BLE, "ERREUR : pas de connexion etablie (
16 onConnectionStateChanged");
17        gatt.close();
18    }

```

Pour se déconnecter, il faut d'une part demander l'arrêt du `gatt`, mais aussi demander un arrêt du `scan` (sans quoi la connexion pourrait se refaire immédiatement). On procède de la sorte :

```

1 gatt.disconnect();
2 scanner.stopScan(scanCallback);

```

Ce travail permet de connecter et déconnecter correctement les deux appareils.

4.4.5 Echanges entre la carte et l'appareil Android

Dans cette dernière partie nous allons voir comment envoyer des informations du support Android vers la carte et inversement.

Dès que vous souhaitez faire une de ces deux actions (lecture ou écriture) il est nécessaire d'appeler la même fonction `onServicesDiscovered()`. C'est cette dernière qui viendra gérer les deux cas. Cependant puisque rien ne diffère dans l'appel de ces actions c'est à vous de gérer manuellement cette différence (à l'aide d'un `if` par exemple). Pour appeler cette fonction `onServicesDiscovered()` il suffit d'écrire :

```

1 gatt.discoverServices();

```

Le code pour la fonction `onServicesDiscovered()` est le suivant :

```

1 @SuppressLint("MissingPermission")
2     @Override
3     public void onServicesDiscovered(BluetoothGatt gatt, int status) {
4         if (status == BluetoothGatt.GATT_SUCCESS) {
5             List<BluetoothGattService> services = gatt.getServices();
6             for (BluetoothGattService service : services) {
7                 List<BluetoothGattCharacteristic> characteristics = service.
8                 getCharacteristics();
9                 for (BluetoothGattCharacteristic characteristic : characteristics)
10                {
11                    // On parcourt l'ensemble des caracteristiques trouvees
12                    if(btn.equals("Lecture")){
13                        Log.d(TAG_HOME_BLE, "On recoit quelque chose de la carte !
14                    });
15                    characteristic.getValue();
16                }
17            }
18        }
19    }

```

```

13                     gatt.readCharacteristic(characteristic);
14                 }
15             if (btn.equals("Ecriture")) {
16                 Log.d(TAG_HOME_BLE, "On envoie quelque chose à la carte !");
17             chain a la carte
18             characteristic.setValue("Allume"); // On envoie cette
19             characteristic.setWriteType(BluetoothGattCharacteristic.
20                 WRITE_TYPE_DEFAULT);
21             gatt.writeCharacteristic(characteristic);
22         }
23     }
24 }
```

Pour chaque action on travaille avec un objet de type `BluetoothGattCharacteristics` qui représente les données qui sont échangées par BLE. Détaillons le fonctionnement de ces deux actions en expliquant le code de la fonction `onServicesDiscovered()` :

1. Pour envoyer des informations à la carte c'est la méthode `setValue()` qu'il faut utiliser. Ainsi la ligne de code 17 permet d'envoyer vers la carte la chaîne de caractères « Allume » (vous pouvez bien évidemment envoyer n'importe quelle chaîne). Les deux lignes qui suivent (18 et 19) servent à expliciter clairement l'action à faire lors d'un envoi de donnée. Elles font appel à une fonction `onCharacteristicWrite()` définie juste après (à noter que dans notre cas on se contente d'afficher un message dans le logcat).

```

1 @Override
2 public void onCharacteristicWrite(BluetoothGatt gatt,
3     BluetoothGattCharacteristic characteristic, int status) {
4     super.onCharacteristicWrite(gatt, characteristic, status);
5     Log.d(TAG_HOME_BLE, "UUID de la caractéristique : " + characteristic
6         .getUuid());
7 }
```

2. Le fonctionnement est similaire au cas précédent : pour lire des informations c'est une méthode `get` qu'il faut utiliser (les informations que l'on peut recevoir depuis la carte sont limitées, vous trouverez en annexe une liste des méthodes possibles). Dans notre cas, la ligne de code 11 permet de récupérer la valeur de la carte (c'est dans le code Arduino présent sur la carte que l'on peut modifier cette valeur). La ligne qui suit (12) permet d'expliquer la procédure à faire pour une réception de donnée. Elle fait appel à une fonction `onCharacteristicRead()` définie juste après. On remarque que cette fonction effectue notamment un décryptage de l'information reçue pour la rendre utilisable par l'application.

```

1 // Lecture d'informations depuis la carte
2 @Override
3 public void onCharacteristicRead(BluetoothGatt gatt, final
4     BluetoothGattCharacteristic characteristic, int status) {
5     // Vérification que ça a fonctionné
6     if (status != GATT_SUCCESS) {
```

```
6         Log.e(TAG_HOME_BLE, String.format(Locale.FRENCH,"ERREUR de
7         lecture pour la caracteristique %s, status %d (onCharacteristicRead)",
8         characteristic.getUuid(), status));
9         return;
10    }
11
12    // Traitement de la caracteristique pour la rendre lisible (byte ->
13    String)
14    byte[] value = characteristic.getValue();
15    String s = new String(value, StandardCharsets.UTF_8);
16    Log.d(TAG_HOME_BLE, "La carte nous envoie : \"" + s + "\"");
17
18 }
```

4.4.6 Résultats

Nous avons pu effectuer un certains nombre de tests qui témoignent du bon fonctionnement de l'ensemble :

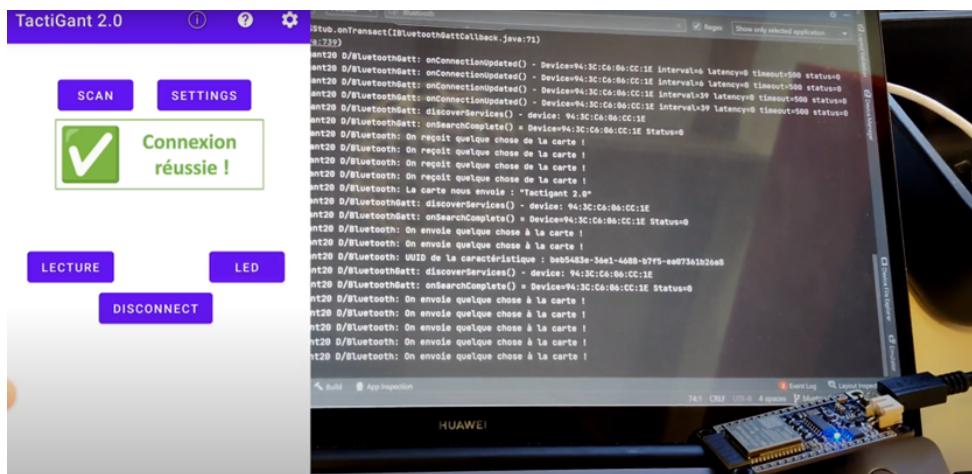


FIGURE 47 – Capture d'écran d'une démonstration de notre prototype (retrouvez la vidéo complète [ici](#))

Des problèmes au bon fonctionnement de la communication ont pu être remarqué avec certaines versions d'Android ou certain système d'exploitation. Cependant cela reste des problèmes mineurs.

4.5 Gestion des notifications du téléphone

4.5.1 Contexte

Le but de l'application Orion est de récupérer les notifications du téléphone afin d'envoyer un message vers la montre si la notification provient d'une application à laquelle est associé un mode de vibration. Cette partie présente la solution adoptée pour récupérer et traiter les notifications.

4.5.2 Service

Rappelons tout d'abord ce qu'est un service. Un service est un composant Android qui s'exécute sans interface graphique. Plus simplement, un service fonctionne en arrière-plan sur le téléphone et ne permet pas d'interaction avec l'utilisateur. Or la récupération et le traitement des notifications ne nécessite pas d'interaction et ne doit pas forcer l'utilisateur à rester sur l'application. L'utilisation d'un service est donc adaptée à la gestion des notifications.

Dans le cadre de notre application, nous avons utilisé un type particulier de service : `NotificationListenerService` qui permet de suivre les notifications du téléphone. Pour créer notre service, nous avons créé la classe `MyNotificationListenerService` qui hérite de la classe `NotificationListenerService`. A noter que la classe `MyNotificationListenerService` est créée dans un fichier à part (le service n'est pas déclaré dans une activité).

```

1 import android.service.notification.NotificationListenerService;
2 public class MyNotificationListenerService extends NotificationListenerService{}
```

Plusieurs imports sont nécessaires au début du fichier `MyNotificationListenerService.java`:

```

1 import android.app.Notification;
2 import android.bluetooth.BluetoothGatt;
3 import android.content.Context;
4 import android.service.notification.NotificationListenerService;
5 import android.service.notification.StatusBarNotification;
```

4.5.3 Autorisations requises

Pour pouvoir étendre la classe `NotificationListenerService` il faut déclarer ce service dans la rubrique "application" du manifest du projet.

```

1 <service
2     android:name=".MyNotificationListenerService"
3     android:label="@string/service_name"
4     android:exported="true"
5     android:permission="android.permission.BIND_NOTIFICATION_LISTENER_SERVICE">
6         <intent-filter>
7             <action
8                 android:name="android.service.notification.NotificationListenerService"/>
9         </intent-filter>
10    </service>
```

Ensuite, notre application souhaite accéder aux notifications du téléphone de l'utilisateur. Sous Android, il est nécessaire de demander l'autorisation à l'utilisateur. Pour s'assurer de cela, lorsque l'application est lancée et que `MainActivity` est créée, on vérifie (dans le fichier `MainActivity.java`) que le service est en train de tourner en arrière-plan. Si ce n'est pas le cas, cela signifie que l'utilisateur n'a pas autorisé l'application à accéder aux notifications de son téléphone. Le cas échéant, on demande à l'utilisateur de donner son autorisation.

```

1 if (!isNotificationServiceRunning()) {
2     AlertDialog.Builder builder = new AlertDialog.Builder(this);
3     builder.setTitle("Information");
4     builder.setMessage(this.getResources().getString(R.string.text_perm_notifications));
5     builder.setPositiveButton("OK", (dialog, which) -> startActivity(new Intent(
6         Settings.ACTION_NOTIFICATION_LISTENER_SETTINGS)));
7     builder.setNegativeButton("Non", (dialog, which) -> dialog.cancel());
8     AlertDialog alertDialog = builder.create();
9     alertDialog.show();
}

```



FIGURE 48 – Demande d'autorisation d'accéder aux notifications

4.5.4 Récupération et traitement des notifications

La récupération des notifications est réalisée par l'intermédiaire de la méthode `onNotificationPosted(StatusBarNotification sbn)` qui est appelée lorsque une notification est reçue. L'argument `sbn` est une structure de donnée contenant à la fois l'objet `Notification` qui a été reçu ainsi que les informations permettant de l'identifier : `tag`, `id` et le nom du *package* dont la notification provient.

Une fois la notification récupérée, on doit la traiter. Pour cela, on récupère sa `category`. Cela nous permet de filtrer les notifications provenant du système qui n'ont pas d'intérêt pour l'application présentée. Le `packageName` permet de connaître l'application dont provient la notification. Cela nous permet donc de connaître, grâce à la méthode `loadVibrationMode`, le mode de vibration à utiliser pour le montre. On souhaite ensuite envoyer le mode de vibration vers la carte. La méthode `discoverServices` fait appel à la méthode `onServicesDiscovered(BluetoothGatt gatt, int status)` (dans le fichier `BluetoothLowEnergyTool.java`) qui choisit le message à envoyer vers la carte en fonction du mode de vibration.

```

1 @Override
2     public void onNotificationPosted(StatusBarNotification sbn) {
3         if (MainActivity.getMyBLET() != null) {

```

```

4         if (MainActivity.getMyBLET().getValeurDeConnexion() ==
5             BluetoothLowEnergyTool.ValeurDeConnexion.CONNECTE) {
6             super.onNotificationPosted(sbn);
7             String packageName = sbn.getPackageName();
8             Notification notif = sbn.getNotification();
9             String category = notif.category;
10            mVibrationMode = MainActivity.getMyVibrationsTool().loadVibrationMode(
11                packageName, getApplicationContext());
12            BluetoothGatt gatt = MainActivity.getMyBLET().getGatt();
13            MainActivity.getMyBLET().setMode("Ecriture");
14            if (category != null) {
15                if (!category.equals("sys")) {
16                    try {
17                        gatt.discoverServices();
18                    } catch (SecurityException e) {
19                        e.printStackTrace();
20                    }
21                }
22            }
23        }

```

Notons la référence au BLE (*Bluetooth Low Energy*) au début de la méthode. Cela permet de vérifier l'état de la connexion. Si le téléphone est connecté à un appareil BLE, alors on peut traiter les notifications, sinon on ne le fait pas.

```

1 @Override
2 public void onServicesDiscovered(BluetoothGatt gatt, int status) {
3     if (status == BluetoothGatt.GATT_SUCCESS) {
4         List<BluetoothGattService> services = gatt.getServices();
5         for (BluetoothGattService service : services) {
6             List<BluetoothGattCharacteristic> characteristics = service.
7                 getCharacteristics();
8             for (BluetoothGattCharacteristic characteristic : characteristics) {
9                 if (mMode.equals("Lecture")) {
10                     characteristic.getValue();
11                     try {
12                         gatt.readCharacteristic(characteristic);
13                     } catch (SecurityException e) {
14                         e.printStackTrace();
15                     }
16                 if (mMode.equals("Ecriture")) {
17                     switch(MyNotificationListenerService.getVibrationMode()) {
18                         case "1":
19                             SendData(gatt, characteristic, "Allume 1");
20                             // On fait de même pour les autres modes
21                             break;
22                         }
23                     }
24                 }
25             }

```

4.5.5 Résultats

Ainsi, grâce à ce travail on arrive à récupérer les informations relatives à une notification reçue par le téléphone. Nous avons constaté quelques limites pour certaines applications. En effet pour un groupe d'applications (notamment pour l'application *Messenger*) on intercepte plusieurs fois la même notification. Ce problème qui peut sembler très contraignant aux premiers abords, est en réalité résolu par la carte électronique, qui lorsqu'elle reçoit une information du téléphone s'occupe également de gérer les possibles doublons.

4.6 Gestion de l'heure

4.6.1 Contexte

Lorsque l'idée de réaliser pas seulement un bracelet mais une *montre* connectée fut entérinée, il devint évidemment nécessaire pour ledit accessoire d'avoir accès à l'heure. L'écoulement du temps est directement accessible depuis la carte en utilisant son horloge, mais celle-ci permet uniquement de mesurer une durée. Néanmoins, il est possible de connaître l'heure actuelle en ayant accès au temps écoulé depuis une heure de référence. L'heure de référence a été définie pour ce projet comme l'heure du téléphone, transmise à la carte à chaque communication vers celle-ci.

4.6.2 Le format "143"

Notons que puisque nous utilisons un anneau circulaire à 12 LED (et qu'une seule LED est utilisée pour les minutes), l'heure ne peut être donnée qu'à 5 minutes près. Rappelons qu'une telle horloge utilise un système horaire sur 12 heures. Avec ces contraintes, il devient possible de représenter l'heure au format "143", c'est-à-dire sous la forme d'un entier donné selon la logique suivante :

Heure standard	Format "143"
00h00	0
00h01	0
00h02	0
00h03	0
00h04	0
00h05	1
00h06	1
...	...
11h58	143
11h59	143
12h00	0
...	...
23h59	143

TABLE 1 – Table de conversion vers le format "143"

Cette conversion correspond à la formule : $heure_{143} = (heure \% 12) \times 12 + minute \div 5$. Par exemple, à 18h02, $heure = 18$ et $minute = 2$ d'où $heure_{143} = 72$.

4.6.3 En pratique

Android permet à toute application de connaître l'heure du téléphone par l'intermédiaire d'une classe dédiée. S'il en existe plusieurs, `GregorianCalendar` est la classe officiellement recommandée. Il a été décidé pour ce projet de créer une classe `OrionTime` avec les arguments suivants :

```

1  private GregorianCalendar mCalendrier;
2  private int mHeure;
3  private int mMinute;
```

```
4 private String mConversion;
```

mConversion représente l'heure au format "143", obtenu avec la fonction suivante :

```
1 public String conversion(int heures, int minutes) {
2     String temp = Integer.toString((heures % 12) * 12 + minutes / 5);
3     switch (temp.length()) {
4         case 1:
5             temp = "00" + temp;
6             break;
7         case 2:
8             temp = "0" + temp;
9             break;
10    }
11    return temp;
12 }
```

Remarquons qu'il est plus pratique d'envoyer à la carte une "trame" dont la longueur est toujours la même. Des "0" sont de ce fait ajoutés, si nécessaire, de manière à ce que la chaîne de caractère envoyée fasse toujours trois caractères de long. À 15h35, on enverra donc "042" au lieu de seulement "42".

Pour récupérer les informations désirées, il suffit d'utiliser la méthode `get()` de la classe `GregorianCalendar` avec en argument la constante de sa sur-classe `Calendar` qui correspond à l'information désirée :

