



RAPPORT IOT

Membre(s) de l'équipe :

CORRE Alexandre

TABLE DES MATIERES

Table des matieres	2
Introduction	3
Présentation du projet	4
Câblage du prototype	7
Dimensionnement de la batterie	8
Fonctionnement de notre solution	9
a) Partie maintenance et prise d'informations via le pupitre	10
b) Partie visuelle et utilisation de l'interface (processing 4)	13
Première partie de l'interface	14
Deuxième partie de l'interface	15
Troisième partie de l'interface	16
Quatrième partie de l'interface	19
c) Partie traitement des données et interprétations de ces données	22
d) Conclusion	25
Prototype	26
Etude de coûts	27
Réflexions sur le projet	28
Les pistes d'améliorations	29
Conclusion	30
Table des figures	31
Table des tableaux	33
Annexes	34

INTRODUCTION

Notre bureau d'études est sollicité pour concevoir une solution technologique robuste et innovante destinée à optimiser les performances des immobilisations d'avions au sol. Cette solution doit répondre aux besoins opérationnels de deux profils principaux : **le Technicien de maintenance** et **le Superviseur de maintenance**.

L'objectif est de développer un système permettant :

- **Identification et gestion des avions** : Répertorier les aéronefs et les maintenances associées.
- **Suivi et validation des tâches** : Faciliter le suivi, la validation, et la remontée d'information sur les interventions de maintenance.
- **Maintenance prédictive** : Intégrer des outils d'aide à la décision pour la planification des maintenances, y compris prédictives.
- **Résilience face aux aléas** : Assurer une communication efficace même dans des conditions défavorables (faible débit, interruption de connexion, etc.).
- **Maximisation de la plus-value client** : Offrir une solution qui optimise les coûts, les temps d'intervention et la satisfaction client.

Le technicien de maintenance sera un acteur sur le terrain, il a besoin d'un outil intuitif pour consulter les tâches assignées, saisir des informations en temps réel, et signaler les incidents.

Le superviseur de maintenance sera responsable de la coordination et de la supervision des équipes, il doit pouvoir surveiller l'avancée des tâches, valider les étapes clés, et analyser les données pour anticiper les besoins.

Dans le cadre de la maintenance des avions au sol, nous envisageons d'intégrer un système d'analyse avancée des performances des immobilisations. Ce système permettra d'identifier rapidement les éléments critiques à contrôler grâce à une interface intuitive et des outils connectés.