```
1 public void miseAJour() {
2     setCalendrier(new GregorianCalendar());
3     setHeure(getCalendrier().get(Calendar.HOUR));
4     setMinute(getCalendrier().get(Calendar.MINUTE));
5     setConversion(conversion(getHeure(), getMinute()));
6 }
```

Enfin, il suffit d'instancier un objet `OrionTime`, et de le mettre à jour à chaque envoi d'informations vers la carte. Il faut récupérer l'heure au format "143" (logiquement avec un `getter` `getConversion()`) et l'adoindre à la trame de commande de la carte (concrètement en la concaténant à la chaîne de caractère qui commande les vibrations des actionneurs).

4.6.4 Résultats

Avec ce travail réalisé, nous parvenons à communiquer à la carte électronique l'heure à afficher. Si nous avons effectivement fait un choix de praticité en proposant une heure précise à 5 minutes près et s'actualisant uniquement lors de la réception d'une notification, notre solution présente néanmoins une alternative viable et cohérente vis-à-vis de nos contraintes.

4.7 Explication du fonctionnement de l'application

Nous allons, dans cette ultime partie, réaliser un tutoriel d'utilisation de l'application en faisant le tour des fonctionnalités disponibles. Il se peut que des redites soient faites avec les précédentes explications. Cette partie s'adresse à un public qui souhaite comprendre comment utiliser notre application et non à un public qui souhaite savoir comment elle a été réalisée.

4.7.1 Structure de l'application

Notre application se compose de trois pages principales (nommée *Accueil*, *Notifications* et *Vibrations*) qui sont accessibles : soit grâce au menu en bas de l'écran, soit en "swipant" (mouvement du doigt sur l'écran) entre les différentes pages. On remarque également la présence d'une barre d'outil (aussi nommée *toolbar*) sur le haut de l'écran avec trois boutons.

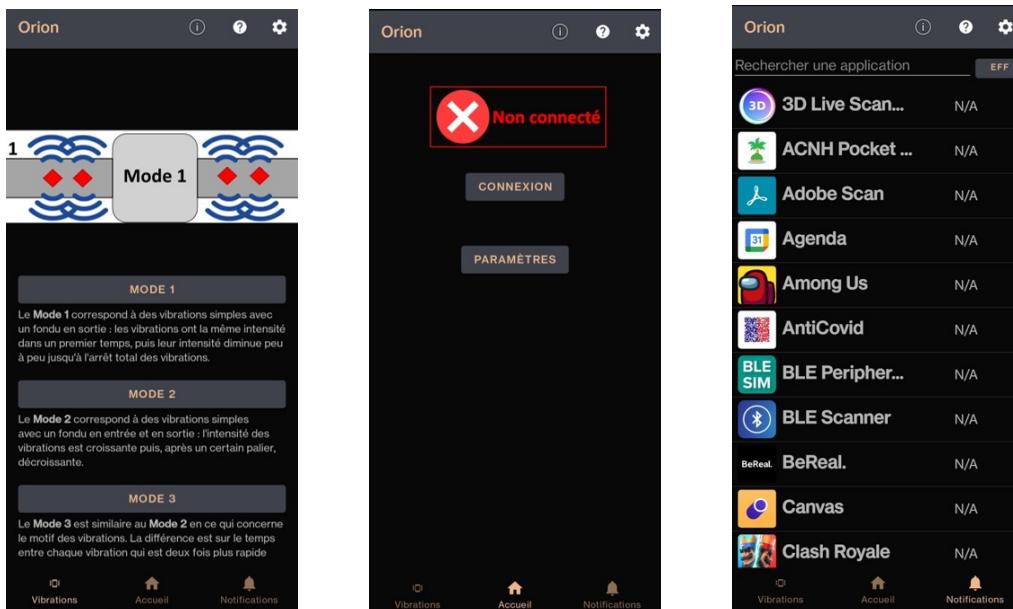


FIGURE 49 – Structure de l'application

4.7.2 La *toolbar*

Chaque bouton sur cette barre d'outil renvoie vers une autre page. Ainsi les deux premiers boutons (en partant de la gauche) permettent d'accéder aux pages *Aide* et *A propos*.

Sur ces pages, l'utilisateur va pouvoir retrouver :

- soit une aide sur le fonctionnement de l'application, avec notamment des explications sur les autorisations requises pour que l'application puisse tourner,
- soit des indications sur le contexte de l'application et sur notre projet de manière générale.

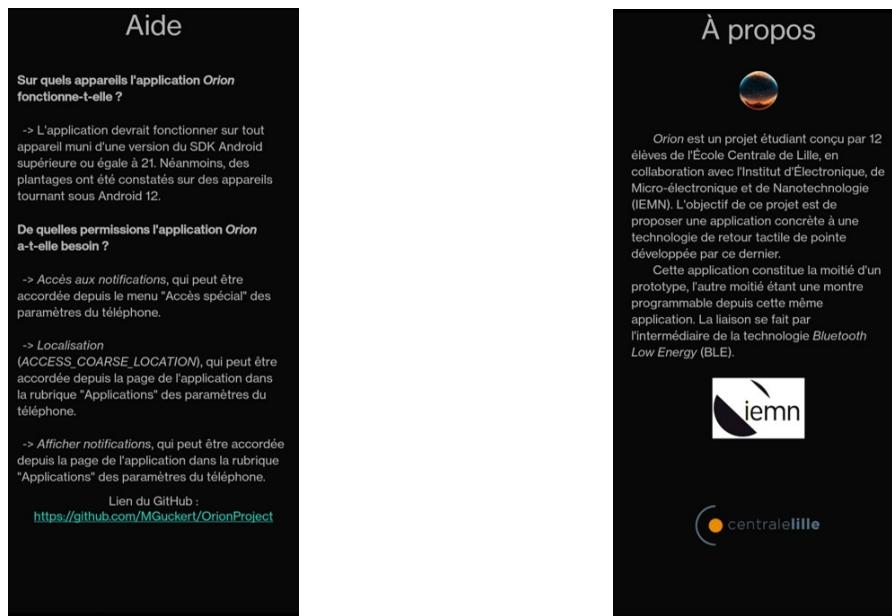


FIGURE 50 – Pages *Aide* et *A propos* de l'application

Enfin grâce au dernier bouton, nous pouvons accéder à la page *Paramètres* de l'application. Sur cette page nous allons retrouver plusieurs fonctionnalités comme la possibilité de changer l'adresse MAC de la carte à laquelle on souhaite se connecter, la possibilité de changer de mode d'affichage et enfin la possibilité de réinitialiser les modes de vibrations (voir la partie 4.7.5 pour plus de détail).

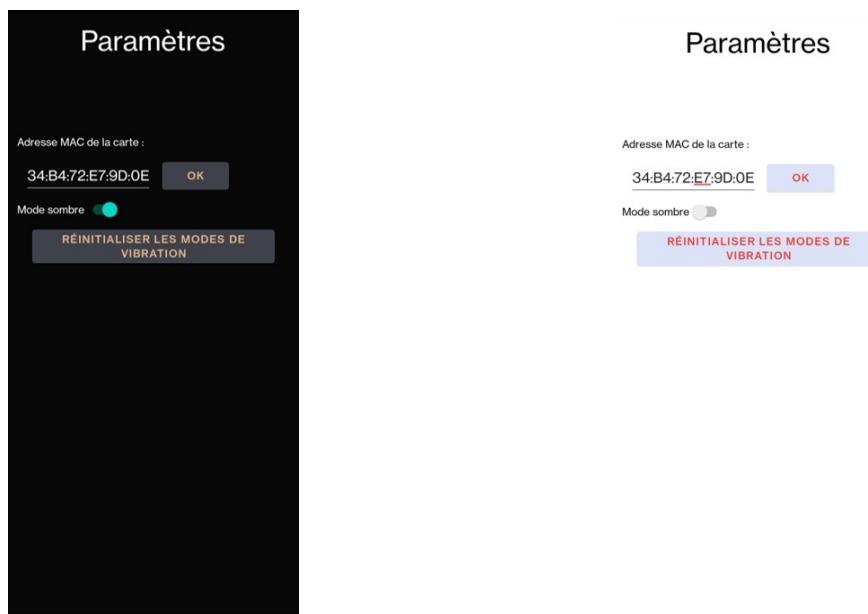


FIGURE 51 – Pages *Paramètres* de l'application en mode sombre et clair

4.7.3 La page Accueil

La page d'Accueil est une page de pilotage du module bluetooth. Sur cette dernière il est possible de se connecter et déconnecter de la montre mais aussi d'accéder au paramètres bluetooth du téléphone.

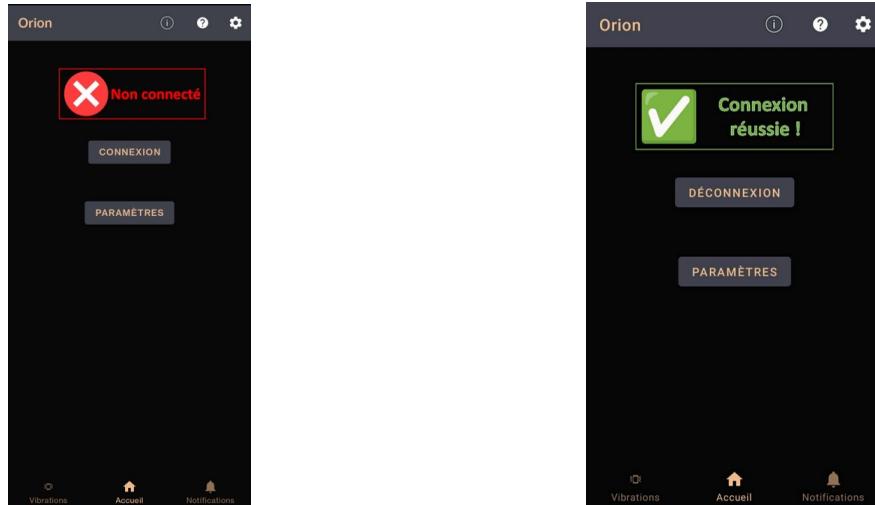


FIGURE 52 – Illustration de la page d'*Accueil* dans le cas déconnecté et connecté à la montre

4.7.4 La page *Vibrations*

Sur la page *vibrations* vous pourrez trouver un ensemble de descriptions sur les différents modes de vibrations disponibles. Chaque mode est illustré avec un gif qui est visualisable lors de l'appui sur le bouton associé.

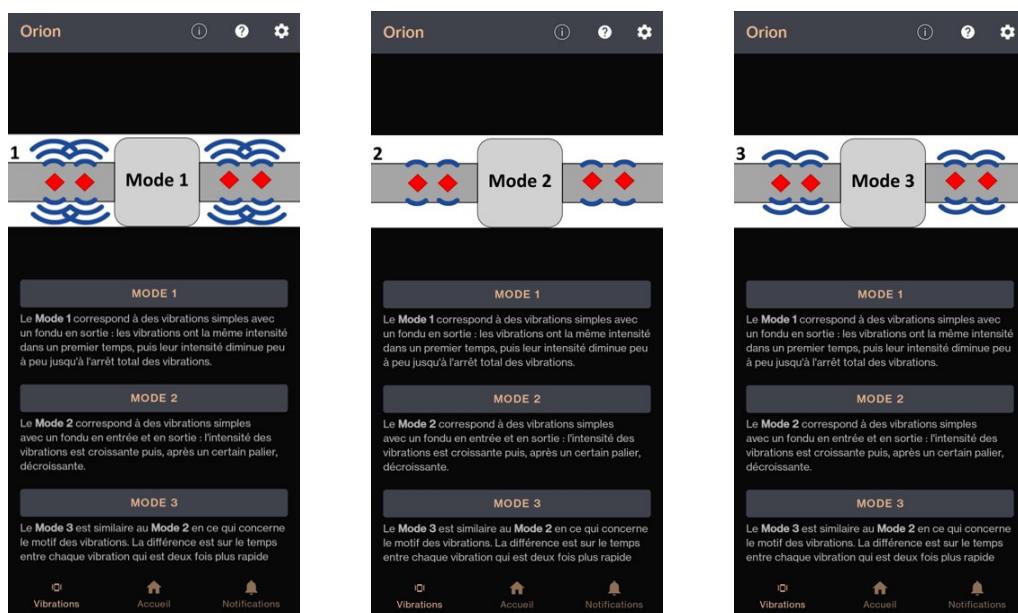


FIGURE 53 – Page *vibrations* de l'application

4.7.5 La page *Notifications*

Le but de cette dernière page est de pouvoir associer chaque application à un mode de vibration. Ainsi à chaque fois que l'utilisateur recevra une notification, sa montre vibrera selon le mode sélectionné. Notre logiciel récupère automatiquement la liste des applications installées et les affiche dans un menu déroulant. En sélectionnant une application l'utilisateur peut choisir son mode de vibration.

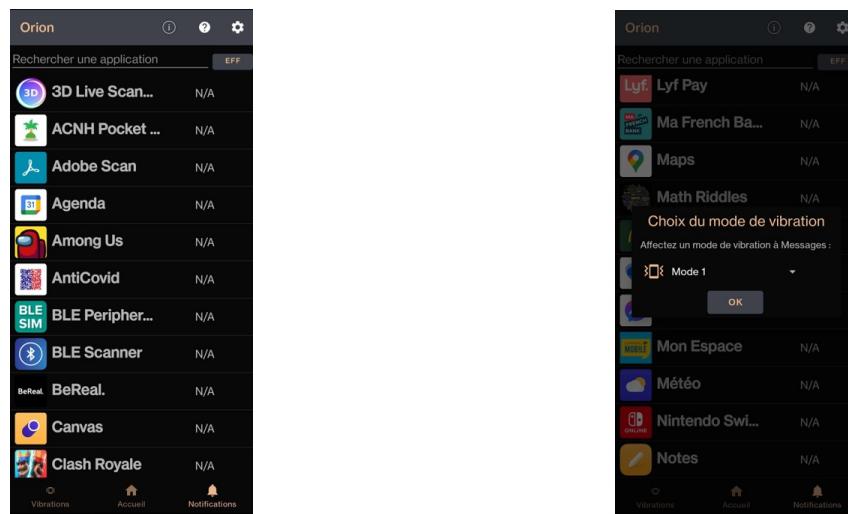


FIGURE 54 – Selection du mode de vibration

Des fonctionnalités supplémentaires ont été rajoutées pour le confort d'utilisation. Ainsi vous pourrez retrouver une barre de recherche pour trouver plus facilement une application.

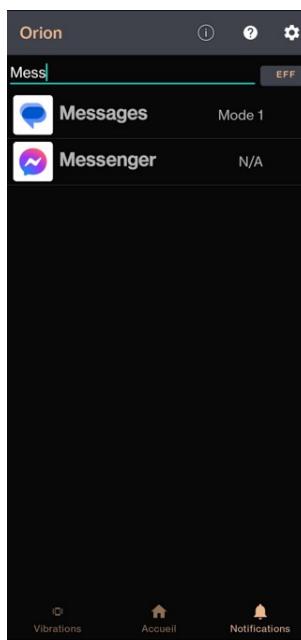


FIGURE 55 – Utilisation de la barre de recherche

4.8 Conclusion

Cette partie présente l'ensemble de notre travail de développement Android dans le cadre du projet Orion. A partir de ce dernier, n'importe quel développeur (ayant un minimum de base en conception d'application Android) devrait être capable de comprendre et reproduire l'ensemble de nos travaux.

Pour des informations complémentaires, vous pouvez accéder à notre dépôt GitHub en suivant ce lien.

5 Rapport scientifique : pôle électronique

5.1 Introduction

Depuis les années 2010 environ, nous pouvons assister à une révolution de l’IoT - Internet of Things, ou internet des objets en français. Cela s’observe avec le nombre croissant d’objets connectées, rendu possible par la création de protocoles d’échange de données, de puces électroniques pouvant les réaliser et leur miniaturisation permettant l’intégration de ces composants dans des objets du quotidien.

Le marché de l’IoT connaît une très forte croissance, et les avancées technologiques majeures sont fréquentes. On peut par exemple remarquer l’émergence de montres connectées avec écran AMOLED pour moins de 60€, intégrant des capteurs de mesure de fréquence cardiaque et avec une autonomie de plusieurs jours, ce qui aurait été infaisable il y a 10 ans. Tout cela en proposant à l’utilisateur une application moderne et complète avec une analyse détaillée de ses données physiologiques, et des possibilités de personnalisation.



FIGURE 56 – Exemple de montres connectées à moins de 60€, de gauche à droite : Mi Smart Band 6, Amazfit GTS 2, Blitzwolf BW-AT2C

Lors de sa réalisation, le projet *Tact Icônes 3*, réalisé entre 2013 et 2015, ne disposait pas des mêmes outils dont nous disposons aujourd’hui. C’est pourquoi l’objectif du pôle électronique était double : réduire l’encombrement de l’électronique dans le bracelet, et ajouter plus de fonctionnalités pour rendre le produit plus attractif et dans l’ère du temps. De plus, nous souhaitions également réduire les coûts de fabrication, ainsi que la consommation en utilisant le Bluetooth Low Energy (BLE).

Pour la réalisation de la carte électronique, nous travaillerons principalement sur KiCad, qui est un logiciel de référence pour la conception et la création de cartes électroniques. Pour la simulation des circuits, nous utiliserons LTSpice pour l’électronique de puissance, et Tinkercad pour tester certaines parties de l’application de la carte gérant la connexion BLE. Enfin, la dite application sera développée sur l’IDE Arduino, qui permet de facilement développer, compiler et tester du code en C++ avec notre carte.

Nous organiserons cette partie en sept parties principales :

- | | |
|--|--|
| 1. Le circuit électrique
2. Choix des composants
3. Réalisation du circuit
4. Validation du circuit | 5. La carte ESP32
6. L’anneau LED NeoPixelRing
7. L’autonomie de la montre |
|--|--|

5.2 Le circuit électrique

Après quelques tests rapides sur les anciens projets et surtout avec les actionneurs, nous avons remarqué que pour obtenir une sensation optimale au niveau du poignet, il était essentiel de réaliser un circuit électronique facilitant la commande. En effet, l'utilisation du signal envoyé par la carte ESP32, n'était pas suffisante pour avoir un ressenti optimal.

Le premier composant du système est le transistor car celui-ci nous permet de contrôler avec un courant faible un courant plus important dans une autre partie du système.

De plus nous avons modélisé dans un premier temps les actionneurs par une bobine résistance dans nos premières simulations. Ces actionneurs sont des bobines-aimants, donc nous souhaitons faire bouger l'aimant qui fera lui-même bouger la membrane, faire passer dans la bobine un courant de type sinusoïdal ou de façon optimale un signal carré. C'est pour cette raison que nous sommes partis sur un transistor MOSFET qui fonctionne comme un commutateur de courant. C'est à dire qu'il laisse passer plus ou moins de courant entre deux électrodes en fonction d'une troisième. Il peut aussi être vu comme un interrupteur.

Ainsi nous avons eu le schéma du circuit suivant (ici, seul un actionneur est simulé) :

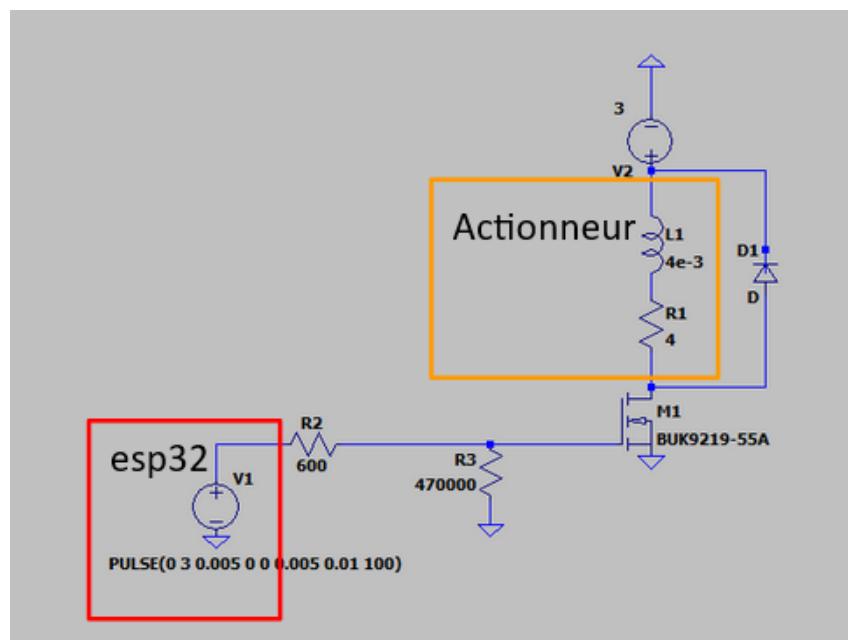


FIGURE 57 – Schéma du circuit réalisé sur LTspice

5.2.1 Les composants du circuit

Les actionneurs : Représenté par une inductance $L1 = 4mH$ et une résistance interne $R1 = 4W$.

L'alimentation : Nous avons souhaité partir sur une miniaturisation de la montre. Ainsi nous avons décidé de partir sur une pile bouton. Cette pile bouton fournira une tension de 3V aux actionneurs permettant ainsi un ressenti optimal. Il permet aussi d'alimenter l'esp32, qui génère les signaux, qui lui fonctionne sous 3V. Une autre version contiendra une pile classique.

Le transistor : Comme nous l'avons vu ci-dessus, nous utiliserons un transistor MOSFET qui fonctionnera en mode commutation. Ainsi pour chaque modèle de transistor MOSFET, on définit deux tension : $V_{GSThresholdmax}$ la tension de saturation et $V_{GSThresholdmin}$ la tension "débogage".

Mode bloqué : Tant que la tension VGS (entre la grille et la source) est inférieur à la tension de débogage, le transistor va être bloqué et donc on aura un courant non nul dans notre actionneur.

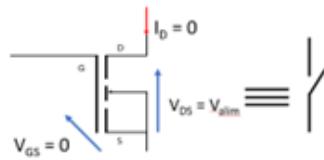


FIGURE 58 – Schéma du transistor mode bloqué

Mode saturé : Lorsque la tension VGS va passer au-dessus de la tension de saturation, le transistor va être passant et donc le courant sera nul dans l'actionneur.

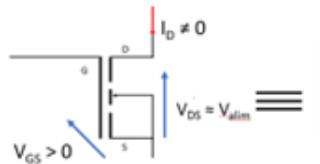


FIGURE 59 – Schéma du transistor mode saturé

Diode : La diode a pour rôle de protéger le transistor. Ainsi lorsque le transistor est en mode saturé, elle permet et force le courant à passer dans les actionneurs. Et lorsque le signal est bloqué, elle permet au courant de circuler dans la bobine et donc de se décharger et de faire revenir le courant à 0.

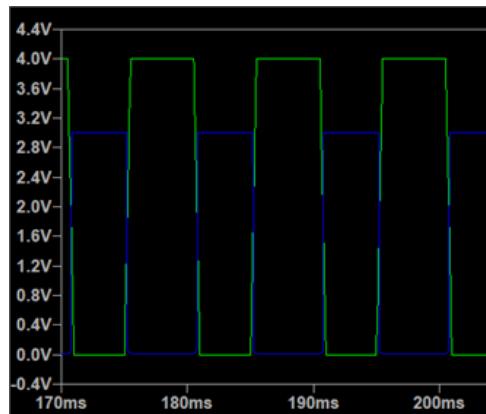
La résistance R3 : Il s'agit d'une résistance de protection. En effet elle permet, lorsque l'esp32 n'a pas de potentiel fixe, de relier le grill du transistor à la masse. Nous fixons donc une résistance élevée, par exemple 47 kΩ.

La résistance R2 : C'est une résistance permettant d'empêcher la surintensité qui pourrait venir de l'esp32, notamment lors de l'allumage.

5.2.2 Simulations et premiers tests

Nous avons réalisé dans un premier temps des simulations sur LTspice permettant alors de comprendre d'une part le fonctionnement des différentes parties du circuit expliqué ci-dessus. Mais aussi être sûr que nous n'allons pas faire brûler une partie ou un composant lors des tests réels.

Ainsi on a eu les résultats de simulation suivants pour les signaux en “entrée” et en “sortie” du transistor :



— : Signal de l'esp32
 — : Signal passant dans l'actionneur

FIGURE 60 – Résultats de la simulation sur LTspice

Une fois le montage validé sur LTspice nous sommes passé à des tests réelles sur une *breadboard*. Nous n'avions à ce moment là pas encore sélectionné réellement notre transistor. Donc nous avons pris un transistor MOSFET qui était disponible à Centrale.

De plus, pour ne pas détériorer les actionneurs que Monsieur TALBI nous avait fournis, nous avons décidé de tester dans un premier temps le circuit avec une résistance. Par la suite nous avons testé la sensation avec les actionneurs et le circuit total pour valider le circuit.