PRESENTATION DU PROJET

1. Description générale de notre projet IOT

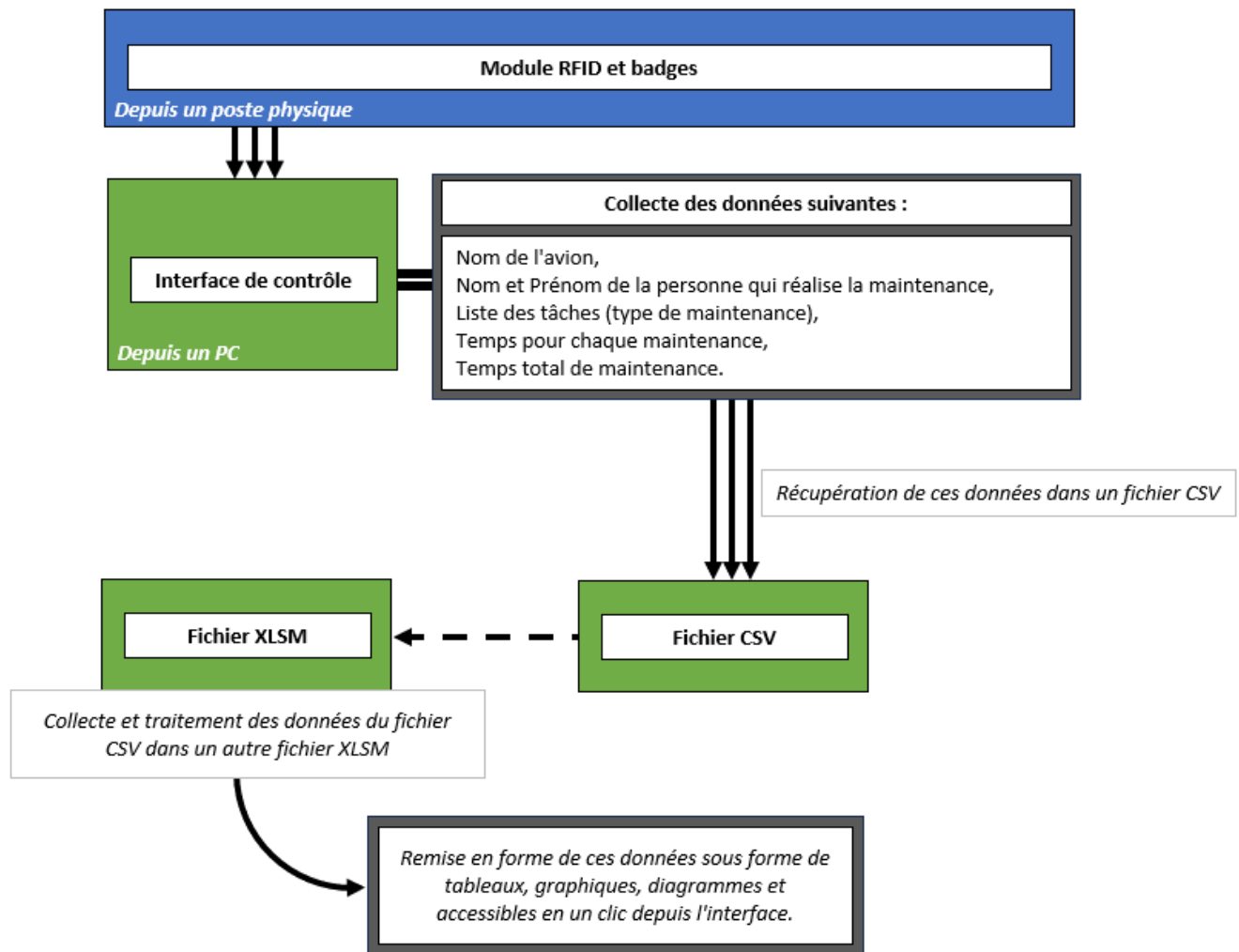


Figure 1 - Description du fonctionnement.

Description détaillée du fonctionnement :

- Une base de données centralisée pour enregistrer les avions, leur état actuel, et leur historique de maintenance.
- Une interface intuitive pour rechercher et identifier les aéronefs rapidement via des identifiants uniques (par exemple : numéro de série, type d'appareil, etc.).
- Un utilisateur est identifié par son UID RFID et associé à un nom.
- Chaque tâche est liée à un tag RFID spécifique et à une LED.
- La LED s'allume pour indiquer qu'une tâche est en cours.
- Le système peut mesurer le temps entre le début et la fin d'une tâche.

2. Matériels qu'on a utilisé durant le projet





Partie électronique		
Nom du composant	Photo du composant	Fonction
Carte Arduino Uno		Elle permet de lire des données de capteurs, de contrôler des dispositifs (moteurs, LED, etc.), et de communiquer avec d'autres systèmes via des ports numériques, analogiques...
Module RFID		Un module RFID permet de lire et/ou écrire des données sur des étiquettes RFID via des ondes radio. Il sert à identifier des objets ou personnes sans contact, à transmettre les données à un système informatique, et est couramment utilisé pour l'authentification ou le contrôle d'accès.
Câbles		Ils permettent de transmettre l'énergie électrique, les signaux ou les données entre différents appareils ou composants.
6 Badges + 2 cartes		Ils permettent d'identifier la tâche en cours de maintenance (badge). Elles permettent d'identifier la personne qui réalise la maintenance (carte).

Tableau 1 - Composants (partie électronique).

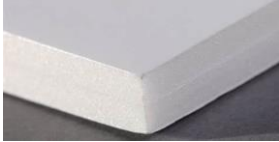

Partie prototype		
Nom du composant	Photo du composant	Fonction
Plaque de carton mousse		Il permet de mettre en forme un petit tableau de contrôle physique qui va venir accueillir l'ensemble des badges et de l'électronique.
Punaises		Elles permettent de relier les badges à la structure, tout en indiquant le lieu où la maintenance sera effectuée.

Tableau 2 - Composants (partie prototype).

Nous avons utilisé les composants précédents de manière à créer un prototype très rapidement et à pouvoir davantage nous concentrer sur l'interface informatique ainsi que sur l'ensemble des fichiers de traitement de données.

Conformément au cahier des charges, nous aurions dû utiliser une carte Arduino Nano.

Cependant, nous avons opté pour une carte Arduino Uno afin de bénéficier de ses ports d'entrées/sorties plus nombreux et de sa meilleure facilité d'accès pour le prototypage. La carte Uno offre également une plus grande stabilité et compatibilité avec le module RFID, tout en restant suffisamment compacte pour être intégrée dans le projet. Ce changement n'affecte pas les performances globales du système et permet d'optimiser le développement en raison de la simplicité d'utilisation de la carte Uno.





Partie informatique		
Nom du logiciel/fichier	Fichier	Fonction
Processing 4		Utilisation de Processing 4 en raison de sa simplicité d'accès et de son environnement de codage intuitif. Ce logiciel permet de créer des applications visuelles rapidement et facilement, ce qui est idéal pour notre projet. Langage de programmation : Java.
Fichier CSV		Il permet de collecter les informations obtenues depuis l'interface Processing 4. Le CSV est simple, léger et facilement compatible avec divers outils. Il permet d'organiser clairement les données, facilitant leur importation dans Excel pour des analyses ou traitements plus complexes.
Fichier XLSM		Il permet de récupérer les données du fichier CSV, de les traiter à l'aide de macros et de formules, et de présenter l'ensemble de ces données sous forme de graphiques, courbes, diagrammes...
Arduino		Il permet de collecter les données reçues par le module RFID avant de les transmettre à l'interface Processing 4. Langage de programmation : C++.

Tableau 3 - Logiciels.

3. Tarification du prototype

Nom du composant	Prix (en €)
Carte Arduino UNO	27.64 €
Module RFID + Badges + Cartes	4.85 €
Câbles	1.76 €
Plaque de carton mousse	11.99 €
Punaises	1.79 €
TOTAL	48.03 €

Tableau 4 - Tarification des composants.

Nous avons donc un prototype d'une valeur de 48.03 €. Nous reviendrons plus tard dans le rapport sur la tarification réelle de ce type de projet.

4. Organisation du travail

Le projet a été réalisé en 7 séances, grâce à une équipe de 7 personnes mobilisées sur différents aspects du développement. Chaque membre a contribué à des domaines spécifiques tels que l'électronique, la simulation 3D, la modélisation 3D, la conception du prototype, le développement de l'interface utilisateur, la programmation, ainsi que d'autres tâches essentielles à la réussite du projet.

CABLAGE DU PROTOTYPE

Pour effectuer le projet, nous avons câblé nos sous-ensembles de la manière suivante :

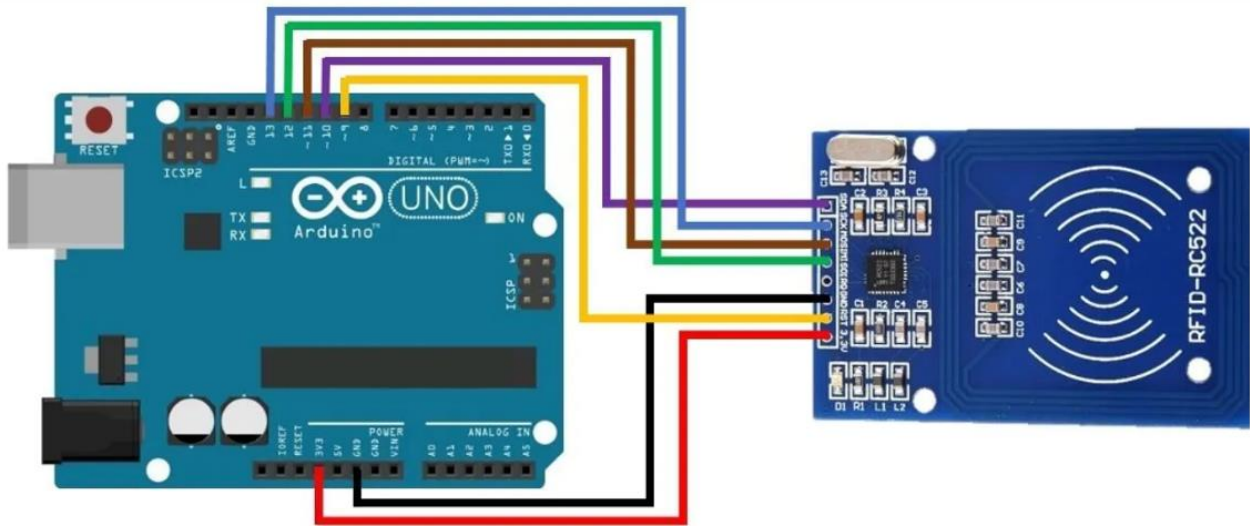


Figure 2 - Câblage du module RFID avec la carte Arduino Uno.

DIMENSIONNEMENT DE LA BATTERIE

Pour réaliser cette partie, nous avons besoin de connaître la consommation de chaque partie du système.

Voici un tableau qui regroupe cela :

Composant	Unité
Carte Arduino	50 mA
Module RFID	30 mA
Tension d'alimentation	5 V

Figure 3 - Consommation électrique des composants.