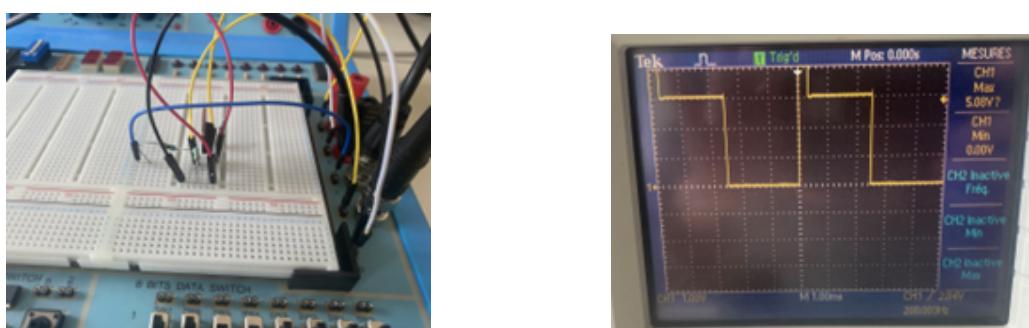


FIGURE 61 – Tests réels et résultats à l'oscilloscope

5.3 Choix des composants

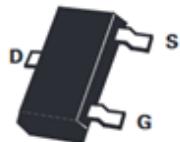
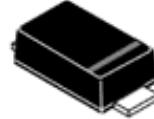
Une fois tous les tests réalisés et validés, nous avons dû commencer à réfléchir aux composants que nous souhaitions mettre en pratique sur la carte que nous allions réaliser.

Sous les conseils de Monsieur THOMY, nous nous sommes orientés vers des composants CMS qui ont l'avantage de ne pas posséder de broche. Ainsi ils sont directement soudés à la surface de la carte électronique, et donc aucun perçage de la carte n'est nécessaire. Ils sont aussi plus fiables car il y a moins de trous de connexion.

Les composants CMS sont aussi connus pour être plus petits que ceux avec broche. Or une de nos contraintes était de miniaturiser la montre et donc d'avoir la carte la plus petite possible ce qui était donc possible avec les composants CMS.

Pour l'alimentation, nous avons décidé de nous orienter vers une version intégrant une pile bouton car celle-ci nous permettrait d'assurer la miniaturisation de notre boîtier qui doit contenir toutes les fonctions électroniques de notre montre.

Ainsi nos composants ont été choisi en fonction des caractéristiques présenté précédemment :

Composants	Caractéristique(s)	Produit choisi
Transistor	$V_{GS(th_{max})} < 2,5 \text{ V}$	ZVN3306F 
Diode	$I = 2A \text{ V}$	ZVN3306F MBR230LSFT1G 
Régulateur	Tension entrée min : 2.7V Tension entrée max : 16V Tension de sortie fixe : 3V	MCP1703A-3002E/DB 
Batterie	Tension nominale : 3.7V	Eunicell-502535 

5.4 Réalisation du circuit

Une fois que nous avons bien compris le fonctionnement du circuit, nous sommes passés à la conception des circuits imprimés. Dans une optique de miniaturisation et d'expérimentation, nous avons réalisé deux versions de circuits. Une version disposant d'un régulateur et une autre sans régulateur.

→ Choix du logiciel :

Pour réaliser notre circuit, notre choix s'est porté sur le logiciel de conception de carte électronique Kicad. Étant largement répandu, la maîtrise des outils que le logiciel n'a pas posé de problème particulier.

→ Réalisation du schéma électrique :

La première étape consiste à réaliser le schéma électrique sur le logiciel.

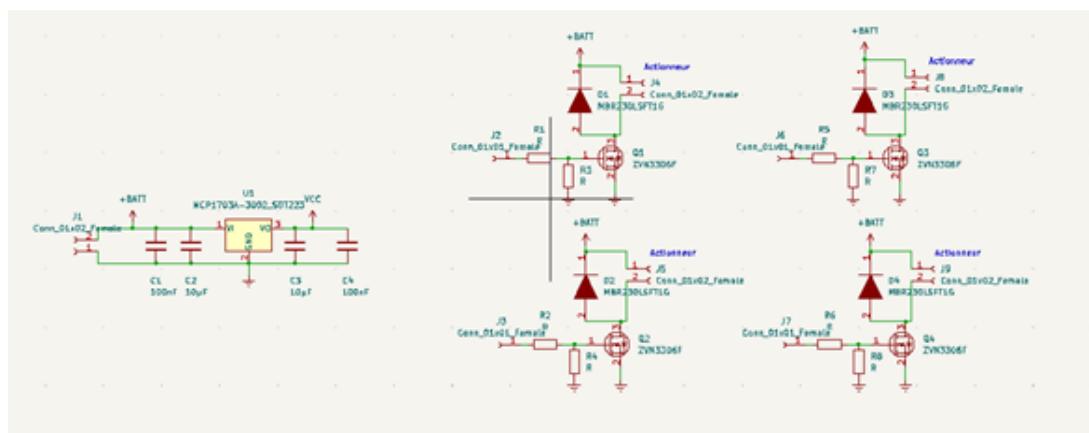


FIGURE 62 – Image du circuit 1 sur Kicad

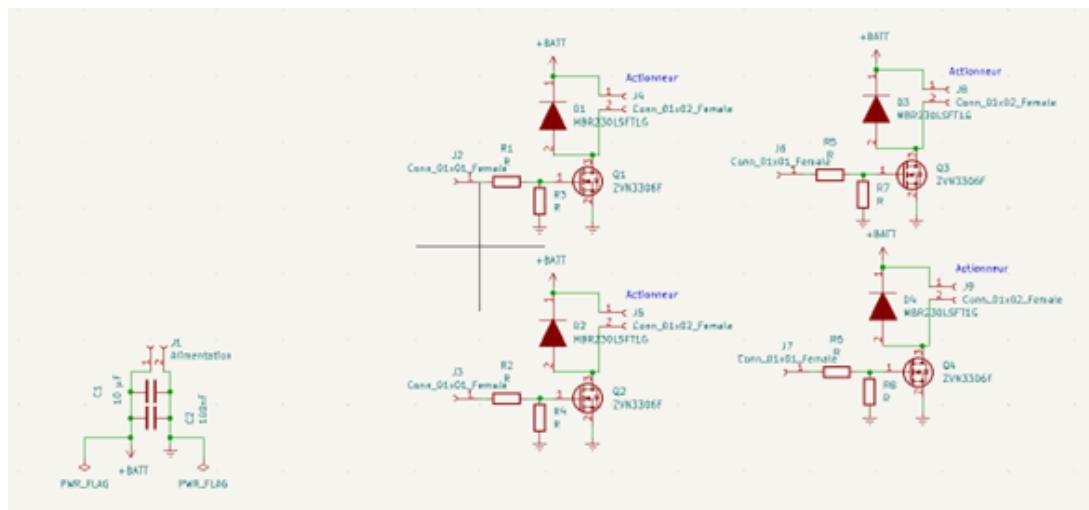


FIGURE 63 – Image du circuit 2 sur Kicad

Une fois que cette étape est validée et qu'aucune erreur n'est détectée, il est possible de passer au routage de la carte. Cette phase est importante car un bon agencement des composants permet d'optimiser la taille de la carte électronique.

→ **Routage du circuit :**

Le routage est conditionné par deux facteurs :

1. Le circuit électrique.
2. La technologie d'impression.

Dans la première version, les capacités C1, C2, C3, C4 sont des condensateurs de découplage. Ils permettent d'augmenter l'immunité électromagnétique et doivent être placés près des pattes d'alimentation sur le circuit, c'est-à-dire qu'il en faut une près du fil relié aux piles, une près du régulateur, et une près des fils de chaque actionneur. Vous trouverez en annexe un passage expliquant l'intérêt des condensateurs de découplage.

La deuxième version dispose aussi de condensateurs de découplage qui sont disposés près des pattes d'alimentation.

Au-delà des contraintes de disposition liées au circuit électrique, nous étions limités par la technologie d'impression de l'école. En effet, la largeur minimale des pistes ne pouvait être inférieure à 0,4 mm.

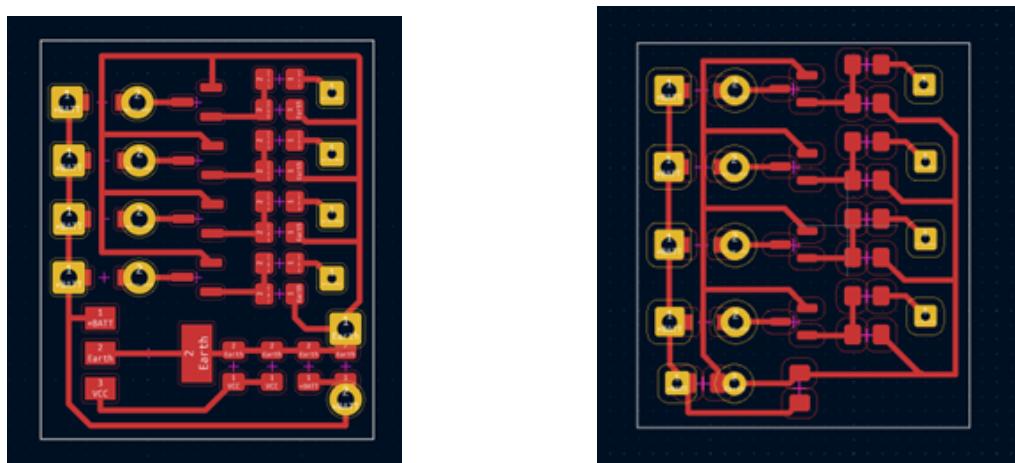


FIGURE 64 – Routage de la première version (à gauche), et de la seconde version (à droite)

Après avoir validé cette étape, nous sommes passés à l'impression de la carte, le perçage des trous et le soudage des composants. Ces derniers ont été réalisés à l'atelier d'électronique de l'école avec l'aide des techniciens de l'atelier.

Le résultat est le suivant :



FIGURE 65 – Cartes imprimées à la sortie du four

Les dimensions finales des cartes sont les suivantes :

1. 24x27 mm pour la première version.
2. 23x27 mm pour la seconde version.

5.5 Validation du circuit

Il est primordial de passer par cette étape de validation afin de cerner tous les problèmes liés à la conception de notre dispositif.

Ne disposant pas de l'alimentation pendant cette phase, afin de tester le circuit, nous avons utilisé un générateur du laboratoire d'électronique.

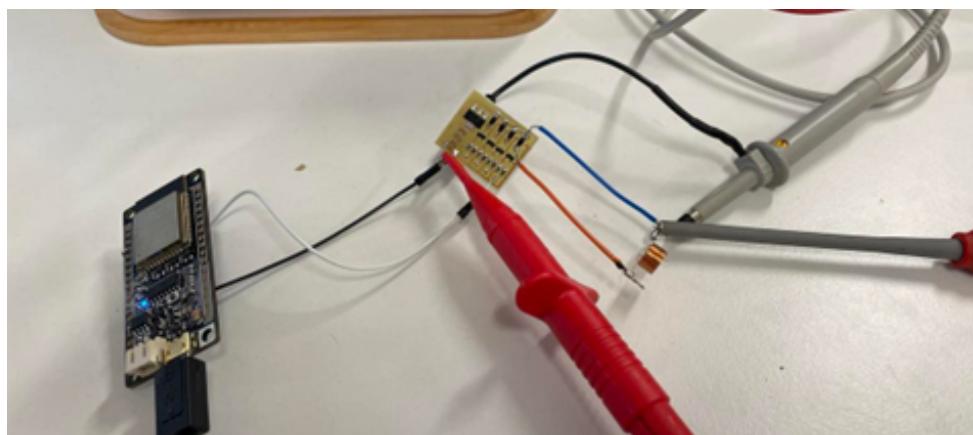


FIGURE 66 – Image de la première phase de test

Dans ce cas ci nous avons soudé une bobine d'actionneur sur le circuit, puis nous avons connecté la carte électronique à l'esp32. Ce dernier permet de générer des signaux créneaux que l'on peut paramétriser. Pour vérifier le bon fonctionnement du circuit nous avons observé le signal électrique sur différents points du circuit avec l'oscilloscope.

Les signaux générés par l'esp32 étaient commandés via bluetooth à l'aide de l'application nRF Connect.

Après avoir validé le bon fonctionnement du circuit, nous sommes passés au test de la chaîne d'information complète (Application → esp32 → Circuit électrique → Vibrations).

5.6 La carte ESP32

5.6.1 Présentation

L'ESP32 est un SoC (socket on chip), ou plutôt une série de micro-contrôleurs développés par Espressif Systems et décliné en différentes versions (ESP32 C3, ESP32 S2, ESP32 S3), intégrant des fonctionnalités de WiFi et de Bluetooth (notamment le BLE 5.0).

Originellement basé sur une architecture Xtensa, ce SoC consomme très peu d'énergie et possède de nombreuses fonctionnalités le rendant particulièrement intéressant pour l'IOT, notamment car il est assez peu cher et facile à programmer, soit avec l'ESP-IDF fourni par Espressif System, soit directement avec l'IDE de Arduino.

Un diagramme fonctionnel de l'ESP32 est disponible ci-dessous. Ce micro-contrôleur supporte de nombreux protocoles, notamment l'UART pour sa programmation via USB, et intègre le PWM, qui est utile pour le pilotage d'actionneurs.

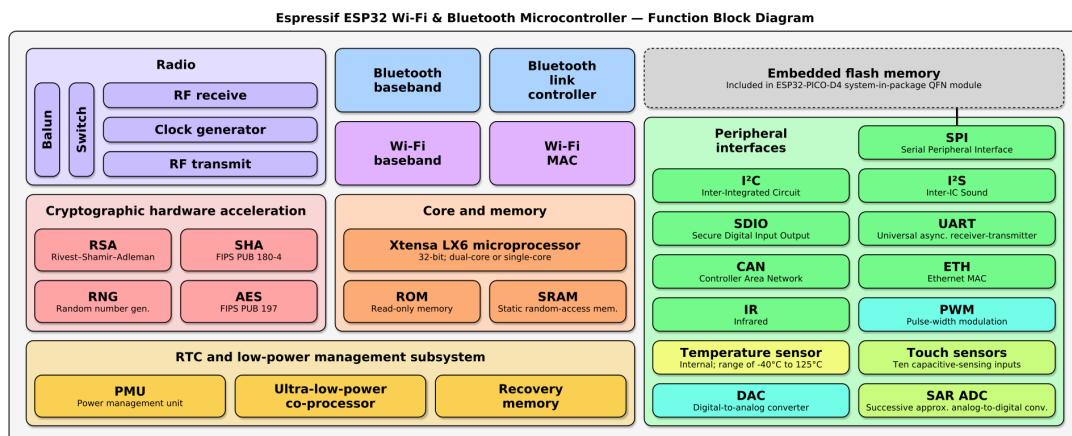


FIGURE 67 – Schéma bloc fonctionnel de l'ESP32

Ce composant doit être alimenté en 3,3V, même s'il peut fonctionner entre 2,3V et 3,6V avec au moins 500mA de courant. Ensuite, il possède 16 GPIO supportant le PWM, avec une tension allant jusqu'à 3,3V.

Des informations complémentaires peuvent être retrouvées dans le datasheet de l'ESP32.

5.6.2 ESP32-C3-DevKitM-1

Afin de pouvoir réaliser un premier prototype électronique, il était nécessaire de trouver une carte de développement adaptée à nos besoins, c'est-à-dire un ESP32 facilement programmable en USB, intégrable à une *breadboard*, et disposant du nécessaire à son fonctionnement (antenne, régulateur de tension...). C'est avec ces critères en tête que nous nous sommes tournés vers l'ESP32-C3-DevKitM-1 distribué par Espressif.

En effet, ce kit de développement est assez peu cher, récent et offre toutes les fonctionnalités souhaitées. Il est basé sur l'ESP32-C3, une version plus récente de l'ESP32 fonctionnant sur une architecture RISC V, qui est donc plus puissant sans consommer plus, et parfait pour une utilisation avec le BLE.

Le composant est protégé par un blindage pour prévenir tout problème électrostatique, comme détaillé dans la figure ci-dessous.

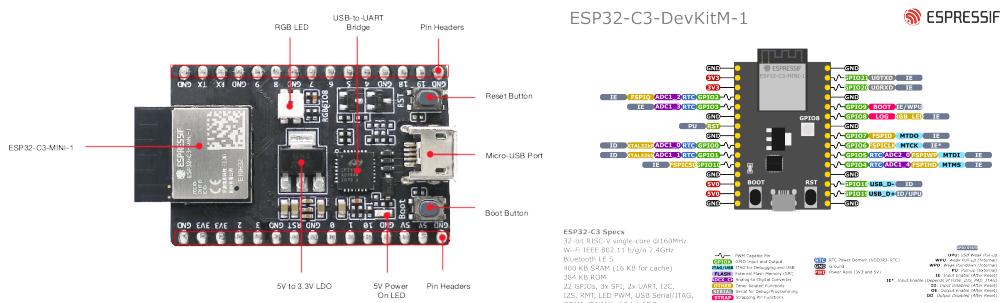


FIGURE 68 – ESP32-C3-DevKitM-1 - image et broches

Le devkit possède notamment un bouton pour redémarrer l'ESP32, un port micro USB pour la programmation avec MicroPython, Arduino ou l'ESP-IDF et une antenne 2.4GHz pour le Wifi et le bluetooth.

5.6.3 Adafruit QT Py ESP32-C3

La carte précédente nous a permis de réaliser un premier prototype dit « monstre » afin de vérifier la compatibilité et le bon fonctionnement des composants. Néanmoins, afin d'obtenir un prototype satisfaisant, il était nécessaire de chercher une carte plus compacte, conservant les mêmes fonctionnalités.

Après des recherches, la carte QT Py ESP32-C3 nous semblait être la plus appropriée car :

1. Ses dimensions sont particulièrement compactes (cf figure 69)
2. Elle possède suffisamment de sorties pour contrôler les 4 actionneurs et l'anneau LED
3. Elle est basée sur la même puce que la carte précédente
4. Elle se programme de la même manière mais en USB type C
5. Elle possède une antenne intégrée très petite
6. Elle s'alimente en 3.3V et possède un régulateur de tension

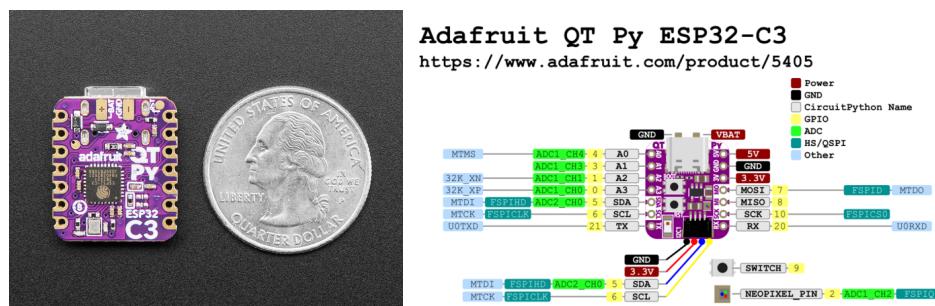


FIGURE 69 – Adafruit QT Py ESP32-C3 - image et broches

5.6.4 Programmation - BLE

→ Présentation du BLE dans la carte ESP32 :

Le Bluetooth Low Energy, ou BLE en abrégé, est une variante économique en énergie du Bluetooth. L'application principale du BLE est la transmission de petites quantités de données sur de courtes distances (faible bande passante). Contrairement au Bluetooth, qui est constamment activé, le BLE est toujours en mode veille, sauf si une connexion est établie.

Ce mode de fonctionnement lui fait consommer très peu d'énergie. Le BLE consomme environ 100 fois moins d'énergie que le Bluetooth (selon le cas d'utilisation).

Pour le Bluetooth Low Energy, il existe deux types d'appareils : les serveurs et les clients. L'ESP32 peut agir à la fois comme client et serveur.

Le serveur annonce son existence afin que d'autres appareils puissent le trouver, et contient des données que les clients peuvent lire. Le client recherche les appareils à proximité et lorsqu'il trouve le serveur qu'il recherche, il établit une connexion et écoute les données entrantes. C'est ce qu'on appelle la communication « peer-to-peer ».

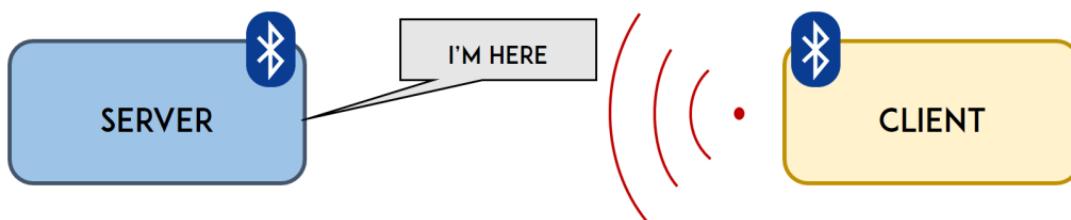


FIGURE 70 – Communication client - serveur en BLE

→ Principe de fonctionnement :

GATT signifie Generic ATTRIBUTES et définit une structure de données en couches qui est exposée aux appareils BLE connectés. Cela signifie que le GATT définit la manière dont deux appareils BLE peuvent envoyer et recevoir des messages standards. La connaissance de ce protocole est importante car cela permet de comprendre plus facilement comment utiliser le BLE et comment écrire l'application. Le diagramme 71 résume la structure du GATT.

Le niveau supérieur de la hiérarchie est un profil composé d'un ou plusieurs services. Habituellement, un appareil BLE contient plus d'un service.

Chaque service contient au moins une caractéristique, ou peut également référencer d'autres services. Un service est simplement une collection d'informations, comme des valeurs de capteurs, par exemple.

Il existe des services standards pour plusieurs types de données classiques, définis par le SIG (Groupe d'Intérêt Spécial Bluetooth) : Niveau de la batterie, Tension artérielle, Fréquence cardiaque, etc

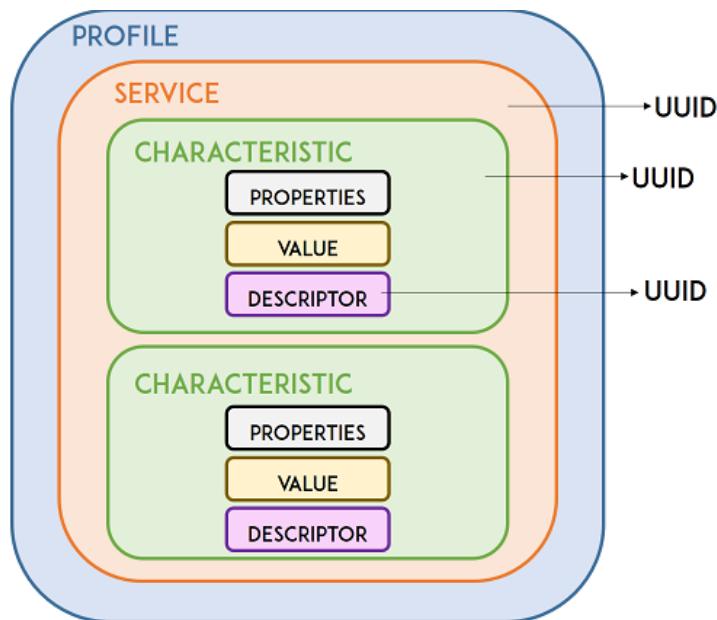


FIGURE 71 – Gatt en BLE

→ **Caractéristique BLE :**

La caractéristique appartient toujours à un service et c'est là que les données réelles sont contenues dans la hiérarchie.

La caractéristique a toujours deux attributs : la déclaration de la caractéristique et la valeur de la caractéristique. De plus, la valeur de la caractéristique peut être suivie de descripteurs, qui développent davantage les méta-données contenues dans la déclaration de ladite caractéristique.

Les propriétés décrivent comment interagir avec la valeur de la caractéristique. En résumé, il contient les opérations et procédures utilisables avec la caractéristique :

- Diffuser
- Lire
- Ecrire sans réponse
- Écrire
- Notifier
- Indiquer
- Écritures signées authentifiées
- Propriétés étendues

→ **Programmation du BLE :**

Pour notre projet on crée un appareil BLE appelé « Orion » :