On calcule la consommation totale des composants de notre système :

$$P = U \times I = 5 \text{ V} \times 0,08 \text{ A} = 0,4 \text{ W} \quad \text{WP} = U \times I = 5 \text{ V} \times 0,08 \text{ A} = 0,4 \text{ W}$$

Nous adressons ensuite la capacité que nous voulons pour notre système :

$$\text{Energie journalière} = 0,4 \text{ W} \times 8\text{h} = 3,2 \text{ Wh}$$

$$\text{Energie hebdomadaire} = 3,2 \text{ Wh} \times 7 = 22,4 \text{ Wh}$$

Avec notre énergie hebdomadaire et notre puissance, nous pouvons calculer la capacité de la batterie. Toujours est-il, nous voulons appliquer un DoD (Depth of Discharge), qui représente la capacité utilisable de la batterie, fixé à 80 %.

Soit :

$$\frac{\text{DoD}}{\text{Energie nécessaire}} = \frac{22,4}{0,8} = 28 \text{ Wh}$$

Enfin, nous pouvons calculer la capacité de la batterie selon nos critères.

$$\text{Capacité (Ah)} = \frac{\text{Capacité totale (Wh)}}{\text{Tension (V)}} = \frac{28}{5} = 5,6 \text{ Ah}$$

On peut ici arrondir notre capacité à 6 Ah et ainsi établir notre marge de sécurité de 0.4 Ah.

Avec ce dimensionnement, nous pouvons prétendre à utiliser une batterie LiPo 5V d'une capacité de 6Ah et ainsi assurer une autonomie d'une semaine en fonctionnement 8H/jour.

FONCTIONNEMENT DE NOTRE SOLUTION

1. Profils des utilisateurs

Concernant notre projet, deux profils de personne vont utiliser notre solution (interface et plateforme physique). Voici leur profil :

Technicien. *Dans notre cas : Alex Martin.*

Son rôle :

Il est directement responsable de l'exécution des opérations de maintenance sur les aéronefs, il va utiliser l'interface de manière à retranscrire et garder les temps de maintenances passés sur les aéronefs.

Responsable. *Dans notre cas : Sophie Dupont.*

Son rôle :

Elle assure la coordination et la gestion stratégique des activités de maintenances pour minimiser les temps d'immobilisation des aéronefs et optimiser les ressources. Pour ce faire, elle aura accès au niveau de l'interface à un bouton lui permettant d'accéder à une page dans laquelle des données sous forme de graphiques, tableaux, diagrammes y seront référencées.

Nous avons donc deux types d'utilisations possibles : d'un côté, une personne qui effectue la maintenance, donc la partie manuelle (profil de technicien), et de l'autre, une personne qui cherche à optimiser les maintenances en traitant les informations des maintenances précédentes pour prédire ou prévoir les prochaines maintenances (maintenances préventives), le tout de manière simplifiée (profil de responsable).

Pour répondre à la demande de ces deux types de profil, nous allons revenir en détail sur la solution que nous avons abordée dans le chapitre « 1. Description générale de notre projet IOT ».

2. Fonctionnement de notre solution

Notre solution se décompose en 3 grandes parties, ces parties sont les suivantes :

- La partie maintenance et prise d'information via le pupitre.
- La partie visuelle et utilisation de l'interface (processing 4).
- La partie traitement des données et l'interprétation de ces données.

Dans les sous-parties suivantes, nous aborderons en détail l'ensemble des 3 points précédents.

a) Partie maintenance et prise d'informations via le pupitre

Dans cette partie, le schéma de fonctionnement est le suivant :

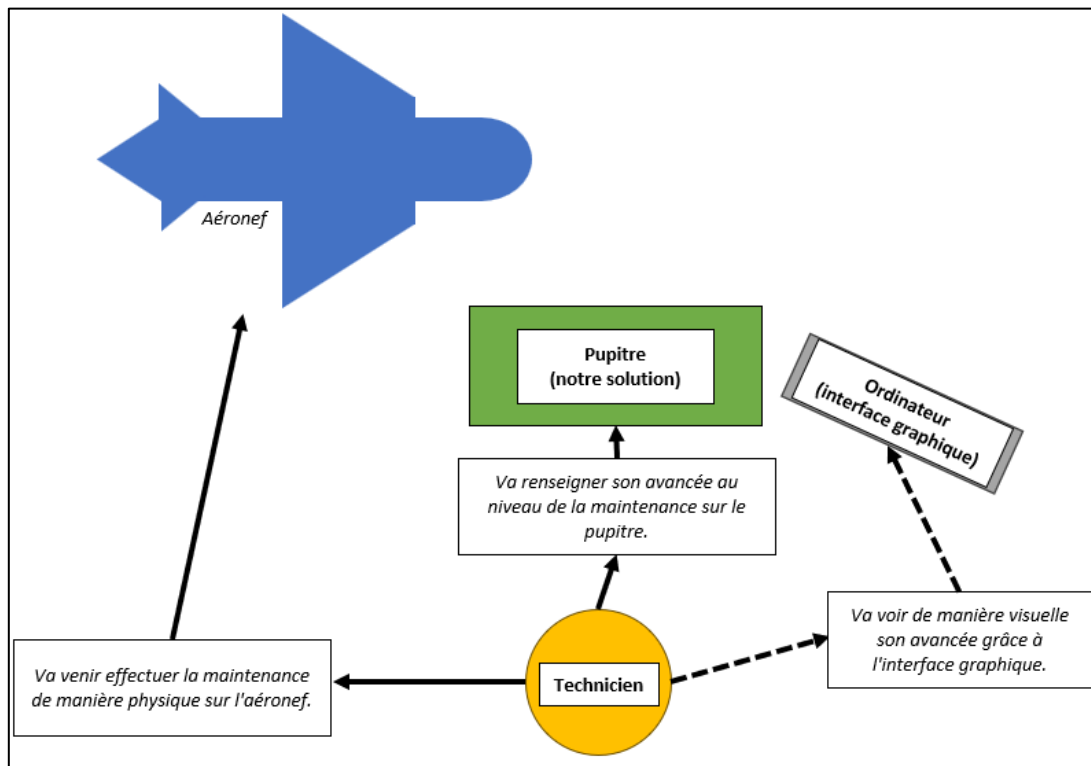


Figure 4 - Schéma de fonctionnement (Partie maintenance et prise d'informations).

Le technicien va avoir à sa disposition le matériel suivant :

- Une carte d'identification contenant son nom, prénom et un ID unique (IUD) de manière à pouvoir l'identifier avec le module RFID.
- Un pupitre équipé de plusieurs badges de contrôle (dans notre cas, 6 badges).
- Les outils nécessaires pour effectuer la maintenance.

Procédure d'utilisation :

- 1 Le technicien va scanner sa carte d'identification au niveau du module RFID situé sur le pupitre.
- 2 Le technicien va sélectionner le nom de l'avion dans l'interface graphique (plus en détail dans la prochaine partie).
- 3 Le technicien va sélectionner un badge attribué en fonction du type de maintenance qu'il va effectuer sur l'aéronef.
- 4 Le technicien va scanner ce badge attribué (1 fois) et va ensuite effectuer la maintenance sur l'aéronef.
- 5 Lorsque la maintenance est terminée, le technicien va scanner à nouveau le badge attribué de manière à indiquer au logiciel qu'il a terminé la maintenance.
- 6 Le technicien va effectuer les deux dernières étapes jusqu'à avoir terminé l'ensemble des maintenances sur l'aéronef.

D'autres étapes sont nécessaires par la suite, mais nous les aborderons dans « La partie visuelle et utilisation de l'interface (processing 4) ».

Fonctionnement au niveau du logiciel (pour la partie pupitre) :

Dans cette section dédiée au pupitre, nous utiliserons principalement du code Arduino. Voici le code détaillé étape par étape afin de suivre les 6 étapes énoncées précédemment :

```

1  #include <SPI.h>
2  #include <MFRC522.h>
3  #define SS_PIN 10
4  #define RST_PIN 9
5  #define LED_PIN 7 // Pin de la LED physique
6
7  MFRC522 mfrc522(SS_PIN, RST_PIN);
8
9  // Structure pour associer UID et noms
10 struct User {
11     byte uid[4];
12     char name[16];
13 };
14
15 // Liste des utilisateurs autorisés (UID principaux)
16 User users[] = {
17     {{0xBB, 0x4E, 0xC6, 0x50}, "Alex:MARTIN"}, // Format : "Prénom:Nom"
18     {{0x64, 0xF2, 0x36, 0x5B}, "Julie:RESP"},
19 };
20
21
22 const int numUsers = sizeof(users) / sizeof(users[0]);
23
24 // Liste des badges secondaires avec leurs tâches spécifiques
25 struct Task {
26     byte uid[4];
27     char taskName[20]; // Tâche associée
28 };
29
30 // Référence des badges secondaires (UID secondaires)
31 Task secondaryTasks[] = {
32     {{0x21, 0xFC, 0x66, 0x1E}, "Avionniques"},
33     {{0x41, 0x86, 0x67, 0x1E}, "APU"},
34     {{0x39, 0x96, 0xAB, 0xE2}, "Moteurs"},
35     {{0x1C, 0x69, 0xB5, 0x6D}, "Train atterrissages"},
36     {{0xD1, 0x07, 0x67, 0x1E}, "Reservoirs"},
37     {{0xB1, 0xF6, 0x66, 0x1E}, "Ailes"},
38 };
39
40 const int numTasks = sizeof(secondaryTasks) / sizeof(secondaryTasks[0]);
41

```

Inclut la bibliothèque SPI nécessaire pour communiquer avec le module RFID.