```
1 BLEDevice::init("Orion");
```

Ensuite on définit le périphérique BLE en tant que serveur :

```
1 BLEServer * MyServer = BLEDevice::createServer();
2 MyServer -> setCallbacks(new MyServerCallbacks());
```

Après cela, on crée un service pour le serveur BLE :

```
1 BLEService * MyService = MyServer -> createService(SERVICE_UUID);
```

Ensute, on définit la caractéristique de ce service, que l'on met en écriture :

```
1 MyService -> addCharacteristic( & Mycarac );
2 MyService -> addCharacteristic( & pCaracteristique );
```

Il y a aussi une classe `MyServerCallbacks` qui définit les fonctions de rappel pour les événements de connexion et de déconnexion pour le serveur BLE, et une classe `MyCallbacks` qui définit la fonction de rappel pour l'écriture sur la caractéristique BLE.

Des instructions `Serial.println()` sont utilisées pour imprimer des messages de débogage dans le moniteur série, très utiles lors de la phase de développement et de test.

5.6.5 Programmation - Actionneurs

La deuxième caractéristique est utilisée pour recevoir des données écrites. Il y a trois modes d'exécution, `mode_1()`, `mode_2()` et `mode_3()`.

```
1 void mode_1() {
2     for (int dutyCycle = 0; dutyCycle <= 1000; dutyCycle++) {
3         // changing the LED brightness with PWM
4         ledcWrite(channel, 500);
5         delay(30);
6     }
7     // decrease the LED brightness
8     for (int dutyCycle = 200; dutyCycle >= 0; dutyCycle--) {
9         // changing the LED brightness with PWM
10        ledcWrite(channel, dutyCycle);
11        delay(15);
12    }
13 }
```

On importe également la bibliothèque `LEDC`, généralement utilisée pour contrôler la luminosité d'une LED connectée à une broche spécifique en utilisant la modulation PWM, mais dans notre cas utilisées pour les actionneurs. Le code définit des propriétés pour la fréquence PWM, le canal de LED et la résolution.

Dans ce code, la fonction `ledcWrite()` est utilisée pour envoyer des signaux PWM à une broche spécifique pour contrôler la luminosité d'un LED connecté à cette broche. Cependant, cette même fonction peut également être utilisée pour contrôler d'autres types d'actionneurs tels que nos actionneurs en fonction de la broche à laquelle l'actionneur est connecté et des paramètres PWM utilisés.

Il y a trois modes d'exécution, `mode_1()`, `mode_2()` et `mode_3()`, qui sont des exemples de séquences d'intensité de vibration pour un actionneur de vibration. Les deux fonctions majeures utilisées sont :

- `ledcSetup(ledChannel, freq, resolution);`
- `ledcWrite(ledChannel, dutyCycle);`

5.7 L'anneau LED NeoPixel Ring - 12 x 5050 RGB LED

5.7.1 Présentation

Afin d'avoir en plus du retour tactile un retour visuel et des informations concernant la connexion/déconnexion bluetooth et l'heure, il a été décidé d'ajouter un anneau LED, plus précisément le NeoPixel Ring - 12 d'Adafruit, équipé donc de douze LED 5050 RGB.

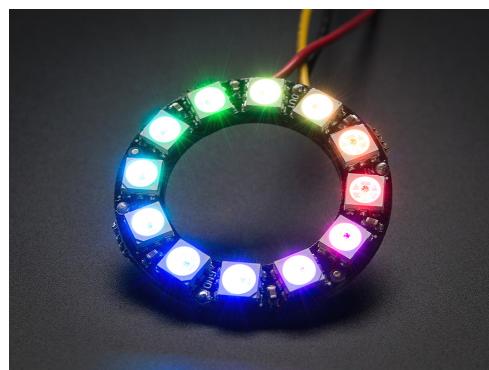


FIGURE 72 – Anneau LED NeoPixel Ring - 12 x 5050 RGB LED

Cet anneau de taille assez réduite (37mm de diamètre) est particulièrement simple d'utilisation, car il suffit de l'alimenter (5V - GND) et de lui fournir un signal adéquat pour déclencher des séquences lumineuses (DATA IN). Dans notre cas d'utilisation, la sortie de données (DATA OUT) ne nous sera pas utile.

La technologie de LED dite des NeoPixel d'Adafruit, disponible en plusieurs formes (anneaux, matrices, rubans...) nous semblait ici appropriée, car assez économique en énergie et également très documentée.

Des informations complémentaires peuvent être retrouvées dans le datasheet des neo-pixels d'Adafruit.

5.7.2 Compatibilité avec l'ESP32

Pour ce qui est de la compatibilité avec l'ESP32, il semble *a priori* y avoir un problème. En effet, le NeoPixel Ring - 12 est fait pour fonctionner à une tension de 5V, c'est-à-dire alimenté en 5V et recevant des instructions entre 0V et 5V donc.

Or l'ESP32 est alimenté en 3.3V et fournit des instructions entre 0V et 3.3V. Comme nous l'avons constaté plus haut, les modèles d'ESP32 choisis fournissent bien un rail de tension de 5V pour alimenter l'anneau, mais les instructions envoyées restent entre 0V et 3.3V.

Néanmoins, en alimentant l'anneau LED en 3.3V, et en prenant la masse sur l'ESP32 et le signal sur le PIN 6, l'anneau LED fonctionne parfaitement, avec toutes les leds fonctionnelles.

5.7.3 Programmation

Dans un premier temps, afin de facilement tester les séquences lumineuses, nous nous sommes basés sur des simulations sur Tinkercad, une plateforme en ligne d'Autodesk permettant de faire des schémas électroniques, et intégrant la possibilité d'intégrer et de programmer un Arduino.

Ainsi, nous avons réalisé un branchement basique de l'anneau LED sur un Arduino, et ensuite pu tester les différentes instructions envoyées.

La programmation de l'ESP32 se faisant également dans le langage Arduino, nous n'avons eu aucun problème à transférer les fonctions développées sur Tinkercad telles quelles.

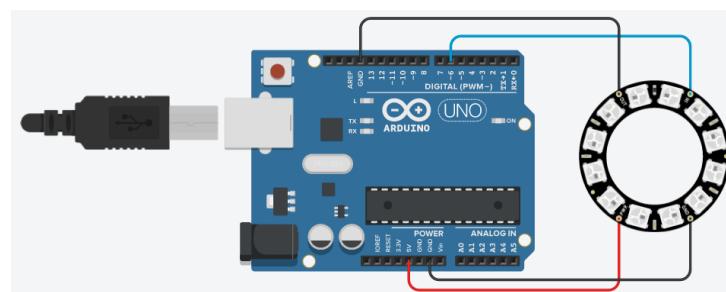


FIGURE 73 – Simulation sur Tinkercad de l'anneau LED

La programmation est facilitée par les nombreux exemples disponibles sur internet, et la bibliothèque Adafruit_NeoPixel fournie par Adafruit.

Quatre fonctions sont utilisées pour créer des affichages sur l'anneau LED. La première fonction consiste à afficher un cercle arc-en-ciel, montrant que la montre attend d'être appairée.

```

1 void rainbow(uint8_t wait) {
2     clearLed();
3     uint16_t i, j;
4     for (i = 0; i < strip.numPixels(); i++) {
5         strip.setPixelColor(i, Wheel((i * 256 / strip.numPixels()) & 255));
6     }
7     strip.show();
8     delay(wait);
9 }
```

La seconde permet de montrer un cercle bleu qui se remplit lors de la connexion en bluetooth.

```

1 void connectedBluetooth() {
2     clearLed();
3     for (uint16_t i = 0; i < strip.numPixels(); i++) {
4         strip.setPixelColor(i, bblue_color);
5         strip.show();
6         delay(100);
7         delay(1000);
8 }
```

La troisième permet de montrer un cercle rouge qui se remplit lors de la déconnexion du bluetooth.

```

1 void disconnectedBluetooth() {
2     clearLed();
3     for (uint16_t i = 0; i < strip.numPixels(); i++) {
4         strip.setPixelColor(i, bred_color);
5         strip.show();
6         delay(100);
7     }
8     delay(1000);
9 }
```

Enfin, lors de la connexion du téléphone, la dernière fonction permet d'afficher le temps (les heures en rouge et les minutes en blanc), à 5 minutes prêt donc puisque les douze LED de l'anneau ne nous permettent une précision bien plus importante.

```

1 void displayTime(uint8_t time) {
2     int minutes = time % 12;
3     int hours = time / 12;
4     for (int t = 0; t < 2; t++) {
5         for (uint16_t i = 0; i < strip.numPixels(); i++) {
6             if (i == hours and i == minutes) {
7                 strip.setPixelColor((i + 3) % 12, mh_color);
8             }
9             if (i == hours and i != minutes) {
10                 strip.setPixelColor((i + 3) % 12, h_color);
11             }
12             if (i == minutes and i != hours) {
13                 strip.setPixelColor((i + 3) % 12, m_color);
14             }
15             if (i != hours and i != minutes) {
16                 strip.setPixelColor((i + 3) % 12, no_color);
17             }
18             strip.show();
19             delay(5);
20         }
21     }
22 }
```

Ci dessous une figure illustrant l'affichage de l'heure sur notre prototype :



FIGURE 74 – Affichage de l'heure sur le prototype. Le point rouge indique les heures et le point blanc les minutes, la montre affiche donc 13h30 (où 01h30)

5.8 L'autonomie de la montre

En utilisant une batterie plate de 400 mAh :

- En utilisant les actionneurs seuls et en supposant que l'utilisateur reçoit 100 notifications par jour, et que la durée de la vibration est de 3s. L'autonomie de la batterie atteint 54 jours.
- L'anneau LED est très énergivore et fait chuter l'autonomie de la batterie. Celle-ci ne dépasse plus 24 h. En effet, le courant minimal est de 18 mA, ainsi, en négligeant les pertes et l'énergie consommée par les actionneurs. La batterie ne peut fonctionner que durant : $400 / 18 = 22.2$.

5.9 Pistes d'améliorations

Pour des raisons d'esthétique, d'ergonomie et de miniaturisation nous avons notamment pensé à utiliser un matériau plus flexible afin d'imprimer notre carte électronique et y placer nos composants. En effet, l'utilisation d'une structure rigide telle que la nôtre nous oblige à utiliser un boîtier assez imposant. La miniaturisation de chaque composant apparaît ainsi comme la piste d'amélioration principale pour notre pôle. Également, travailler vers une industrialisation des actionneurs pour faciliter leur implémentation, pourrait être une idée.

5.10 Conclusion

Nous avons résumé dans cette partie l'ensemble du travail effectué en électronique pour ce projet. L'alimentation des actionneurs ayant été détaillée, il ne devrait pas être difficile de concevoir d'autres systèmes utilisant plus ou moins d'actionneurs pour un lecteur possédant des bases en électronique. Il est également possible d'apporter des modifications au code de l'ESP32 pour rajouter des fonctionnalités, changer les modes de vibration et les animations de l'anneau LED.

Le détail du code peut être retrouvé sur Github en suivant ce lien.

6 Rapport scientifique : pôle conception

6.1 Introduction

Le dernier projet avec pour objectif la conception d'un bracelet connecté, le *Tact Icônes 3* a été réalisé entre 2013 et 2015, et depuis les standards et technologies dans le domaine des objets connectés ont beaucoup évolué. En effet, les montres connectées sont désormais monnaie courante et sont devenues très fines, confortables et légères. On retrouve à la fois des montres connectées possédant des écrans digitaux (comme l'*Apple Watch* par exemple) voire aussi des montres sur le segment du luxe telle que la *Montblanc Summit 3*) ou des montres à affichage hybride analogique/digital embarquant des technologies de suivi de l'activité cardiaque (comme la *Withings ScanWatch*).



FIGURE 75 – De gauche à droite : *Withings Scanwatch*, *Montblanc Summit 3*, Prototype *Tact'Icônes 3*.

Il s'agissait donc pour nous concevoir, dans le cadre de ce projet, une montre connectée confortable, avec un plus faible encombrement et un aspect esthétique plus discret sur le poignet, qui corresponde mieux aux attentes actuelles. Le pôle Conception a donc eu pour fil rouge ces objectifs, et c'est aussi pourquoi nous avons choisi -contrairement à l'ancien prototype- de concevoir et fabriquer nous même le bracelet en plus du boîtier, avec pour but de pouvoir y intégrer directement les actionneurs vibrant tout en pouvant agir sur la forme et le matériau du bracelet pour en maximiser le confort. Cela permettra donc aussi de diminuer significativement les dimensions du boîtier rigide qui n'aura plus à accueillir d'actionneur, alors même que le nombre total de vibrateurs intégrés a doublé : on passe de 2 actionneurs dans l'ancien projet à 4, ce qui augmente les possibilités de spatialisation des vibrations (on peut alors différencier plus de notifications). Ce nouveau projet a également pour objectif de pouvoir embarquer un nouvel élément : l'anneau LED intégré au boîtier qui contribuera non seulement à l'aspect esthétique de l'objet, mais apportera aussi de nouvelles fonctionnalités, intégré au boîtier il permettra d'afficher l'état de fonctionnement de la montre ainsi que l'heure.

Nous organiserons cette partie en six parties principales :

1. Conception et réalisation du boîtier
2. Conception et réalisation du bracelet
3. Eléments standards
4. Conception et réalisation de la membrane
5. Test du premier assemblage
6. Test du second assemblage

6.2 Conception et réalisation du boîtier

6.2.1 Contexte

Comme nous l'avons évoqué dans l'introduction, il s'agit au cours de la conception du boîtier d'optimiser l'espace, en travaillant en relation avec le pôle électronique pour faire en sorte que le système s'intègre correctement au *casing*. Un autre point d'attention est de veiller à ce que le câblage puisse être effectué correctement entre l'électronique et les actionneurs. Quant à l'ergonomie, nous avons cherché à obtenir un boîtier adapté à la courbe et à la taille du poignet, tout en gardant un design le plus sobre et agréable à l'œil possible. Enfin, un enjeu important réside dans la qualité esthétique et la netteté des surfaces du boîtier produit que nous avons tâché d'optimiser grâce à notre choix de matériau et de technique d'impression (en PLA puis en résine).

6.2.2 Première conception

La forme de ce boîtier est inspirée des montres connectées de type Apple Watch, tout en prenant en compte le besoin d'espace pour la carte électronique et en intégrant une partie voûtée au niveau du poignet pour soigner l'ergonomie. Les “cornes” (terme consacré en horlogerie) pour fixer le bracelet sont quant à elles semblables à celles qu'on peut trouver sur une montre d'horlogerie classique.



FIGURE 76 – Illustration d'un boîtier horloger classique avec des “cornes”

La première mission du pôle mécanique fût d'imaginer la forme du boîtier. Cette conception s'est tout d'abord appuyée sur les différentes observations et remarques concernant le prototype du projet précédent “Tact’Icônes 3” dont une photo est présente en introduction de cette partie.

Nos remarques préliminaires portant sur ce précédent projet sont les suivantes, ce sont des observations qui devront nous guider dans la conception de nos différentes versions :

- *Taille* : le boîtier a une hauteur bien trop conséquente qui, en dehors du côté esthétique, rend difficile son utilisation. On ne peut pas le faire passer sous un pull par exemple.
- *Ergonomie* : boîtier très volumineux donc peu pratique à garder sur le poignet, on remarque également la présence d'angles saillants qui peuvent être désagréables à porter ou dangereux dans certaines situations.
- *Esthétique* : imprimé en PLA puis peint en rouge, on remarque les couches d'impression sur les grandes surfaces.

Il en résulte ainsi un premier objectif : que le boîtier soit le plus petit possible et avec des angles adoucis. Il a donc fallu échanger avec le pôle électronique quant à la taille des composants qu'ils voulaient intégrer. Nous avons alors choisi de prendre de la marge pour la taille du premier prototype, les tailles intérieures choisies étant de 40x30mm. On déterminera de manière expérimental la courbure à donner au boîtier en utilisant les différents poignets de l'équipe projet.

De ce travail préliminaire a donc débuté la première conception CAO du boîtier sur Onshape. La première esquisse, très simple, dans le plan "Top" permet de définir les côtes de la base du boîtier. On observe sur la capture d'écran ci-dessous que l'esquisse est composée de deux rectangles de même centre mais de tailles différentes. Il est fortement conseillé pour la suite de faire coïncider les centres des rectangles et le centre du repère de Onshape. L'écart de 2mm entre leurs arêtes permet de définir l'épaisseur des parois latérales qui résulteront de l'extrusion de ce contour. De plus, à gauche et à droite, on remarque 4 rectangles de taille 5x7mm. Ces derniers permettront, après avoir été aussi extrudés, de créer les attaches pour le futur bracelet.

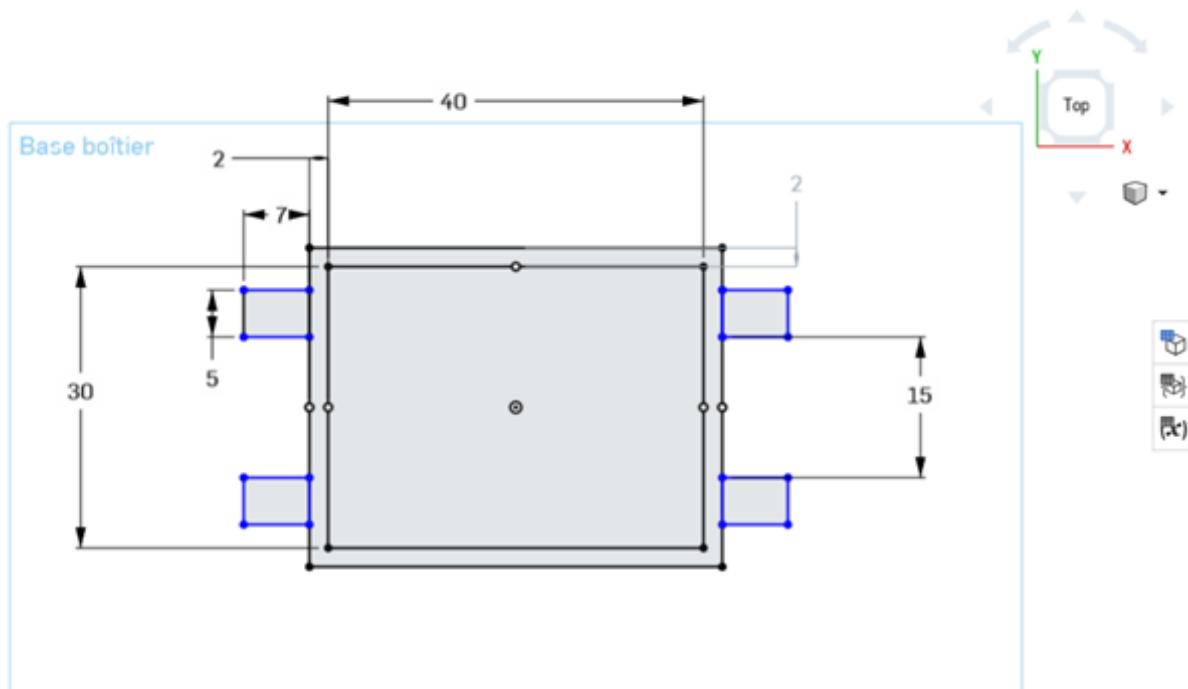


FIGURE 77 – Esquisse du boîtier dans le plan "top"

Une extrusion de 15 mm des contours selon l'axe Oz positif permet de créer les parois latérales. Il ne faut pas oublier les branches de 3,2mm selon l'axe Oz positif.



FIGURE 78 – Extrusion de l'extérieur.

Enfin, on obtient le fond du boîtier par l'extrusion sur 2mm du rectangle central :

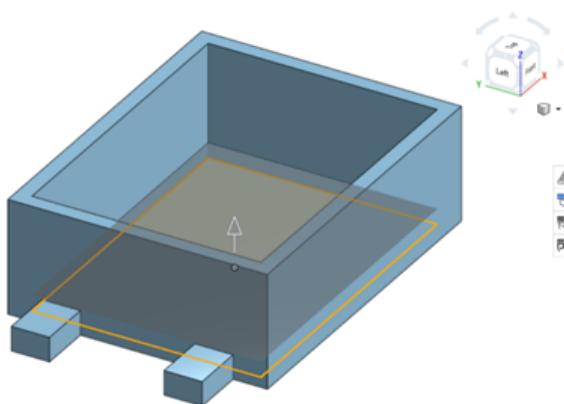


FIGURE 79 – Extrusion de l'intérieur

Il faut ensuite créer les fentes sur les faces latérales à gauche et à droite pour que les câbles des actionneurs puissent passer. Nous avons délibérément choisi de faire de gros trous, cela facilitera le premier montage du prototype. On pourra par la suite, lorsque tout fonctionnera et que les dimensions du câblage seront connues précisément, décider de réduire leur taille afin d'avoir une finition optimale. Ainsi, on crée une esquisse sur une face latérale avec les dimensions ci-dessous.

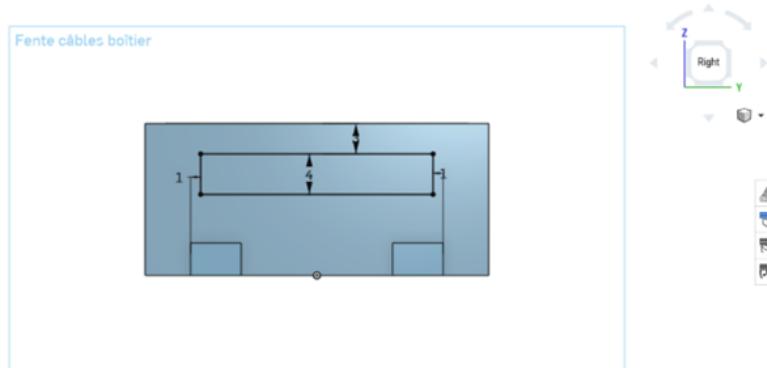


FIGURE 80 – Esquisse pour la création des fentes latérales

Une simple suppression en sélectionnant l'esquisse précédente nous amène aux fentes voulues :

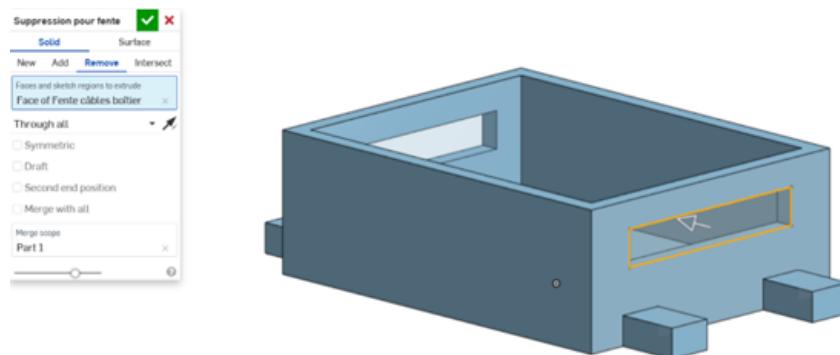


FIGURE 81 – Finalisation des fentes

Il faut maintenant s'occuper de la partie inférieure du boîtier, à savoir la courbure en contact avec le poignet et les trous dans les branches pour le système de maintien du bracelet. Pour cela, il va falloir créer une nouvelle esquisse dans le plan "Front" de notre fichier. Il s'agit de dessiner cette courbure en coupe. Il a une seule côte à imposer, la symétrie à gauche (ou droite) ainsi que la tangente de l'arc avec l'arête horizontale de face mèneront au résultat voulu.

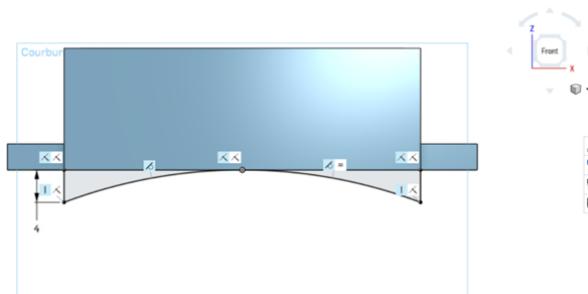


FIGURE 82 – Esquisse pour la courbure du boîtier

Il faut ensuite extruder cette esquisse selon l'axe Oy de manière symétrique. On coche donc l'option "Symmetric" et on rentre la valeur 34mm pour la profondeur.

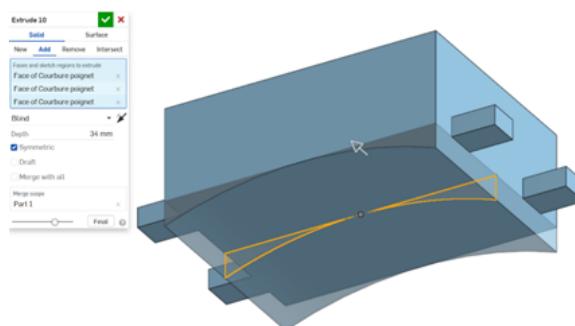


FIGURE 83 – Processus de symétrisation

Pour que les attaches du bracelet soient assez épaisses, il faut les étendre de 4mm selon Oz négatif comme ceci :

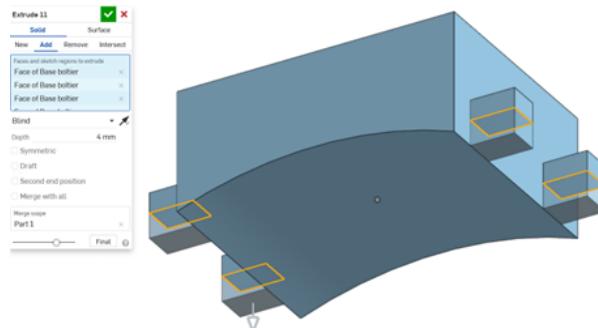


FIGURE 84 – Extension des attaches

Il est nécessaire ensuite de creuser le trou pour le maintien du bracelet. On crée donc une esquisse sur une face latérale des branches et on dessine deux cercles de 1.5mm de diamètre écartés de 5 mm du boîtier.

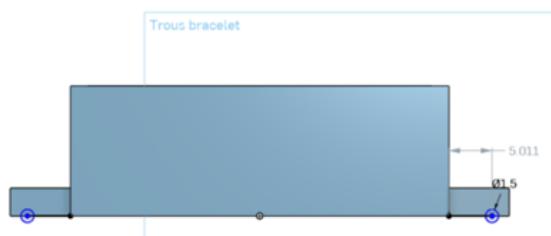


FIGURE 85 – Esquisse pour les trous du boîtier

En retirant les cylindres créés par les deux cercles, on obtient très simplement nos trous.

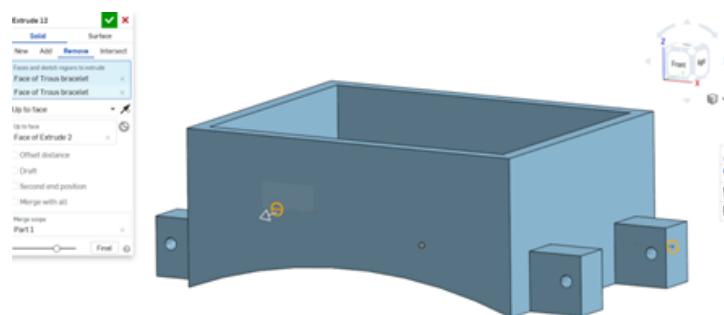


FIGURE 86 – Finalisation des trous du boîtier

Enfin, pour un design plus doux, il ne nous reste plus qu'à arrondir quelques angles et le boîtier sera fini. Voici les rayons choisis :

- 2.6mm pour les arêtes verticales intérieures et extérieures du parallélépipède ainsi que pour celles de la courbe inférieure.
- 4.3mm pour l'arrondi inférieur des branches
- 3.2mm pour l'arrondi supérieur des branches.

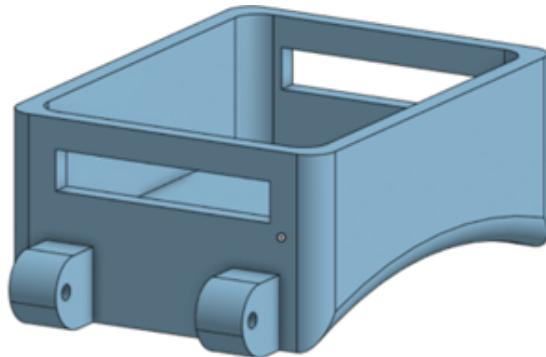


FIGURE 87 – Congés du boîtier

Un couvercle a aussi été modélisé selon l'esquisse ci-dessous puis extrudé de 2mm pour la surface et de 3.5mm pour le contour qui permet la tenue au boîtier.

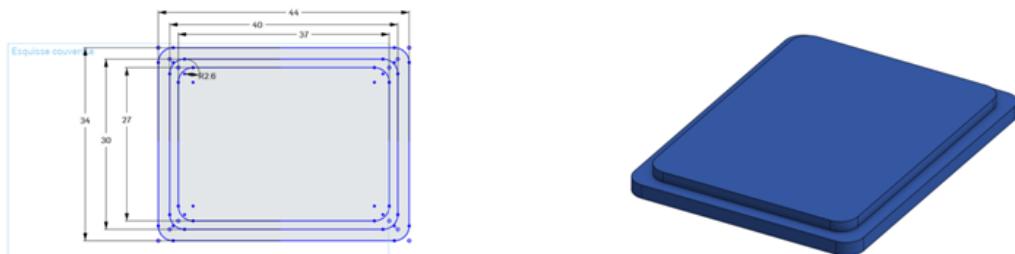


FIGURE 88 – Modélisation du couvercle

Le boîtier est enfin fini, nous l'avons exporté en format STL pour l'impression.

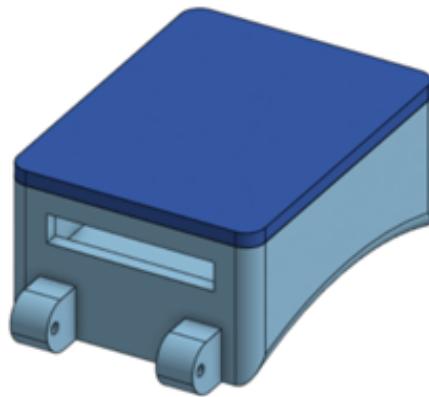


FIGURE 89 – Première modélisation terminée

6.2.3 Première fabrication

La première impression 3D de ce moule a été réalisée sur une imprimante 3D à filaments, afin d'obtenir un premier rendu et de s'assurer que la courbure du boîtier convenait. L'objectif n'était pas d'obtenir un état de surface parfait nous avons donc choisis d'imprimer en PLA, avec de couches de 0.15mm ce qui correspond aux réglages par défaut



FIGURE 90 – Premières impressions

de l'imprimante. L'imprimante utilisée était une prusa i3 Mk3 S/S+. Voici le résultat de cette impression :

La qualité en surface était convenable et la forme du boîtier convenait parfaitement à n'importe quel type de poignet. Néanmoins certains points étaient décevants :

- Une partie des supports utilisés par l'imprimante ne pouvait être détachée de la partie inférieure du boîtier.
- Les trous pour la fixation du bracelet n'étaient pas ronds et n'étaient pas exploitablest tel qu'on le souhaitait.
- Le couvercle ne se fermait pas correctement.

6.2.4 Deuxième fabrication

Une fois les paramètres principaux vérifiés lors de la première impression, nous avons réalisé un deuxième prototype dans l'objectif d'avoir cette fois-ci quelque chose de rapprochant le plus possible d'un produit fini. Dans cette optique, nous avons réalisé cette V2 dans un autre matériau : la résine. La méthode d'impression est différente et permet d'avoir des états de surface beaucoup plus propres et précis. L'épaisseur des couches d'impression demeure néanmoins inchangée. Voici le résultat :



FIGURE 91 – Deuxièmes impressions

Ce deuxième produit était nettement plus convaincant :

- une fermeture du couvercle ajustée grâce à la grande précision de l'imprimante
- un très bon état de surface malgré quelques artefacts dus au support
- un modèle légèrement plus fragile mais également plus léger
- une sensation agréable sur le poignet

6.2.5 Deuxième conception

Après concertation avec le pôle électronique, il était nécessaire d'agrandir le boîtier afin d'inclure de nouveaux éléments. Pour cela, nous avons travaillé sur une modélisation de l'encombrement de l'ensemble des pièces électroniques dans le boîtier pour ainsi proposer une solution adéquate. Seules des fonctions `move face` pour la réalisation de ce dernier ont été nécessaires.

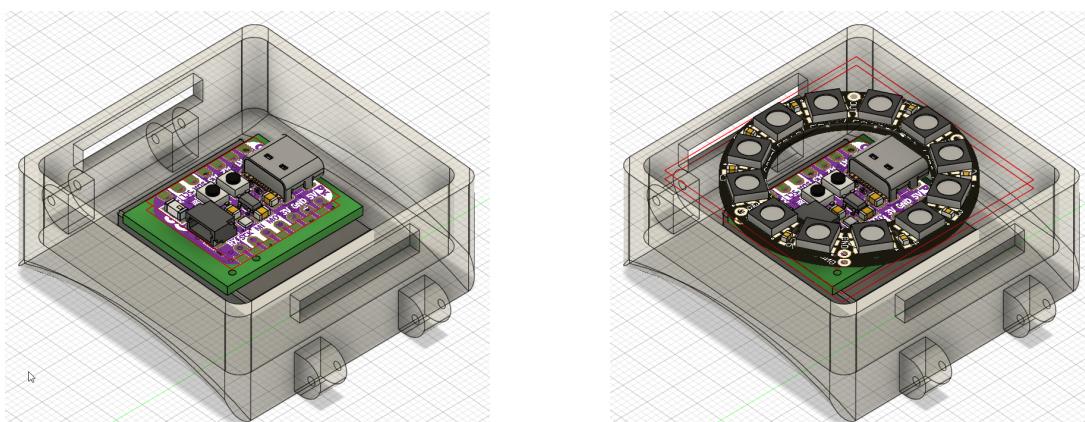


FIGURE 92 – Modélisation de l'encombrement du boîtier

Nous avons également créé une nouvelle pièce : un déflecteur pour l'anneau LED, qui permet d'atténuer l'intensité des LED pour un meilleur confort de lecture pour l'utilisateur.

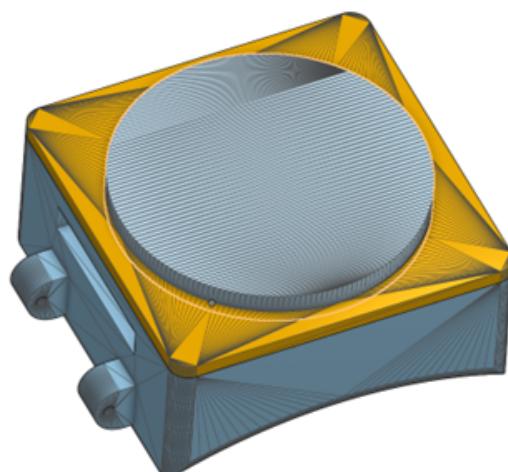


FIGURE 93 – Deuxième modélisation terminée

6.2.6 Troisième fabrication

Pour le boîtier nous avons réutilisé l'impression résine car le résultat était très satisfaisant. Pour ce qui est du diffuseur nous l'avons imprimer avec une imprimante à filaments Zortrax ce qui nous a permis d'utiliser de la fibre de verre afin d'avoir un rendu transparent.

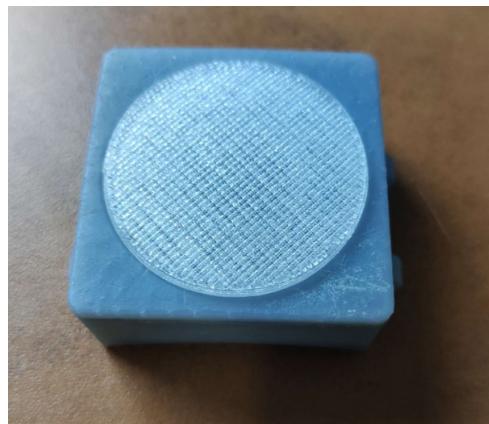


FIGURE 94 – Troisième fabrication terminée

6.3 Conception et réalisation du bracelet

6.3.1 Contexte

Lors du précédent projet portant sur un bracelet connecté, c'est un bracelet déjà pré-existant qui a été utilisé pour tenir le boîtier. Dans le cadre du projet Orion nous avons préféré concevoir et fabriquer nous-même le bracelet afin d'avoir la possibilité de personnaliser totalement le résultat et notamment de pouvoir améliorer l'ergonomie et intégrer les actionneurs à l'intérieur même de ce bracelet. Il a fallu plusieurs étapes de brainstorming pour le choix de la formule (intégrer les actionneurs au sein du bracelet ou non) et surtout du matériau ainsi que du moyen de fermer ce bracelet efficacement. Notre choix s'est porté sur le silicone (plutôt que le cuir, le plastique, le métal, le bois articulé, ou un matériau textile). Nous allons donc dans cette section présenter nos différents choix de conception et de fabrication concernant le bracelet et les moules utilisés pour produire celui-ci.

6.3.2 Première conception

Dans un premier temps on choisit le silicone comme matériau utilisé plutôt que plastique, céramique, cuir, textile ou bien métal, car c'est la solution répondant au mieux aux attentes du cahier des charges :

- Moulable au FabMéca à Centrale afin d'obtenir une forme ergonomique et de pouvoir intégrer les actionneurs dans le bracelet (**FC1.1**)
- Confort (toucher doux) (**FC1.1**)
- Résistance à l'humidité et la sueur (**FC3**)
- Résistance à l'utilisation intensive (résistant aux égratignures) (**FC3**)
- Élasticité (résistance / contact avec la peau) (**FC2.1 et FC3**)
- Légèreté du matériau (**FC1.1**)
- Transmission modérée des vibrations (**FC2.2**)

Il faut tout d'abord modéliser le bracelet. On utilisera aussi le logiciel Onshape comme pour le boîtier. On commence donc par faire une esquisse de la base du bracelet. Notre choix des côtes se base sur celles des bracelets de nos propres montres. Pour ce qui est des trous pour les actionneurs, nous utilisons les mesures des ces derniers que nous avions à notre disposition.

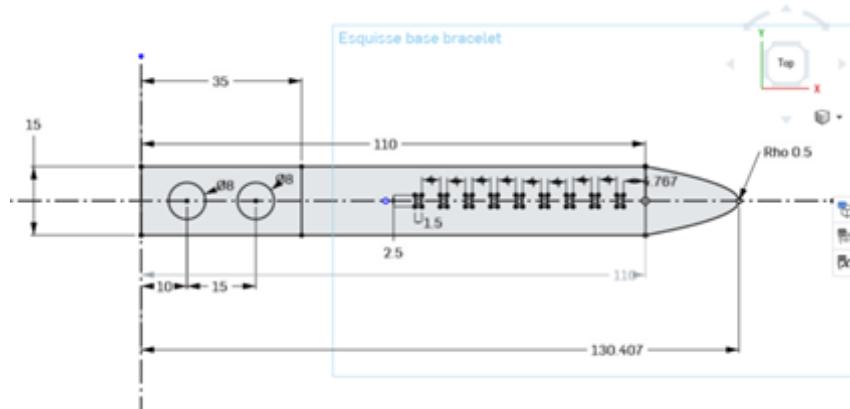


FIGURE 95 – Esquisse du bracelet

De cette esquisse il est ensuite possible de l'extruder. On épaisse davantage la partie droite afin d'avoir la place d'accueillir les actionneurs.



FIGURE 96 – Extrusion du bracelet

Il ne reste plus qu'à faire le trou horizontal pour la tige de maintien au boîtier. Comme pour le boîtier, on fait une esquisse sur la face latérale du bracelet et on dessine un cercle de 1.5mm de diamètre. On viendra par la suite supprimer le cylindre créé. Enfin, on crée les congés et arrondis pour obtenir le design souhaité.

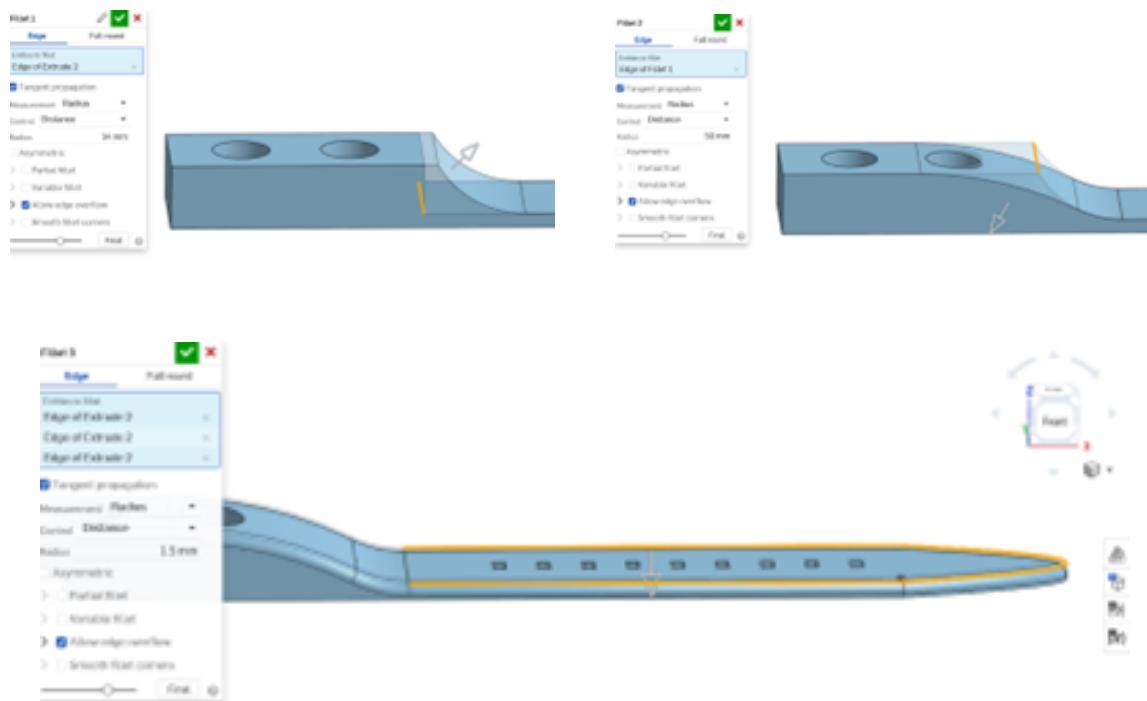


FIGURE 98 – Visualisation des différents congés du bracelet

Nous avons ainsi la première pièce du bracelet. L'autre partie n'est pas percée et doit pouvoir tenir l'ardillon que nous souhaitons utiliser. On fait le même travail que pour la première partie mais de longueur réduite. On crée le petit support pour l'ardillon en extrudant le bout comme ceci :

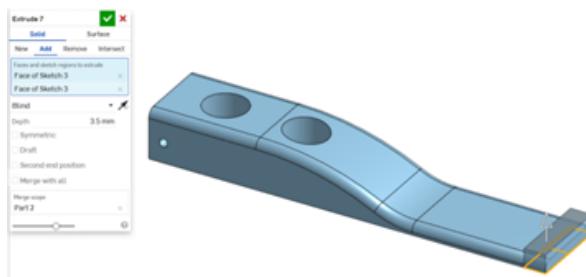


FIGURE 99 – Support ardillon

La modélisation est maintenant terminée, il ne reste plus qu'à créer le moule avant de pouvoir couler le silicone.

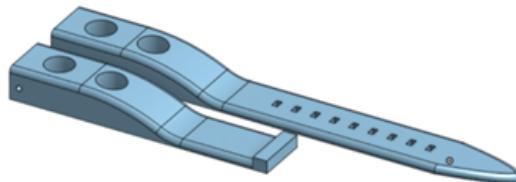


FIGURE 100 – Première modélisation bracelet

Une fois les formes du bracelet adaptées, on réalise un négatif de ce dernier dans un bloc massif de matière en utilisant la fonction Boolean sur Onshape. Après cette opération il faut vérifier manuellement l'absence de contre-dépouille et réaliser de faibles dépouilles sur les surfaces perpendiculaires au plan de joint afin de faciliter le démoulage du bracelet. Une fois ces étapes réalisées on creuse le moule afin de pouvoir injecter le silicone à l'intérieur. Il est important de réaliser un trou conséquent afin de créer une réserve de matière lors de la coulée.

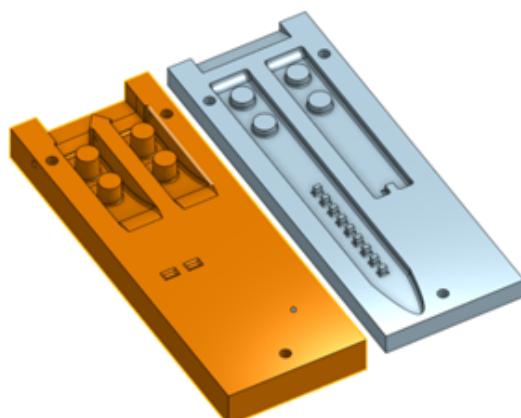


FIGURE 101 – Moule pour la coulée

6.3.3 Première fabrication

Suite à la conception du moule et au choix du matériau, il faut ensuite l'utiliser pour couler le bracelet en silicone, et pour cela nous avons utilisé le silicone *SORTA-Clear* du fournisseur *Smooth-On*.

- La première étape consiste en la préparation du silicone liquide qui va polymériser, en mélangeant la solution de silicone liquide et le durcisseur permettant la polymérisation (mélange opéré selon les proportions indiquées dans la notice écrite par le fournisseur du silicone, ici 1 :1). Et pour faciliter la coulée, on procède avant l'ajout du durcisseur à l'ajout de fluidifiant à la solution (sans dépasser les 10% de la masse de silicone), pour que le silicone se répande bien jusque dans les plus petits recoins du moule. (on peut également ajouter un colorant si l'on désire une couleur en particulier, dans ce cas nous avons ajouté quelques gouttes de colorant bleu).
- Pour éliminer les potentielles bulles d'air pouvant apparaître durant la polymérisation, on laisse alors la solution reposer pendant quelques minutes sous-vide, jusqu'à ne plus observer de bulle ou dégagement gazeux provenant de la solution.
- On badigeonne ensuite chaque partie du moule avec la solution, tout en passant dans les recoins et les angles avec un fil de fer pour éviter la formation de toute bulle d'air pouvant fragiliser le bracelet une fois la polymérisation finie (on perce également tout bulle d'air en surface de la solution).