Inclut la bibliothèque spécifique pour le module RFID RC522, permettant de lire les cartes RFID.

Pour interagir avec le module RFID en utilisant les broches définies précédemment.

Format : « Prénom:Nom » grâce au char name [16] précédent.

Liste des utilisateurs autorisés avec leur UID unique et nom.

Calcul automatiquement le nombre d'utilisateurs dans la liste **users**.

Définit une structure **Task** pour associer un UID RFID à une tâche spécifique.

Liste des tâches secondaires avec les UID correspondants.

Calcule le nombre de tâches secondaires.

Figure 5 - Code Arduino (Partie pupitre).

Voici le code, accompagné du rôle de chaque sous-partie en référence à la procédure d'utilisation :

```

1  #include <SPI.h>
2  #include <MFRC522.h>
3  #define SS_PIN 10
4  #define RST_PIN 9
5  #define LED_PIN 7 // Pin de la LED physique
6
7  MFRC522 mfrc522(SS_PIN, RST_PIN);
8
9  // Structure pour associer UID et noms
10 struct User {
11     byte uid[4];
12     char name[16];
13 };
14
15 // Liste des utilisateurs autorisés (UID principaux)
16 User users[] = {
17     {{0xBB, 0x4E, 0xC6, 0x50}, "Alex:MARTIN"}, // Format : "Prénom:Nom"
18     {{0x64, 0xF2, 0x36, 0x5B}, "Julie:RESP"},
19 };
20
21
22 const int numUsers = sizeof(users) / sizeof(users[0]);
23
24 // Liste des badges secondaires avec leurs tâches spécifiques
25 struct Task {
26     byte uid[4];
27     char taskName[20]; // Tâche associée
28 };
29
30 // Référence des badges secondaires (UID secondaires)
31 Task secondaryTasks[] = {
32     {{0x21, 0xFC, 0x66, 0x1E}, "Avionniques"},
33     {{0x41, 0x86, 0x67, 0x1E}, "APU"},
34     {{0x39, 0x96, 0xAB, 0xE2}, "Moteurs"},
35     {{0x1C, 0x69, 0xB5, 0x6D}, "Train atterrissages"},
36     {{0xD1, 0x07, 0x67, 0x1E}, "Reservoirs"},
37     {{0xB1, 0xF6, 0x66, 0x1E}, "Ailes"},
38 };
39
40 const int numTasks = sizeof(secondaryTasks) / sizeof(secondaryTasks[0]);
41

```

1

3

Figure 6 - Code Arduino (Partie pupitre). (2)

Voici la suite du code Arduino :

```

41 void setup() {
42   Serial.begin(9600);
43   SPI.begin();
44   mfrc522.PCD_Init();
45   pinMode(LED_PIN, OUTPUT);
46   digitalWrite(LED_PIN, LOW);
47   Serial.println("Approche ta carte...");
48 }
49 
```

Configure la communication série pour afficher des messages dans le Sérial Monitor Arduino.

Initialise la communication SPI.

Initialise le module RFID.

Initialise le module RFID.

Affiche un message.

```

49 void loop() {
50   // Attente de la détection d'une nouvelle carte
51   if (!mfrc522.PICC_IsNewCardPresent() || !mfrc522.PICC_ReadCardSerial()) {
52     return;
53   }
54 }
55
56 byte* uid = mfrc522.uid.uidByte;
57 bool userFound = false;
58
59 // Vérifie si l'UID correspond à un utilisateur enregistré
60 for (int i = 0; i < numUsers; i++) {
61   if (compareUID(uid, users[i].uid)) {
62     Serial.print("USER_");
63     Serial.println(users[i].name); // Envoie "USER_Prenom:Nom" à Processing
64     digitalWrite(LED_PIN, HIGH); // Allume la LED physique pour succès
65     delay(2000); // Maintient la LED allumée pendant 2 secondes
66     digitalWrite(LED_PIN, LOW);
67     userFound = true;
68     break;
69   }
70 }
71
72 // Vérifie si l'UID correspond à un badge secondaire pour les tâches
73 if (!userFound) {
74   for (int i = 0; i < numTasks; i++) {
75     if (compareUID(uid, secondaryTasks[i].uid)) {
76       Serial.print("TASK_");
77       Serial.println(secondaryTasks[i].taskName); // Envoie le nom de la tâche à Processing
78       digitalWrite(LED_PIN, HIGH); // Allume la LED pour indiquer une tâche secondaire
79       delay(2000);
80       digitalWrite(LED_PIN, LOW);
81       userFound = true;
82       break;
83     }
84   }
85 }
86
87 // Si aucun utilisateur ou tâche n'a été trouvée avec cet UID
88 if (!userFound) {
89   Serial.println("UNKNOWN_CARD"); // Envoie le message pour carte inconnue
90   digitalWrite(LED_PIN, LOW); // Assure que la LED reste éteinte
91 }
92
93 mfrc522.PICC_HaltA(); // Arrête la communication avec la carte
94 }
95 
```