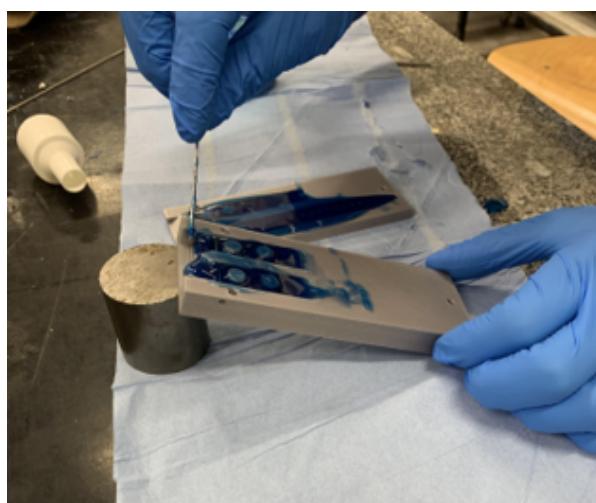


FIGURE 102 – Élimination des bulles d'air sur le moule badigeonné de silicone

- On réunit les deux parties du moule que l'on solidarise avec 3 vis passant de part en part dans les 2 parties du moule et 3 écrous. On insère ensuite un fil de fer de diamètre 1,2mm de part en part du moule pour obtenir des trous sur chaque partie du bracelet qui permettront de faire passer les barres de fixations au boîtier. On fait passer un deuxième fil du même diamètre mais à l'autre extrémité du moule et au travers de seulement la moitié de celui-ci, pour obtenir un trou sur une des parties du bracelet afin de pouvoir y loger la boucle ardillon qui permettra de fermer le bracelet.
- Enfin on redresse le moule à la verticale pour remplir la forme d'entonnoir dans le moule avec le silicone liquide qui va pouvoir couler et remplir les parties restant vides à l'intérieur du moule. (L'idéal est en réalité de couler le silicone dans le moule

sous vide, mais le matériel permettant cela n'est pas en service au laboratoire de mécanique).

- Il s'agit alors de laisser reposer le tout durant environ 8 heures afin de laisser la polymérisation se dérouler correctement.



FIGURE 103 – Moule rempli et mis à la verticale

- Une fois la polymérisation finie, il suffit de desserrer le moule pour en extraire le bracelet fini qui ne colle que très peu au moule. Nous avons alors procédé à quelques finitions au scalpel pour ajuster de légères bavures.



FIGURE 104 – Bracelet obtenu après démoulage

6.3.4 Deuxième conception

Après les tests sur le premier modèle, il nous semblait nécessaire, pour un meilleur ressenti des actionneurs, de refaire un bracelet en silicone mais cette fois-ci en changeant sa forme au repos en pré-courbant le bracelet.

Cette volonté va bouleverser la modélisation, en effet nous avons travaillé précédemment avec une esquisse de la partie inférieure du bracelet plane qui est désormais courbée. Pour pallier ce problème, il faut donc revoir l'orientation de la première esquisse.

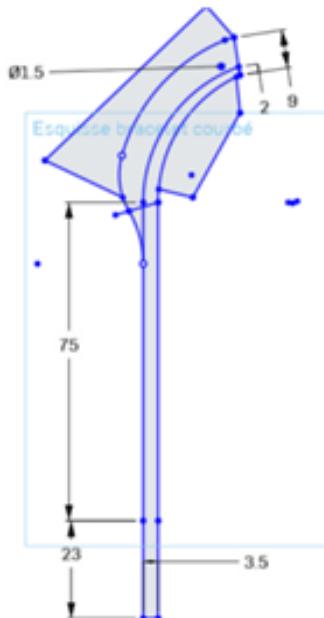


FIGURE 105 – Nouvelle esquisse du bracelet

L'extrusion de cette esquisse nous amène à la forme principale du bracelet. On crée ensuite les mêmes cavités et trous que le modèle plat.

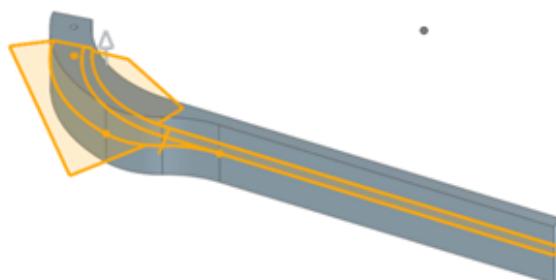


FIGURE 106 – Extrusion du nouveau bracelet

De la même manière que les autres modélisations, on améliore l'esthétique finale grâce aux congés et arêtes.

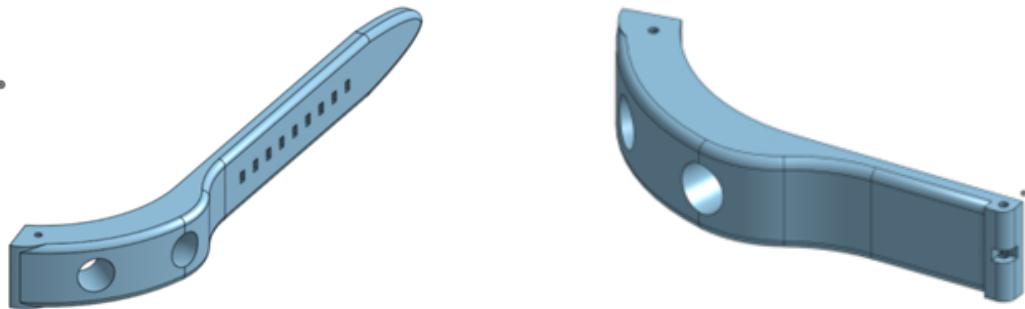


FIGURE 107 – Visualisation des congés du nouveau bracelet

Une fois la modélisation terminée on répète les mêmes étapes que lors de la première conception pour avoir un moule, bien que cette fois-ci on se permet un plan de joint courbé.

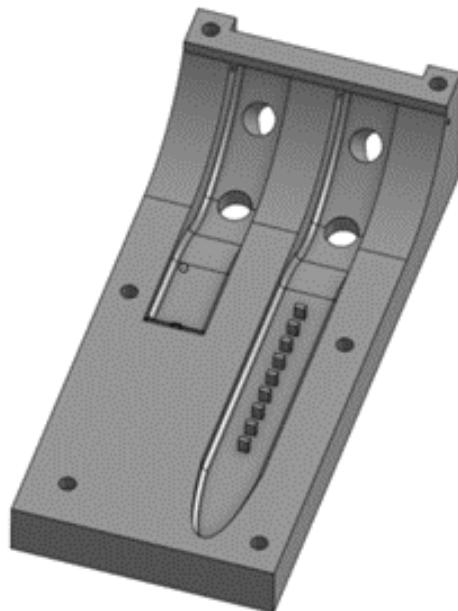


FIGURE 108 – Moule du nouveau bracelet

6.3.5 Deuxième fabrication

Pour cette fabrication, on répète les mêmes opérations que précédemment. Néanmoins il a été nécessaire d'usiner des moyeux pour réaliser les trous dans le bracelet final. Nous avons pour cela utiliser des barres d'acier normalisées à 8mm de diamètre que nous avons découpées et sur lesquelles nous avons réalisé des chanfreins pour faciliter l'insertion de ces dernières. L'alésage sur le moule a également été retravaillé à l'aide d'un alésoir afin d'avoir un ajustement dit "ajusté" (afin de pouvoir retirer ces moyeux sans soucis en ne laissant pas de fuite possible pour le silicone).



FIGURE 109 – Deuxième fabrication du bracelet

6.4 Eléments standards

6.4.1 Choix des éléments standards

Pour permettre de relier le bracelet en silicium au boîtier, ainsi que pour permettre de fermer le bracelet, nous avons employé des pièces standardisées utilisées habituellement dans l'horlogerie traditionnelle, car ce sont des solutions simples, solides et éprouvées depuis longtemps. De plus, il existe de très nombreuses dimensions de pièces disponibles sur internet. Nous avons donc sélectionné ces solutions :

- Des barrettes à ressort pour lier le bracelet au boîtier. Nous avons opté pour un diamètre de 1,2mm, et une longueur de 15mm correspondant à l'écart entre les "cornes" (terme horloger désignant les proéminences sur le boîtier permettant l'attache du bracelet). Ces barrettes à ressorts sont à manipuler avec un pointeau de pose que nous avons également dû commander pour pouvoir fixer ou désolidariser le bracelet avec le boîtier. En effet l'avantage de ce système est que malgré les différentes versions du bracelet et des boîtiers, on peut l'utiliser dans tous les cas de figure.



FIGURE 110 – A gauche : Pointeau de pose, à droite : barrettes à ressort

- Une “boucle ardillon”, système de fermeture classique permettant de relier les deux parties du bracelet et d’ajuster la taille de celui-ci en insérant la barre de la boucle ardillon dans un des trous présent sur l’autre partie du bracelet. L’ajustement de la taille permet le confort de l’utilisateur et de bien maintenir les actionneurs en contact avec le poignet de l’utilisateur. Nous avons opté pour un modèle avec une largeur de 16mm pour correspondre à la largeur du bracelet.



FIGURE 111 – Boucle ardillon

6.4.2 Implémentation

Suite au choix des pièces standardisées et leur réception, leur utilisation a posé certaines questions et a nécessité quelques ajustements.

- Un problème est que sur les sites pour commander les pièces horlogères, toutes les cotes ne sont pas détaillées ce qui fait qu’au lieu de la barrette à ressort de 15mm de long nous avons dû utiliser des barrettes de 16mm.
- Le diamètre de la boucle ardillon n’était pas précisé sur le site du fournisseur, alors que le bracelet était déjà moulé. Le diamètre était légèrement supérieur à celui du perçage prévu à cet effet mais l’élasticité du silicone permet que cela fonctionne tout de même et maintient convenablement la boucle.
- L’utilisation du pointeau de pose demande de prendre le coup de main, il s’agit d’en glisser la pointe entre la corne et le bracelet pour presser sur la barrette à ressort et l’extraire du logement dans la corne.

6.5 Conception et réalisation de la membrane

Afin de faire la liaison avec les aimants des actionneurs il était nécessaire de réaliser des membranes souples. Pour ce faire nous avons modélisé un moule simple sur onshape comportant différentes épaisseurs de membranes afin de voir les différences de ressenties en fonction de leur épaisseur :

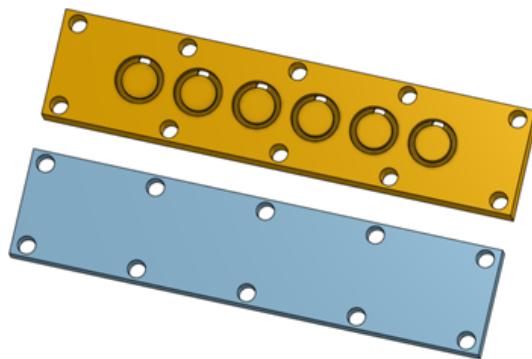


FIGURE 112 – Moule membrane

La coulée s'est faite en PDMS (référence : "RTV-4250-S"), un silicium plus souple que celui utilisé pour le bracelet. Elle a été réalisée par L'IEMN qui disposait d'un four spécial (4 heures de cuisson à 70°C) ainsi que d'un dispositif pour réaliser une coulée sous vide. Voici le résultat :

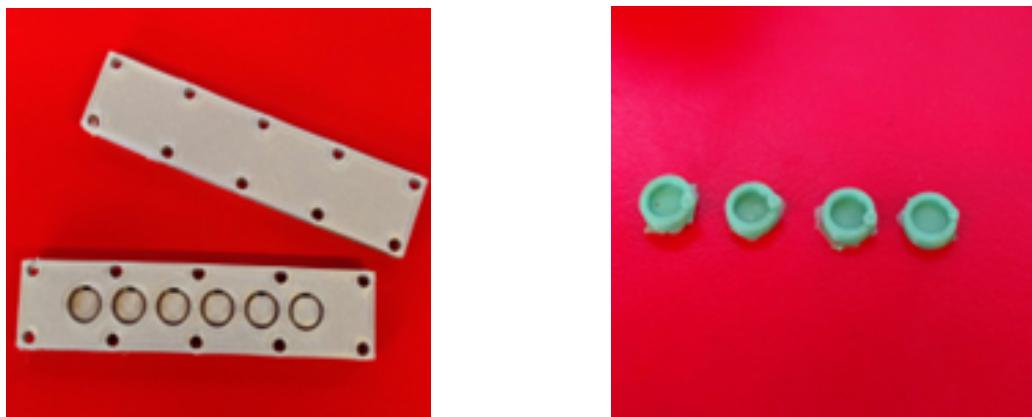


FIGURE 113 – Moule et coulée des membranes

6.6 Test du premier assemblage

Afin de tester l'assemblage du boîtier, du bracelet avec les pièces standardisées nous avons mis en place une première version avec le boîtier vide et sans anneau LED ainsi que la version plate du bracelet.



FIGURE 114 – Premier assemblage

Nous avons donc répertorié les points positifs et négatifs qui nous ont donc permis de diriger nos améliorations pour les versions suivantes du bracelet et du boîtier.

Points positifs	Points d'améliorations
<ul style="list-style-type: none"> — Agréable au porter (le bracelet se déforme bien pour prendre la forme du bras, la courbure du boîtier épouse bien la forme du poignet) — Facile à fermer — Pas trop encombrant sur le poignet — Esthétique plutôt discrète 	<ul style="list-style-type: none"> — Bracelet très flexible et qui présente de très léger signes d'usure (petites fentes) au niveau des angles — Les logements pour les actionneurs se déforment de manière significative, ce qui pourrait poser problème une fois ceux-ci implantés — Une fois le bracelet fermé, la partie réglable inutilisée du bracelet pend dans le vide

Ainsi ces observations nous ont menées à vouloir concevoir une deuxième version du bracelet qui est déjà courbée afin d'épouser la forme du poignet et d'avoir moins de déformations au niveau des logements des actionneurs.

6.7 Test du second assemblage

Le second assemblage comprend les versions améliorées des différents éléments de la montre. En effet, il intègre la dernière version du boîtier (V3) dont les dimensions sont revues légèrement à la hausse (cf. plans figure 130) afin de pouvoir accueillir toute l'électronique ainsi que l'anneau LED. On retrouve de plus sur le boîtier une nouvelle partie "diffuseur" en fibre de verre permettant de protéger l'anneau LED et d'en diffuser la luminosité. Concernant le bracelet, ce deuxième assemblage utilise la V2 de celui-ci. On a donc un bracelet déjà pré-courbé pour mieux épouser la forme du poignet du porteur.



FIGURE 115 – Second assemblage

Points positifs	Points d'améliorations
<ul style="list-style-type: none"> — Fermeture toujours aisée du bracelet — Possibilité d'intégration complète du système électronique au sein du boîtier et du bracelet — Diminution de la déformation des logements destinés aux actionneurs grâce à la courbure du bracelet épousant la forme du poignet — Confort au porter accru grâce à la courbure — La forme et la faible déformation des logements des actionneurs permettront un meilleur contact entre les actionneurs et la peau de l'utilisateur — Diminution des signes d'usures visibles par rapport au bracelet du premier assemblage grâce à la diminution de la flexion nécessaire du bracelet 	<ul style="list-style-type: none"> — La partie réglable inutilisée du bracelet n'est pas maintenu — Piste d'amélioration de la gestion des câbles reliant les composants du boîtier aux actionneurs dans le bracelet : possibilité d'intégrer les câbles à l'intérieur du bracelet pour des raisons de sécurité

6.8 Conclusion

Cette partie résume la totalité de notre travail de conception, de fabrication et de choix de pièces standardisées lors du projet Orion. A partir de ces indications et des plans que nous avons fournis, il est donc possible de répliquer les différents boîtiers et bracelets que nous avons pu produire (à condition évidemment de pouvoir réunir les matériaux et équipements nécessaires).

7 Prototypage

Le prototypage est une étape cruciale dans un projet puisqu'elle permet de fournir une première solution au client, en effet un prototype est la meilleure illustration possible de la faisabilité d'un projet. Mais aussi, cette étape permet à l'équipe projet de comprendre et d'identifier certains obstacles parfois inconnus jusqu'à lors.

Nous allons vous présenter dans cette partie nos différents prototypes. Cette présentation permettra de remettre en contexte les différentes versions des travaux effectués qui ont été expliquées dans la partie technique de ce rapport.

7.1 Premier prototype

Notre premier prototype mettait en place uniquement certaines fonctionnalités du cahier des charges.

Ce prototype comprenait :

- l'application pour smartphone, qui permettait d'associer chaque notification d'application à un mode de vibration,
- la carte électronique, qui récupérait l'information transmise par le téléphone et pouvait piloter les actionneurs,
- le bracelet et le *casing* qui pouvaient contenir les composants électroniques principaux.

Voici comment il se présentait :

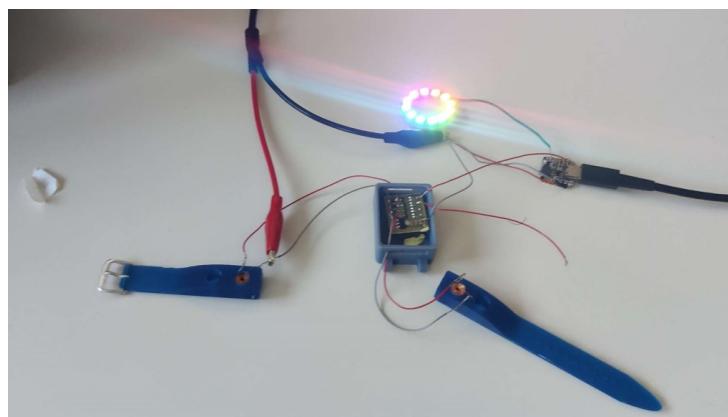


FIGURE 116 – Premier prototype

Cependant, nous avons remarqué plusieurs défauts importants :

- le *casing*, trop petit, qui ne pouvait pas accueillir tous les composants électroniques,
- l'alimentation non implémentée, devait se faire directement via un générateur,
- l'anneau LED, trop lumineux, qui pouvait gêner l'utilisateur.

L'ensemble de ces constats ont été nos principales pistes d'améliorations pour notre deuxième version.

7.2 Deuxième prototype

Notre deuxième prototype répondait ainsi à certaines des problématiques remarquées :

- le retravail du *casing*, permettait d'accueillir l'ensemble des composants électroniques,
- la présence d'une pile cylindrique, permettait d'alimenter la montre,
- l'ajout d'un déflecteur, permettait d'atténuer l'intensité de l'anneau LED.

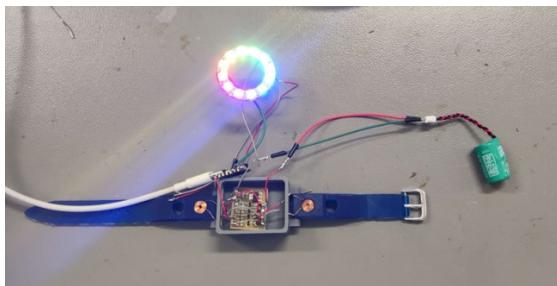


FIGURE 117 – Deuxième prototype. Sur la photo de droite le téléphone affiche *Connexion réussie*, phénomène aussi visible de par la couleur bleutée de l'anneau LED.

Néanmoins, même après ce retravail, d'autres défauts sont apparus :

- lors du port de la montre sur un poignet, les actionneurs avaient tendances à sortir de leur emplacement sur le bracelet,
- le circuit électronique, trop volumineux empêchait le convercle de se fermer complètement,
- la montre était incapable de mettre à jour correctement l'heure.

7.3 Troisième prototype

Notre dernier prototype cherche à répondre au maximum des problèmes identifiés :

- la modification de la forme du bracelet, permet de répondre au problème des actionneurs fuyants,
- le circuit électronique, retravaillé notamment avec une batterie plate, permet au convercle de se fermer complètement,
- la modification du code de l'application et de l'ESP32 permet de régler les soucis d'affichage d'heure.



FIGURE 118 – Troisième prototype

8 Valorisation

8.1 Charte graphique

Dès les premières semaines de travail nous avons cherché un logo comme identité visuelle pour le projet. Une première version a vite été proposée. Cependant, du fait de l'ambiguïté sur ce dont allait vraiment porter notre travail, cette dernière s'est retrouvée vite incohérente vis-à-vis de notre projet. Nous avons ainsi procédé à une refonte graphique de notre logo. Cette fois-ci, nous avons décidé de travailler autour d'un thème et d'un concept plutôt qu'autour d'un produit. Nous avons choisi le thème de l'espace pour son aspect fascinant qui suscite la curiosité et l'envie de découverte.

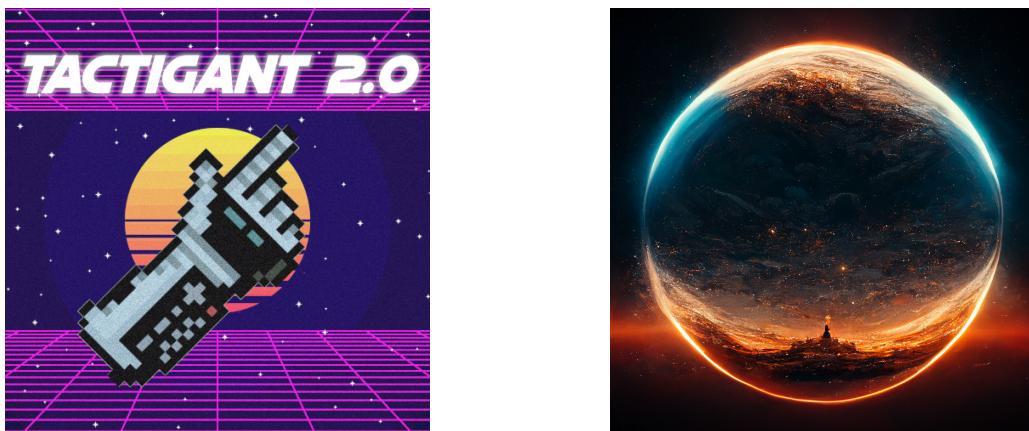


FIGURE 119 – Évolution des logos : à gauche le plus ancien, à droite le plus récent.

8.2 Réseaux sociaux

Les réseaux sociaux sont devenus un outil de marketing incontournable pour la commercialisation de produits. Ils permettent de toucher une audience large et ciblée, de créer de la notoriété et de l'engagement autour d'une marque ou d'un produit. De plus, les réseaux sociaux permettent de mesurer facilement l'impact d'une campagne de marketing grâce à un panel d'outils de suivi et d'analyse de performance. Egalement, les réseaux sociaux permettent aux commerçants de communiquer directement avec leurs clients en répondant à leurs questions et préoccupations.

On comprend ainsi pourquoi les réseaux sociaux sont considérés comme de puissants outils de commercialisation. C'est la raison pour laquelle nous avons décidé de créer un compte Instagram (visible en suivant ce lien) servant ainsi de vitrine pour notre produit.

8.3 Affiche et contenu promotionnel

La publicité est un élément crucial dans la commercialisation d'un produit. Elle permet de mettre en avant les caractéristiques et les avantages de celui-ci, ainsi que de lui donner une identité et une image de marque. Elle vise à attirer l'attention des consommateurs et à les inciter à acheter le produit en leur donnant envie de posséder ce qu'il offre.

Nous avons ainsi élaboré deux éléments publicitaires. Le premier est une affiche qui illustre les caractéristiques de notre produit.



FIGURE 120 – Affiche promotionnelle du projet Orion (cliquer ici pour voir en plus grand)

Le second est un film promotionnel qui à pour but de susciter l'intérêt du public et donner envie de découvrir notre produit.



FIGURE 121 – Extrait du film promotionnel (retrouvez la vidéo complète ici)

8.4 Partager une application

8.4.1 GitHub

Mise à part le partage direct des fichiers sources, la manière la plus naturelle de partager le code source d'une application est par l'intermédiaire de *GitHub*. Tant que le *repository* du projet est public, tout le monde peut télécharger le code avec la commande `git clone URL`. Cette première possibilité est principalement utilisée dans le cadre de projets collaboratifs itératifs, pour construire sur une version intermédiaire. De plus, *GitHub* permet la publication de *releases*, qui sont des versions considérées "stables" par le(s) propriétaire(s) du *repository*. Ce sont généralement des versions "prêtes-à-l'emploi", destinées aux utilisateurs finaux qui ne sont pas nécessairement concernés par le code en lui-même. Les *releases* sont parfois accompagnées de versions précompilées (exécutables) prévues pour ces mêmes utilisateurs qui souhaitent utiliser le programme sans devoir compiler le code eux-mêmes. Dans le cas d'une application Android, un fichier `.apk` est souvent joint à la *release*. Par ailleurs, on définit parfois *debug* comme le contraire de *release*.

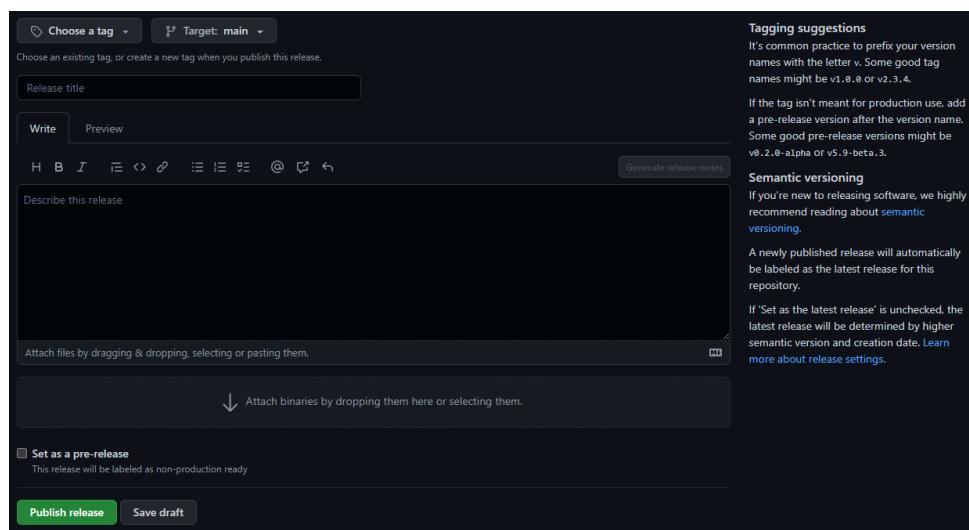


FIGURE 122 – L'interface de *GitHub* pour le partage de *release*

8.4.2 APK

Un fichier **APK** (*Android Package Kit*) est une archive du code source d'une application Android qui peut être générée directement depuis Android Studio. Une application Android peut directement être installée à partir de son APK une fois celui-ci copié sur le téléphone. Si une entreprise souhaite que son application soit téléchargeable directement depuis son site (c'est-à-dire sans passer par un *store*), il lui suffit de rendre un APK accessible.

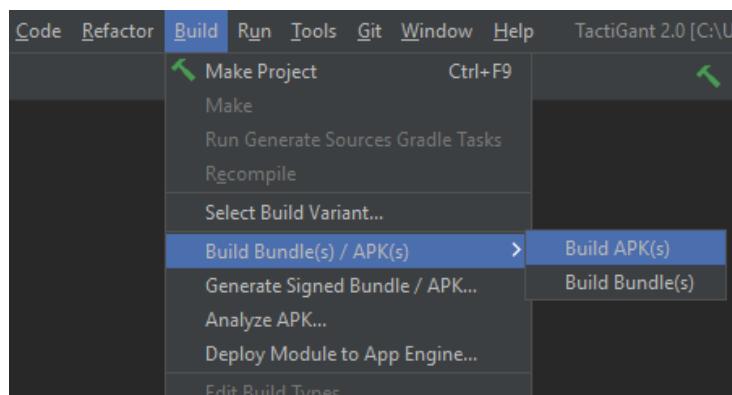


FIGURE 123 – Où cliquer pour générer un APK via Android Studio

8.4.3 Google Play Store

Tout développeur qui souhaite présenter son application à un maximum d'utilisateurs potentiels doit la publier sur le *Store* officiel des applications Android. Les étapes à suivre pour ce faire sont les suivantes :

1. Adapter le code à une version *release* : les éventuels *logs* et fichiers de tests doivent être retirés du code et les fichiers *Gradle* mis à jour. Ces derniers doivent contenir la mention `debuggable false` et un numéro de version.
2. Relire et tester l'application sur un maximum d'appareils.
3. Générer un AAB (*Android APP Bundle*) signé. Un AAB est une application Android compilée qui peut à son tour générer des APK. Lorsqu'un utilisateur installe l'application, l'AAB génère un APK "sur-mesure" en fonction du smartphone, ce qui réduit la taille des fichiers d'installation. Une "clé" est concrètement un ensemble de données cryptées sauvegardées par son propriétaire et qui permettent d'identifier ce dernier. Une clé de type *release* est générée depuis Android Studio et, toujours depuis Android Studio, l'application est signée à l'aide de cette clé. Du point de vue du développeur, le processus revient simplement à la création d'un mot de passe qui est ensuite renseigné au moment de générer l'AAB.
4. Mettre en ligne l'AAB sur le site *Google Play Console*. Ce site présente une interface permettant de gérer, entre autres, le prix de l'application, les pays dans lesquels elle est disponible ou sa page de présentation sur le *Store*. Il donne aussi accès à des statistiques permettant d'en mesurer le succès. La création d'un compte *Google Play Console* coûte 25 USD.
5. Publier l'application. *Google Play Console* permet de modifier en temps réel si l'application est téléchargeable ou non.

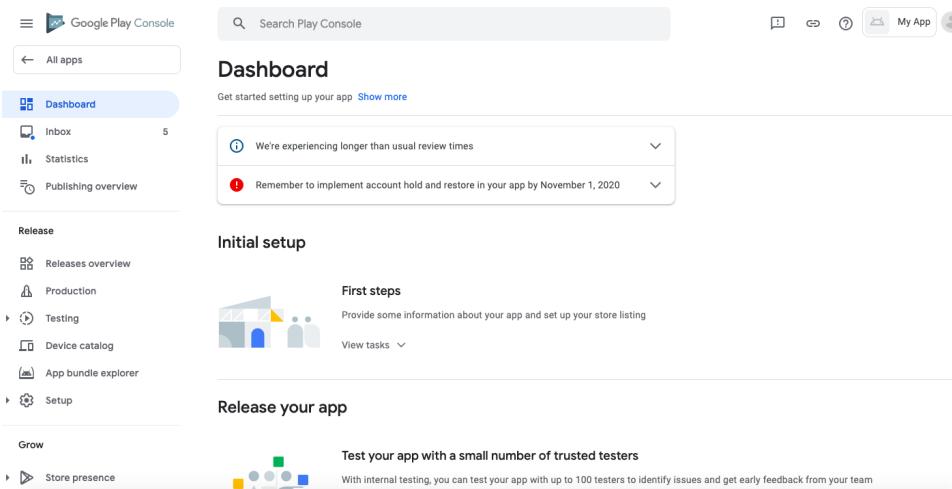


FIGURE 124 – La page d'accueil de *Google Play Console*

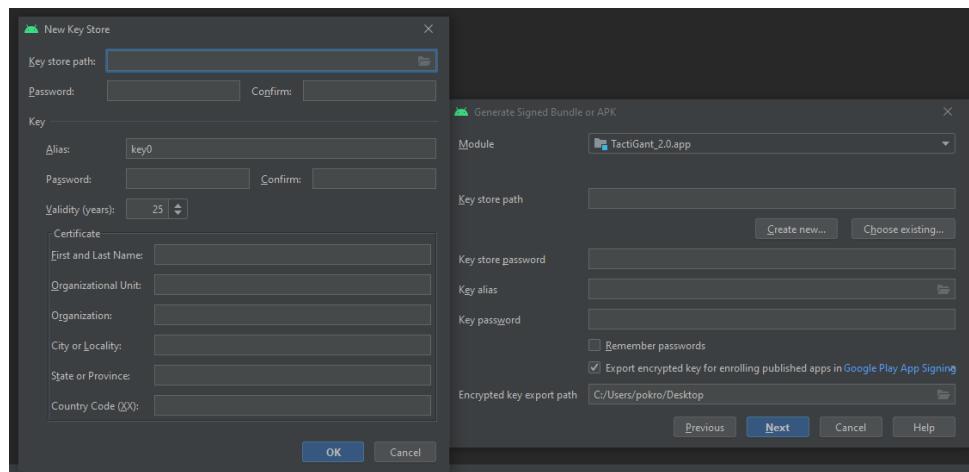


FIGURE 125 – Les interfaces de création de clé et de *Bundle* signé sur Android Studio

9 Suite du projet et améliorations

9.1 Pistes d'améliorations de notre projet

Même si la solution que nous proposons est fonctionnelle et répond aux problématiques imposées, il est important de présenter un ensemble de piste d'amélioration possible pour notre projet.

Concernant l'application pour smartphone, son plus gros défaut reste son exclusivité au marché Android. En effet, même si ce choix a été justifié et expliqué, il conviendrait, pour un projet souhaitant faire suite au nôtre, de proposer une solution fonctionnelle pour tout type de smartphone. Ainsi, le développement d'une application de type cross-platform pourrait être privilégié. Egalement, l'aspect interface utilisateur pourrait être amélioré, notamment sur la partie UI.

Concernant la partie électronique du projet, la piste d'amélioration principale serait de miniaturiser au maximum la solution. Cela signifierait de travailler avec du matériel spécialisé, très proche que ce que l'on peut trouver sur le marché actuel des montres connectées. Egalement, travailler sur une industrialisation des actionneurs, pour ainsi faciliter leur intégration ou leur possible remplacement, pourrait être une idée.

Enfin, pour la partie conception, la piste la plus intéressante serait sûrement de s'intéresser plus en détail sur les matériaux à sélectionner pour la réalisation des différentes pièces. En effet faire une étude plus approfondie que celle que nous avons réalisé sur le type de matériau à choisir pour un meilleur ressenti ou un meilleur confort pourrait être intéressant. Egalement, travailler sur l'intégration de la partie électronique (les actionneurs et leurs fils) directement dans le bracelet, c'est à dire venir positionner ces composants directement lors du moulage, pourrait être pertinent.