Vérifie si une carte RFID est détectée :
 PICC_IsNewCardPresent() : Vérifie s'il y a une nouvelle carte.
 PICC_ReadCardSerial() : Lit l'UID de la carte si elle est détectée.
 Si aucune carte n'est détectée, la boucle se termine.

Récupère l'UID lu dans une variable **uid**.
 Initialise un drapeau **userFound** pour indiquer si une correspondance est trouvée.

Cette fonction va parcourir la liste des utilisateurs de la ligne 16.
 Si l'UID correspond à un utilisateur de cette liste, son nom est envoyé sur le port série avec le préfixe **USER_**.
 La led s'allume pendant deux secondes pour indiquer la réussite du scanne.
 La boucle est interrompue avec **break**.

Si l'UID n'est pas associé à un utilisateur principal, il est vérifié dans la liste des tâches secondaires :
 Si une correspondance est trouvée, le nom de la tâche est envoyé avec le préfixe **TASK_**, et la led s'allume.

Si l'UID ne correspond à aucun utilisateur ou tâche, un message **UNKNOWN_CARD** est envoyé, et la led reste éteinte.

Met fin à la communication avec la carte détectée.

Figure 7 - Code Arduino (Partie pupitre). (3)

```

96 // Fonction pour comparer deux UID
97 bool compareUID(byte* uid1, byte* uid2) {
98   for (byte i = 0; i < 4; i++) {
99     if (uid1[i] != uid2[i]) return false;
100   }
101   return true;
102 }
103 
```

Compare deux UID octet par octet.
 Retourne **true** si les UID sont identiques, sinon **false**.

Figure 8 - Code Arduino (Partie pupitre). (4)

b) Partie visuelle et utilisation de l'interface (processing 4)

Dans cette partie, le schéma de fonctionnement est le suivant :

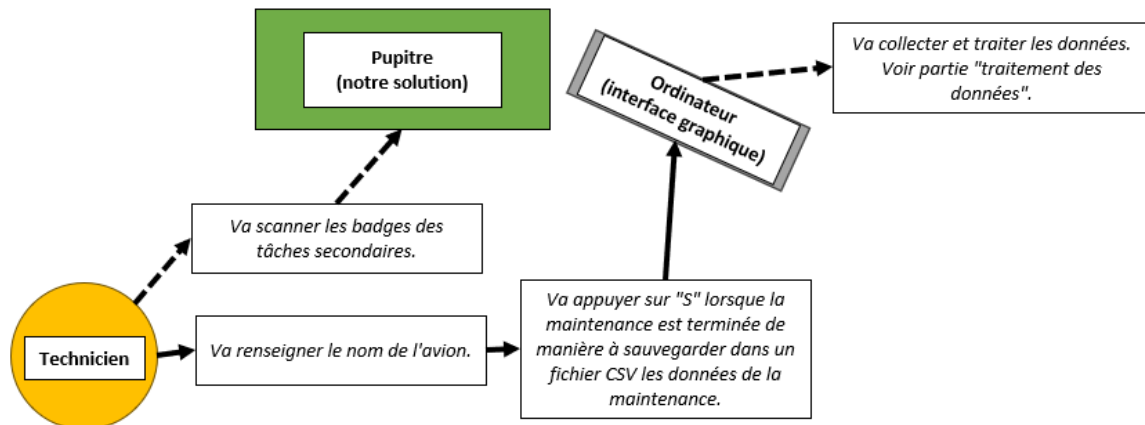


Figure 9 - Schéma de fonctionnement (Partie visuelle et utilisation de l'interface (processing 4)).

Le technicien va avoir à sa disposition le matériel suivant :

- Le pupitre + badges + cartes (matériels de la partie précédente).
- Une interface (processing 4) située sur son ordinateur.

Procédure d'utilisation :

- 1 Le technicien va sélectionner le nom de l'avion via une liste déroulante située dans l'interface graphique.
- 2 Le technicien va effectuer en amont l'ensemble de la maintenance sur l'aéronef comme énoncé dans la sous-partie précédente.
- 3 Le technicien va venir enregistrer l'ensemble des données relatives à la maintenance effectuée dans un fichier CSV en appuyant sur « S » depuis l'interface.