9.2 Autres idées d'applications de la technologie

Si la mise au point d'une montre connectée est un exemple concret et parlant des possibilités offertes par les actionneurs développés par l'IEMN, d'autres applications plus ambitieuses sont imaginables.

Une finalité de la miniaturisation d'objets vibrants est la possibilité de recréer, par l'intermédiaire d'une matrice de ces objets, des sensations tactiles extrêmement localisées. En jouant sur la fréquence et la position des vibrations, ces sensations peuvent être très proches du réel. Sinon, sans chercher à approcher des sensations réalistes, la petite taille de ces actionneurs permet d'ajouter une dimension tactile *localisée* à n'importe quel objet, plus élaborée qu'un simple vibreur.

Si l'idée d'ajouter une dimension tactile en faisant vibrer les manettes de jeux vidéos est loin d'être nouvelle, la possibilité de générer des vibrations "précises" ou proches du réel enrichirait encore les sensations perçues par le joueur. Plus généralement, l'idée serait de mettre sur un pied d'égalité les sens de la vue, de l'ouïe et du toucher, à l'instar du cinéma 4D. Des réseaux d'actionneurs placés sur le corps grâce à une combinaison, un plastron, un gant... peuvent être envisagés. Des applications couplées avec de la réalité virtuelle semblent naturelles. Ces deux derniers points mis ensemble suggèrent des applications dans le domaine en pleine évolution des métavers. Des retours haptiques qui

retranscrivent directement les interactions virtuelles décupleraient l'immersion. Ceux-ci aurait des applications riches dans le domaine des simulateurs, pour l'entraînement des pilotes ou des chirurgiens, par exemple.

En outre, des applications pour accélérer la récupération musculaire sont imaginables. Sachant que des systèmes d'électrodes à effet antalgique existent déjà, des vibrations très localisées pourraient peut-être permettre d'optimiser le procédé.

Nous avons mentionné le fait de mettre sur un pied d'égalité les sens de la vue, de l'ouïe et du toucher. Mais dans le cas où l'un de ces sens venait à manquer, il pourrait peut-être être "compensé" par des sensations tactiles riches. Pour compenser la cécité, un gant traduisant du texte en braille et le transcrivant en vibrations peut être imaginé. De plus, des vibrations plus ou moins intense selon la distance à un objet permettraient de prévenir les chocs, et la précision des retours tactiles aiderait à estimer la position de l'objet, et non pas uniquement sa distance. Un groupe d'actionneurs à retours de force variables permettrait de trianguler ledit objet.

De même, un militaire peut perdre temporairement l'usage de la vue ou de l'ouïe s'il se retrouve à proximité d'une explosion. La communication avec ses équipiers doit pouvoir continuer dans ces conditions, et l'emploi de petites vibrations peut être une alternative. Peut-être que des échanges en morse ou en braille, communiqués par vibrations, seraient une solution temporaire utile. Encore une fois, la petite taille du système vibrant permet de l'insérer facilement dans les équipements existants.

10 Conclusion

Pour conclure sur ce projet, nous vous avons présenté une solution répondant aux problématiques et au cahier des charges imposés. Nous avons ainsi réalisé une montre facilement contrôlable grâce à une application pour smartphone. Cette montre, alimentée par batterie, offre des sensations de toucher unique à l'utilisateur grâce à la technologie innovante proposée par l'IEMN. Enfin son bracelet et boîtier ergonomiques permettent un confort optimal d'utilisation pour l'utilisateur.

Nous souhaitions clore ce projet et ce rapport sur nos impressions individuelles sur ces 1 an et demi de travail en commun :

Mohammed CHADLI

Ce que j'ai aimé à propos du projet c'est qu'il touche à beaucoup de domaines ainsi entre membres d'équipe on devrait communiquer afin d'avancer dans le projet. De plus l'ambiance dans l'équipe était juste incroyable .Je ne regrette pas mon choix du projet.

Murilo DEL VECCHIO

Faire partie du projet Orion a été une expérience très enrichissante pour ma formation d'ingénieur généraliste à Centrale Lille. La réalisation d'un projet de cette taille et de cette importance était fondamentale pour l'application réelle des concepts et des outils étudiés. En tant qu'étudiant international, le projet a également contribué à mon intégration et à mon développement dans la langue française.

Jules DUMEZY

Ayant commencé le projet en tant que chef de projet, j'ai été heureux de pouvoir apporter mon expérience pour bien démarrer le projet. En effet, étant en G1' j'avais déjà participé à un autre projet, que j'ai malheureusement dû arrêter après un semestre. J'ai ensuite passé la casquette de chef de projet, afin de pouvoir plus me concentrer sur la partie électronique de commande.

Durant ce projet, j'ai eu l'occasion de développer de nombreuses compétences techniques en électronique, plus particulièrement sur les cartes électroniques permettant la communication bluetooth et leur programmation. Mais j'ai également pu profiter de cette expérience pour apprendre à travailler en équipe, et cela m'a beaucoup apporté sur le plan personnel et relationnel.

Ce projet, bien que challengeant, a été une expérience très positive qui me servira très certainement dans ma future vie professionnelle.

Mathis GUCKERT

Après cette année et demi de projet, je suis satisfait de ce qu'on a produit. On peut être quelque peu frustré d'avoir fait "si peu" en "autant" de temps, mais rester motivé et efficace sur 3 semestres n'a rien d'évident. Nous sommes allés au bout, en présentant un prototype et une documentation plus que solide, donc c'est très positif. J'ai aussi beaucoup appris, que ce soit en connaissances techniques pures ou en gestion de projet et organisation en équipe (notamment le travail collaboratif sur GitHub). Je me suis senti particulièrement à ma place au sein du pôle informatique, et notre cohésion d'équipe m'a aidé à rester motivé au mieux. Je pense que cet enseignement est important pour les futurs ingénieurs que nous sommes. Appréhender et mener à bien un projet sur le long

terme a été très enrichissant, et cette expérience nous servira, sans doute, dans notre vie professionnelle.

Paul GUILLAN

J'ai beaucoup apprécié travailler au sein de ce groupe projet tant pour le principe de valoriser la technologie de l'actionneur vibrant, que pour pouvoir vivre une première expérience de projet en groupe sur une aussi longue durée. J'ai pu acquérir des connaissances dans plusieurs domaines touchant à la conception d'une montre connectée, ce qui m'a d'autant plus motivé de par ma passion pour l'horlogerie. Au total, je suis fier du travail que nous avons accompli et des résultats obtenus avec "Orion".

Alexis KRAFT

J'ai trouvé ce projet très intéressant en raison de son caractère pluridisciplinaire, qui m'a permis de travailler avec des personnes ayant des compétences très différentes des miennes. Cela m'a donné l'occasion de me former et de me préparer à la collaboration dans un contexte professionnel, ce qui peut parfois être difficile entre les différentes parties impliquées dans un projet. Je suis plutôt satisfait du résultat final de ce projet, car nous avons pu proposer une solution qui répondait aux exigences de notre client. De plus, ce projet m'a permis de me former et de mettre en pratique de nombreuses connaissances en matière de fabrication lors de sa réalisation. Si je devais formuler une critique, ce serait à propos de la taille des groupes, qui selon moi est trop importante, ce qui peut entraver les processus d'innovation sans apporter une véritable valeur ajoutée par rapport à des équipes plus petites.

José Vitor MAKIYAMA

J'ai été satisfait du résultat du projet, bien que j'aurais aimé aider davantage, étant donné la façon dont je suis entré à la fin du projet en raison de ma situation d'échange, je pense que j'ai contribué autant que possible à la dernière ligne droite de ce projet extrêmement intéressant et enrichissant. Il convient également de noter que l'équipe était très accueillante et compréhensive, il était très satisfaisant de travailler avec eux sur ce projet.

Wassim MEJJAD

J'ai beaucoup apprécié ce projet. J'ai eu la chance de voir d'une part le côté technique avec le pôle électronique et dans un second temps plus l'organisation en tant que chef de projet. Donc ce projet m'aura énormément appris, en termes de compétence et en termes de connaissance. En conclusion j'ai adoré travailler sur ce projet et surtout avec cette équipe.

Thibaud PICCINALI

Concernant mon ressenti, je suis personnellement très satisfait de ce projet G1-G2, tant sur le travail réalisé que sur les moyens mis en place pour y arriver. En effet, je suis fier du prototype que l'on a proposé puisqu'il intègre parfaitement l'ensemble du travail de chacun et est quasiment fini. En ce qui me concerne, je suis très satisfait des nombreuses compétences que j'ai acquises, notamment dans le développement d'une application Android : c'est un sujet sur lequel j'ai toujours souhaité me former, et je peux désormais affirmer que je possède les compétences pour faire une application sous Android. Je suis également fier des compétences de management/gestion de projet que nous avons tous

développé dans ce projet. En effet, travailler avec un groupe de 13 étudiants est une tâche compliquée mais avec une bonne organisation et de bonnes méthodes on peut parvenir à tirer un résultat concluant. Je pense que cette expérience sera très utile dans notre future vie professionnelle ou ce genre de projet de groupe seront beaucoup plus fréquent. Enfin, j'aimerais ajouter que j'ai adoré travailler au sein de cette équipe, l'ambiance au sein du groupe était toujours excellente. Si au début du projet je voyais notre groupe comme une simple réunion d'élèves devant travailler sur un sujet, je vois aujourd'hui une bande d'amis qui travaille ensemble sur un projet commun dans la bonne humeur, l'entraide et la solidarité.

Lucas SALAND

Pour moi, ce projet a été une véritable opportunité de développer de nombreuses compétences techniques et sociales. En effet, pour ma part, le développement de l'application Android fut marquée par l'importance du travail d'équipe. Les réunions hebdomadaires ont permis un avancement régulier du projet en plus d'améliorer les rapports au sein de l'équipe. Ce fut également l'occasion pour moi de participer à un projet long et complet.

Omar SAUBRY

Globalement, je reste sur ma faim. Le projet m'a permis d'explorer certaines techniques de travail en électronique et d'enrichir mes connaissances en termes de conception d'un système. Cependant, je trouve que l'on a pas exploré certaines solutions techniques qui peuvent être, à mon sens, plus professionnelles et plus adaptées pour le développement d'une montre connectée.

Paul SYLVESTRE

J'ai apprécié cet projet très complet car touchant électronique, informatique et CAO. Le côté appliqué avec un prototype concret et fonctionnel à la fin donne une réelle finalité au projet.

Roman TIÉDREZ

Le fait de le mener sur un an et demi, mais aussi et surtout le fait de le partager avec douze autres membres motivés, font de ce projet une expérience unique et riche qui ont indéniablement redéfini pour moi le concept de "travail en équipe". Cet avant-goût du monde professionnel est une preuve par l'exemple de l'intérêt de prendre du temps en amont d'un projet pour se répartir des rôles et en construire les objectifs. Plus que tout, pouvoir apprendre des compétences concrètes et avoir matière à les mettre en application furent de puissants vecteurs de motivation. La satisfaction éprouvée lorsqu'on observe des fonctionnalités qu'on a soi-même développées à partir de zéro, tout en sachant que quelques mois plus tôt les technologies associées nous étaient inconnues, est immense. Si j'en sors avec, entre autres, la maîtrise d'un langage de programmation et la capacité de programmer des applications Android, les compétences humaines et le sens des responsabilités acquis au cours de ce projet en sont probablement les enseignements les plus précieux.

11 Bibliographie

Vous retrouverez dans cette partie l'ensemble des sources qui ont été utilisées au cours de ce projet. Chaque point de cette bibliographie est cliquable et renvoie vers la référence Internet en question.

11.1 Etat de l'art

- Article de Donald Melanson *Medic Vision intros haptic Mediseus Surgical Drilling Simulator*.
- Article sur la machine *Desktop 6D*.
- Article de Bobby Cummings *Tactically Reconfigurable Artificial Combat Enhanced Reality (TRACER)*.
- Article de Mariella Moon *UK military's bomb disposal robots come with haptic feedback*.
- Thèse de Bertrand Tornil *Adaptations et Interactions gestuelles et haptiques, ciblées utilisateurs*.
- Article de Victor PÉROUSE *Retour d'effort haptique et tactile*.
- Thèse de Sherry Turkle *Always On/Always-On-You: The Tethered Self*.
- Etat de l'art des technologies mobiles
- Article sur les types d'applications mobiles
- Tableau comparatif des types d'applications
- Parts de marché des systèmes d'exploitation pour mobiles
- Évolution des parts de marchés des systèmes d'exploitation mobile
- Page wikipédia sur iOS
- Article de Bartosz Szczygieł comparant Android et iOS
- Article de Setra *Montres connectées : Apple est toujours le leader, mais la concurrence est rude*.
- Site officiel de l'*Apple Watch*.
- Site officiel de la *Galaxy Watch*.
- Site officiel de *Fitbit*.
- Infographie Wikipédia Peau
- Article sur le toucher
- Article sur le traitement sensoriel
- Article Wikipédia *Neurosciences/Le toucher et la proprioception*

11.2 Pôle informatique

- Page internet *Material Design*.
- Documentation Android *Présentation des ressources d'application*.
- Documentation Android *Intents and Intent Filters*.
- Question Stackoverflow *Should I always use ConstraintLayout for everything ?*.
- Question Stackoverflow *How to implement a swipe-gesture between Fragments ?*.
- Question Stackoverflow *How to combine BottomNavigationView and ViewPager ?*.
- Question Stackoverflow *How to implement a swipe-gesture between Fragments ?*.
- Question Stackoverflow *What is the different between onCreate() and onCreateView() lifecycle methods in Fragment ?*.
- Page internet *Le Tutoriel de Android Service*.

- Page internet *ViewPager with FragmentPagerAdapter*.
- Page internet *Creating and Using Fragments*.
- Page internet *Intentions sous Android*.
- Page internet *Tutoriel Android : Apprendre à utiliser les ressources*.
- Page internet *MVVM (Model View ViewModel) Architecture Pattern in Android*.
- Documentation Android *Bluetooth basse consommation*.
- Documentation Android *Find BLE devices*.
- Question Stackoverflow *Unable to scan or discover BT and BLE devices in android*.
- Question Stackoverflow *How can I check bluetooth LE ScanResult with android ble scanning app ?*.
- Question Stackoverflow *Android 4.3: BLE: Filtering behaviour of startLeScan()*.
- Question Stackoverflow *Bluetooth Low Energy Scan Fails Immediately*.
- Page internet *Android BLE Issues*.
- Article de Martijn van Welie *Making Android BLE work — part 1*.
- Article de Martijn van Welie *Making Android BLE work — part 2*.
- Article de Martijn van Welie *Making Android BLE work — part 3*.
- Article de Chee Yi Ong *The Ultimate Guide to Android Bluetooth Low Energy*.
- Documentation Android *Préparer la publication de votre application*.
- Documentation Android *Signature d'applications*.
- Documentation Android *Publier votre application*.
- Documentation Android *À propos d'Android App Bundle*.
- Article de Avinash Sharma *How To Upload An App To Google Play Store?*

11.3 Pôle électronique

- Site de *Irf*.
- ESP32 PICO D4 Datasheet.
- Documentation du devkit ESP32-C3-devkitM1.
- Documentation de l'ESP-IDF.
- Documentation du module ESP32 Arduino.
- Amplification avec transistor.
- Site d'Adafruit.
- Schéma fonctionnel de l'ESP32
- Article de Lady Ada *Adafruit QT Py ESP32-C3 WiFi Dev Board*
- Article *Esp32 Bluetooth Low Energy BLE sur Arduino IDE*
- Site d'Adafruit pour l'anneau LED

11.4 Pôle conception

- Thèse de Hubert GUVIER *Silicones SI*.
- Page internet sur les différents type de fermoir de bracelet.
- Page internet sur les différents type de barrette de montre.
- Page internet sur les type de matériaux pour le bracelet d'une montre.

12 Annexes

12.1 Cahier des charges

CAHIER DES CHARGES FONCTIONNEL

TITRE DU PROJET	Orion	DATE DE MAJ	04/01/23	
FONCTION	ÉNONCÉ	CRITERES	NIVEAU(X)	FLEXIBILITÉ
FP1	Connexion du bracelet au téléphone de l'utilisateur grâce à l'application	Distance maximale de maintien de la connexion	Portée > 3m	2
FP2	Utilisation du bracelet pour des notifications de nombreuses applications	Fonctionne pour le réveil, les messages, les appels, la musique, les réseaux sociaux, le sport...		
FP3	Réglage des différentes options de vibration/retour haptique	Différents modèles de vibration/retour haptique sélectionnables	- Durée de vibration/retour (au moins 3 différentes) - Spatialisation des vibrations (au moins 3 différentes)	2
FP4	Synchronisation des vibrations/retours haptique à la musique	Différentes vibrations selon les fréquences et le tempo du morceau joué		
FP5	Activation et désactivation du bracelet	Possibilité d'allumer ou d'éteindre le bracelet sans passer par l'application : présence d'un bouton On/Off		
FC1.1	Le bracelet doit être ergonomique	Poids et forme adapté à l'utilisateur	- Poids total < 150g - Matériaux souples et non irritants	2
FC2.1	Les actionneurs doivent bien transmettre l'information tactile à l'utilisateur	Contact des actionneurs avec le poignet	Contact constant et uniforme	2
FC2.2	L'utilisateur doit sentir la "spatialisation" et les différentes intensités	Sensations au poignet	L'utilisateur peut aisément différencier l'intensité et l'emplacement des vibrations	1
FC2.3	Les actionneurs ne doivent pas déranger l'utilisateur	Comfort du bracelet les actionneurs éteints		2
FC3	Le bracelet doit résister aux chocs et aux contraintes habituelles (humidité, poussière, champs magnétique...)	Solidité, étanchéité et bonne fixation	- Capable d'être utilisé par temps pluvieux - Résistant à une chute < 2m	2
Carte de commande et énergie				
FP6	Transmettre aux actionneurs un signal délivré par l'application de manière instantanée	Temps de réponse	t5% < 200ms entre l'application et les actionneurs	1
FP7	Communication avec les actionneurs conçus par l'IEMN	Puissance et nombre de sorties suffisantes	- Puissance de 0,3125W par actionneur (à discuter) - au moins N sorties (N à définir)	1
FP8	Communication avec l'application dans des environnement variés	Utiliser une technologie sans-fil (Bluetooth/WiFi)	Puissance suffisante pour fonctionner à travers le tissu ou une cloison fine	1
FP9	Recharge facile	Utilisation d'une batterie et d'un port de chargement classique (USB)		
FP10	Comfort et simplicité			
FP11	Ne pas être trop complexe	Simplicité de développement (outils adaptés à des étudiants sans expérience)		
FP12	Respect de l'environnement	Matériaux et consommation d'énergie		
FC10.1	L'ensemble carte électronique + batterie ne doit pas être trop volumineux	Dimensions	Volume de moins de 15cm3	2
FC10.2	Le système doit avoir une autonomie suffisante	Autonomie	Autonomie d'au moins 6h	3
FC10.3	Le système ne doit pas être dangereux	Intensité du courant et force des actionneurs	Intensité de moins de 100mA et contact limité à l'électronique	3
FC11.1	Le système doit pouvoir être facilement débogué	Interface de débogage présente		
FC11.2	Le système ne doit pas être trop coûteux	Prix	Prix d'une unité à moins de 50€ en matières premières	3
FC11.3	Le système doit être rapidement démontable	Temps de démontage	Démontage en moins d'une minute	3
FC11.4	Le système doit avoir une durée de vie suffisante	Obsolescence	Système fonctionnant pendant au moins 2 ans	1
Actionneurs				
FP13	Conversion entre le signal bluetooth et la carte électronique	Communication possible		
FP14	Amplification du signal	Efficacité temporelle et énergétique		
FC15	Style	Produit "moderne" : retours visuels, affichage de l'heure...		
Application iOS-Android				
FP15	Application ergonomique	Application avec une GUI simple, lisible et design		
FP15.1	Application optimisée	Le code est efficace	Application tournant sans surcharger un téléphone entrée de gamme	2
FP15.2	Application facile à mettre à jour	Mise à jour aisée de l'application		
FP15.3	Application multiplateforme	Compatibilité avec Android et iOS		
FP15.4	Application permettant de régler le bracelet	Personnalisation des réglage et du bracelet		
FP15.5	Application accédant aux données Bluetooth/WiFi du téléphone pour communiquer avec le bracelet	Compatibilité avec les composant Bluetooth/WiFi des appareils		
FC16	Codage collaboratif	Possibilité de coder de manière collaborative		

FIGURE 126 – Cahier des charges fonctionnel (cliquer ici pour voir en plus grand)

12.2 Pôle informatique

- Autres types de filtres pour le scan :

Filtre par UUID :

```

1 UUID BLP_SERVICE_UUID = UUID.fromString("00001810-0000-1000-8000-00805f9b34fb");
2 UUID[] serviceUUIDs = new UUID[]{BLP_SERVICE_UUID};
3 List<ScanFilter> filters = null;
4 if(serviceUUIDs != null) {
5     filters = new ArrayList<>();
6     for (UUID serviceUUID : serviceUUIDs) {
7         ScanFilter filter = new ScanFilter.Builder()
8             .setServiceUuid(new ParcelUuid(serviceUUID))
9             .build();
10        filters.add(filter);
11    }
12 }
13 scanner.startScan(filters, scanSettings, scanCallback);

```

Filtre par nom :

```

1 String[] names = new String[]{"Polar H7 391BB014"};
2 List<ScanFilter> filters = null;
3 if(names != null) {
4     filters = new ArrayList<>();
5     for (String name : names) {
6         ScanFilter filter = new ScanFilter.Builder()
7             .setDeviceName(name)
8             .build();
9         filters.add(filter);
10    }
11 }
12 scanner.startScan(filters, scanSettings, scanCallback);

```

- Liste non exhaustive des méthodes `get` possibles pour la lecture d'informations reçues :

Type	Nom de la méthode	Retour
byte[]	getValue()	L'information reçue par le téléphone du service BLE
int	getWriteType()	Type d'écriture de la characteristic
UUID	getUuid()	UUID de la characteristic
int	getProperties()	Proritéés
int	getPermissions()	Permissions de la characteristic

12.3 Pôle électronique

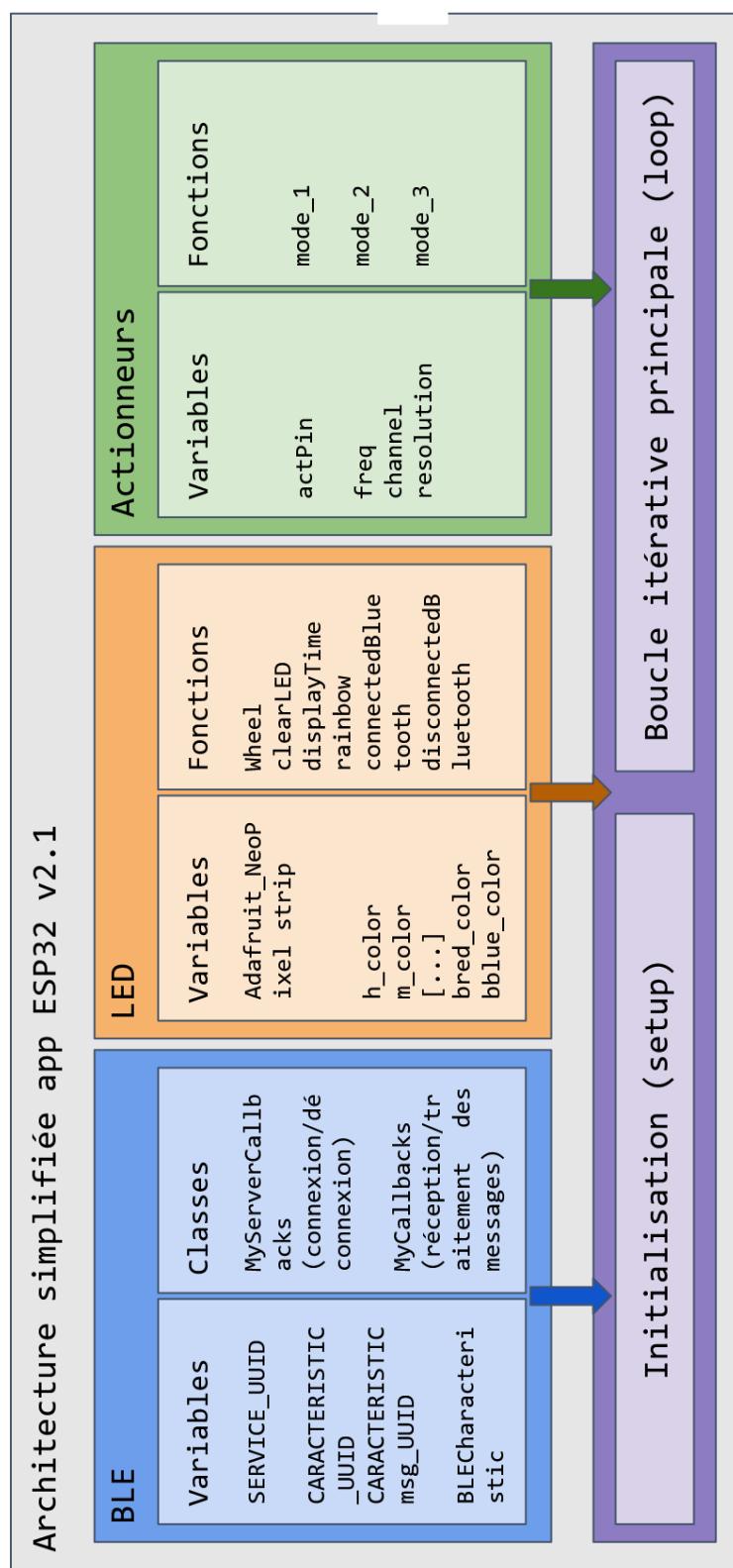


FIGURE 127 – Architecture simplifiée ESP32

12.4 Pôle conception

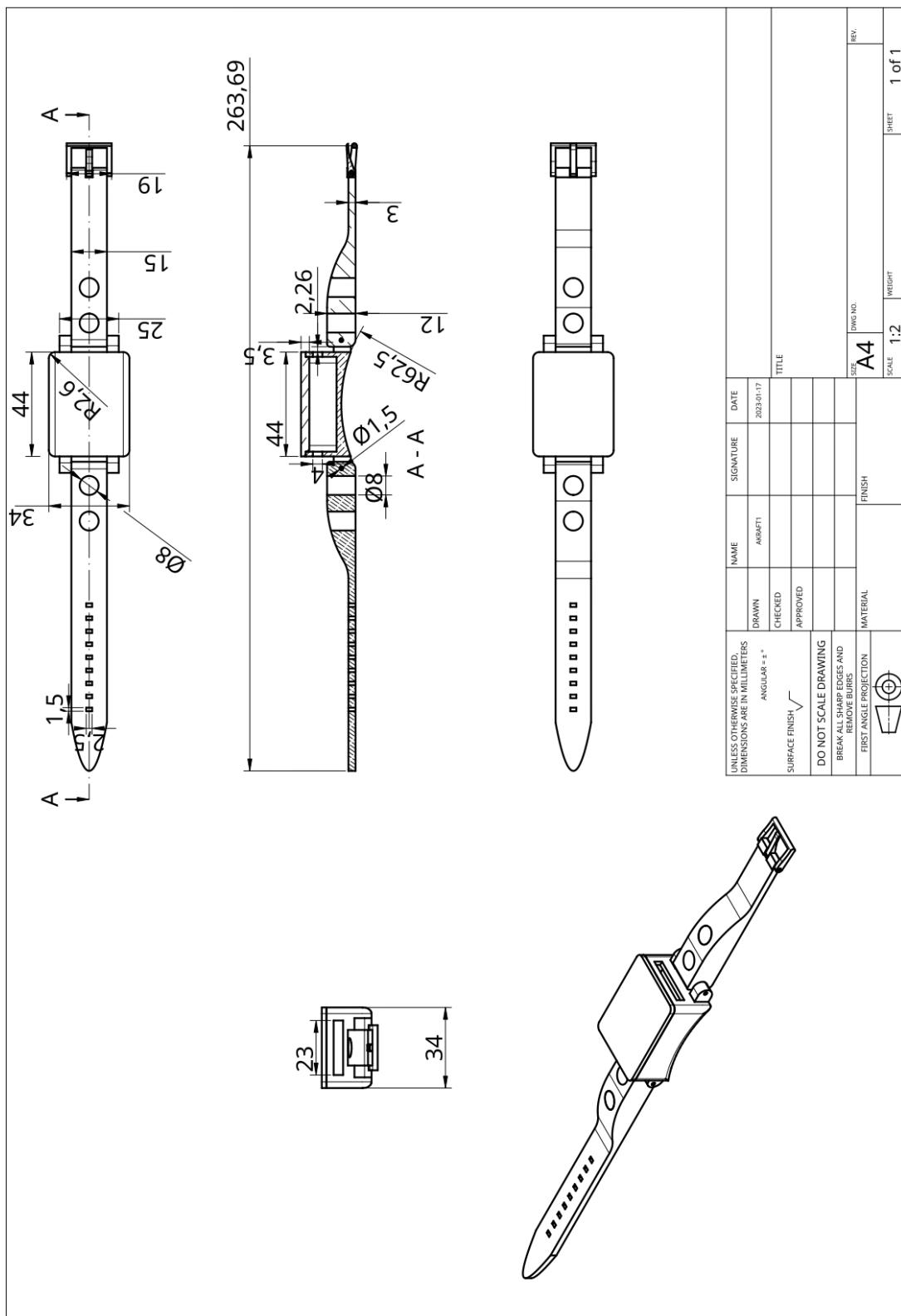


FIGURE 128 – Plan de la première version du bracelet

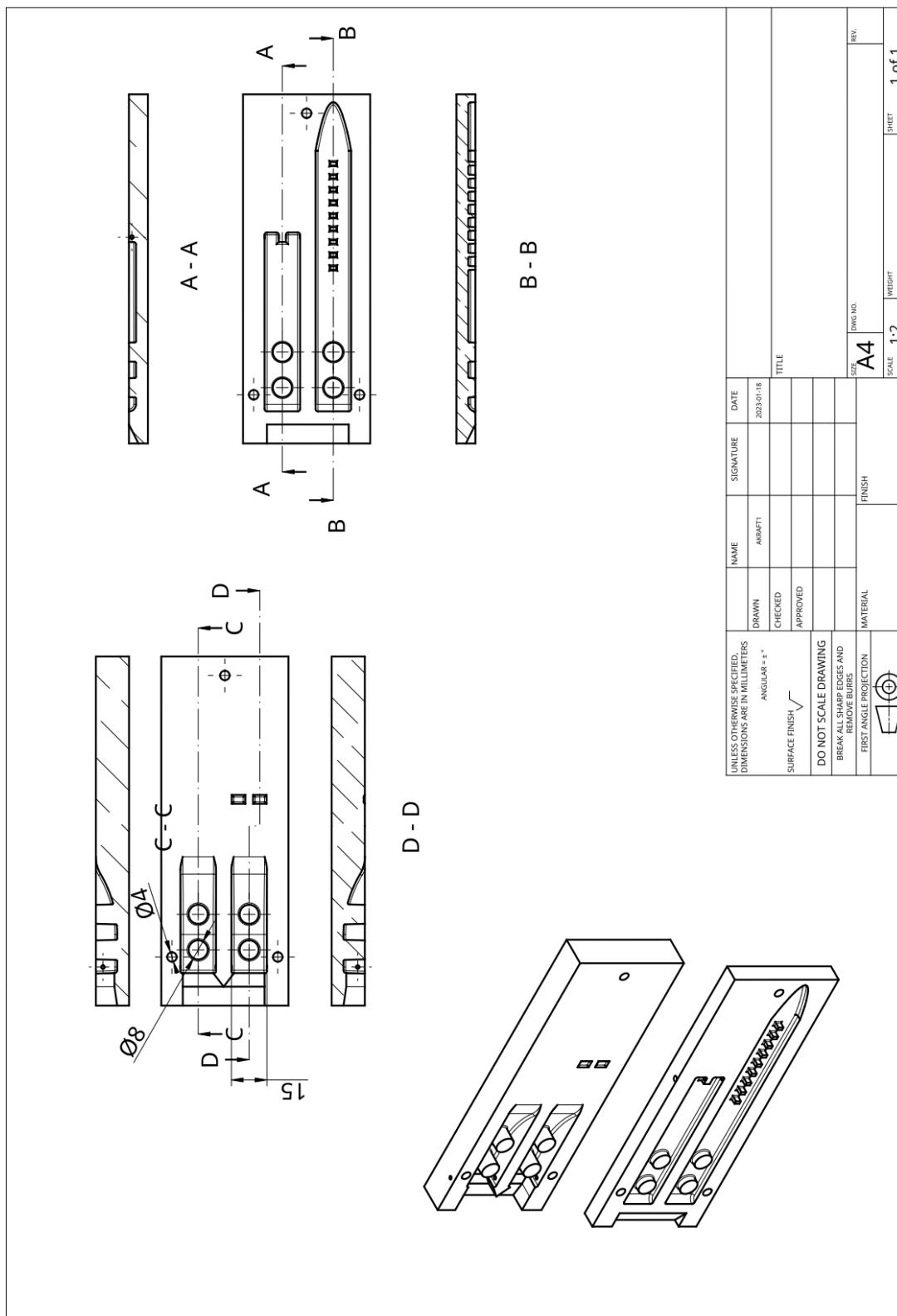


FIGURE 129 – Plan du moule pour la coulée (première version)

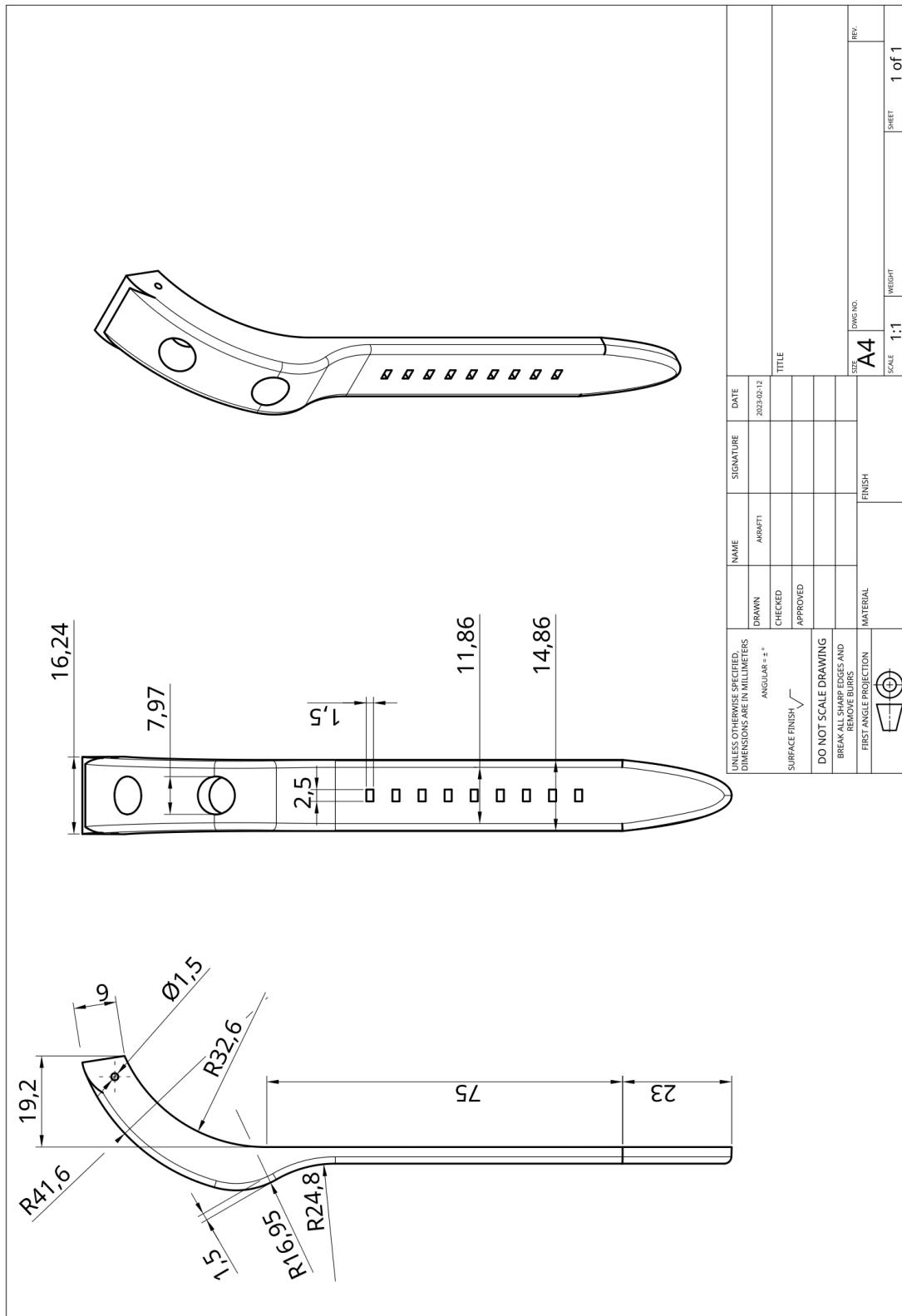


FIGURE 130 – Plan de la deuxième version du bracelet (partie 1)

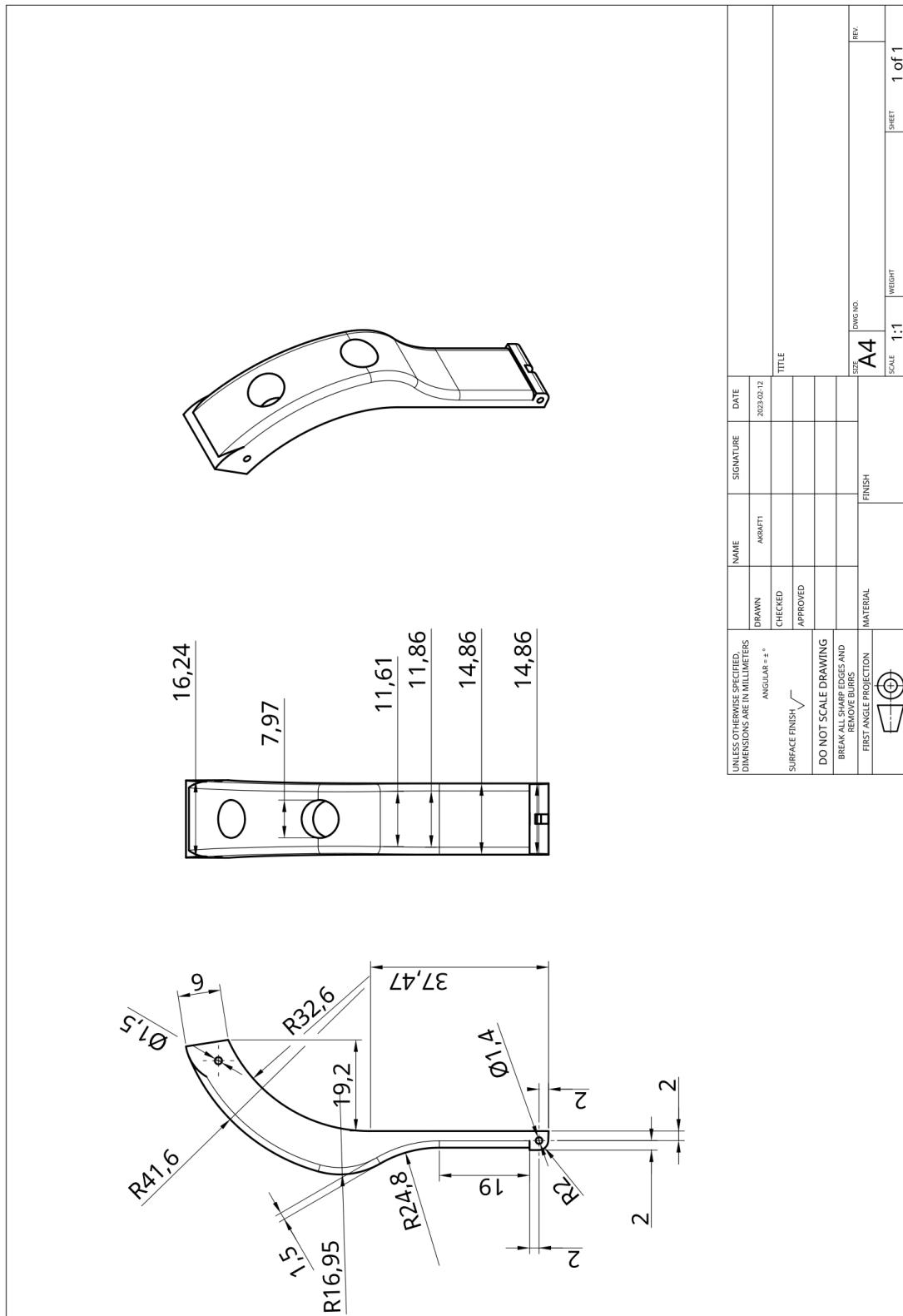


FIGURE 131 – Plan de la deuxième version du bracelet (partie 2)

12.5 Management du projet

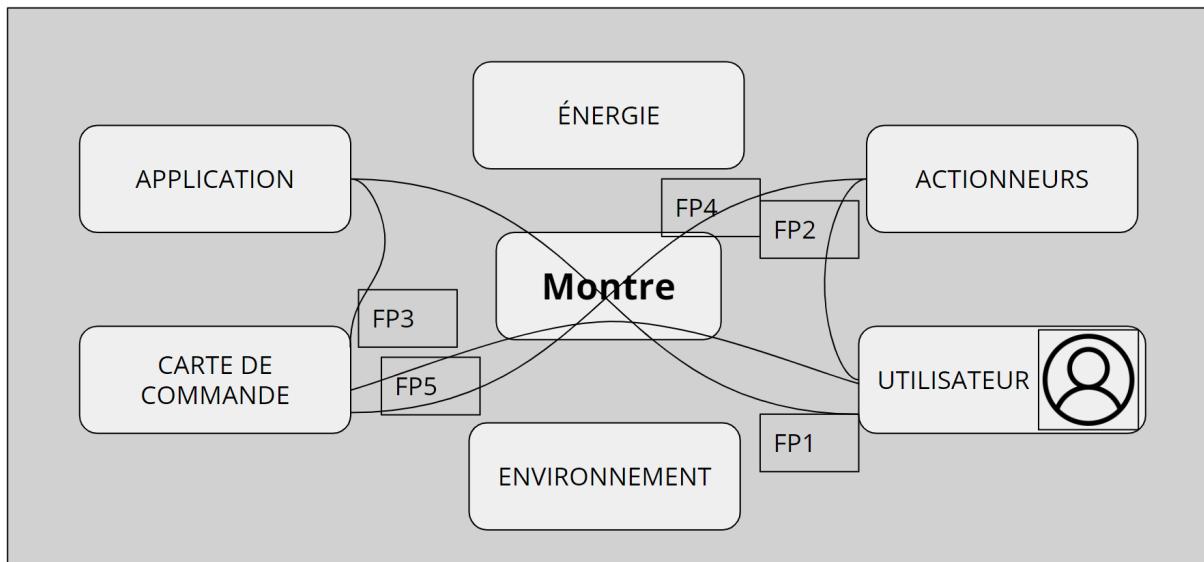


FIGURE 132 – Diagramme pieuvre fonctions principales

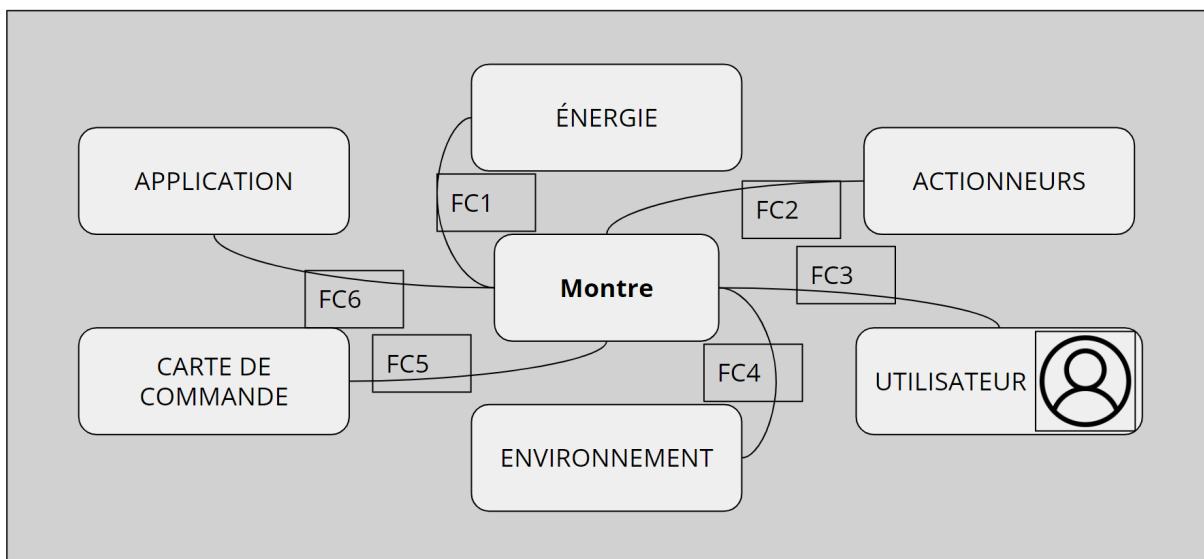


FIGURE 133 – Diagramme pieuvre fonctions contraintes

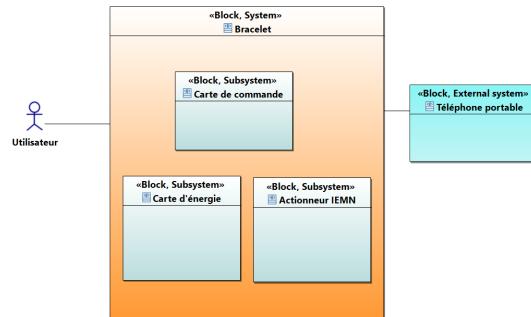


FIGURE 134 – Diagramme de contexte (cliquer ici pour voir en plus grand)

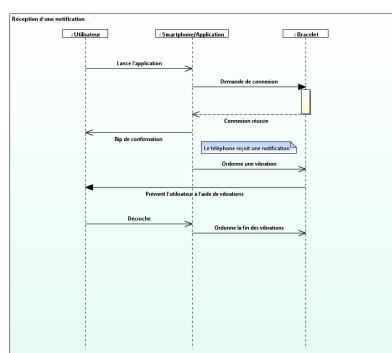


FIGURE 135 – Diagramme de séquence (cliquer ici pour voir en plus grand)

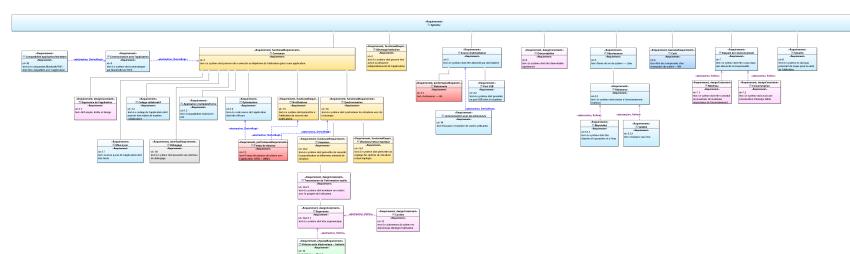


FIGURE 136 – Diagramme d'exigence (cliquer ici pour voir en plus grand)

13 Lexique

- **layout (fichier de)** : Un fichier qui liste l'ensemble des *widgets* d'une page, d'un menu... et décrit généralement leur agencement.
- **UI** : *User Interface* : L'ensemble des éléments visuels auxquels l'utilisateur fait face. L'UI assure l'esthétique du logiciel mais joue aussi un grand rôle dans son ergonomie.
- **widget** : Tout élément avec lequel l'utilisateur peut interagir (bouton, menu déroulant...).
- **BLE** : Bluetooth Low Energy, une norme Bluetooth dont la consommation est au moins deux fois plus faible que le Bluetooth "standard" en échange d'une portée maximale réduite à quelques dizaines de mètres.
- **GATT** : Generic ATtribute, une description/définition d'une connexion entre appareils.
- **Client** : L'appareil qui effectue le scan et donne des ordres. Ici le smartphone.
- **Serveur** : Un appareil qui est détectable en utilisant un scan BLE. Ici la carte électronique. C'est "l'appareil auquel on demande des choses".
- **Caractéristique** : Un type prédéfini de données qui peuvent être échangées par BLE.
- **Service** : Un ensemble de caractéristiques.
- **UUID** : Universally Unique IDentifier, un nombre codé sur 128 bits qui sert à identifier un service ou une caractéristique.
- **Scan** : Le fait de chercher et éventuellement lister les appareils BLE alentour.