Fonctionnement au niveau du logiciel (pour la partie interface (via processing 4)) :

Dans cette section dédiée à l'interface, nous utiliserons principalement du code processing (donc du Java). Le code étant long et très détaillé (car beaucoup d'esthétique pour pouvoir dessiner un avion et avoir quelque chose de très visuel), nous allons montrer une image de l'interface et aborderons le code utilisé pour chaque partie de l'interface (en termes de fonctionnement, vous retrouverez la partie design de l'avion et mise en forme de l'interface en Annexe).

Voici une image de l'interface créée avec processing 4 :

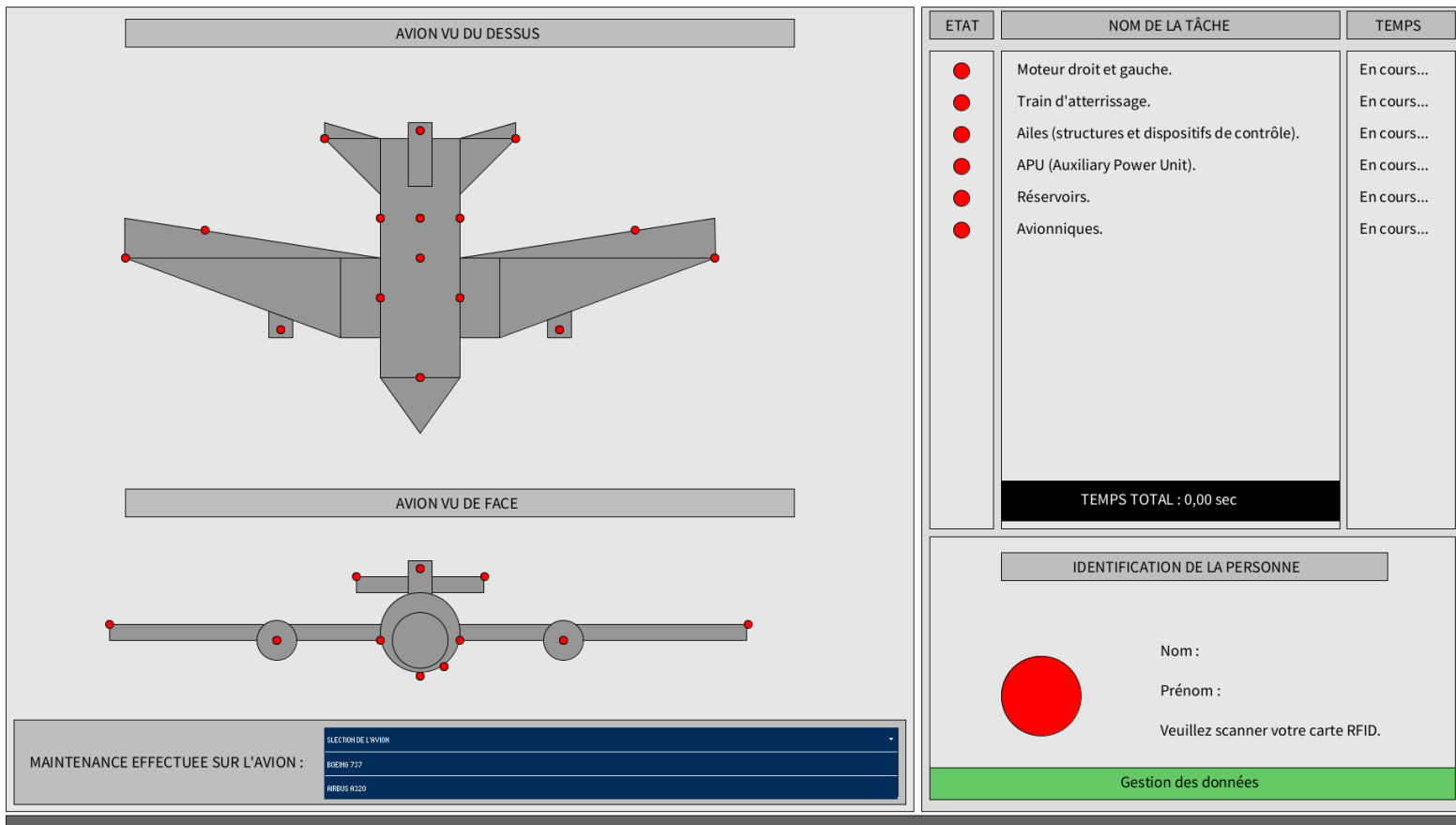


Figure 10 - Interface processing 4.

Cette interface se divise en 4 parties, les voici :

Première partie de l'interface

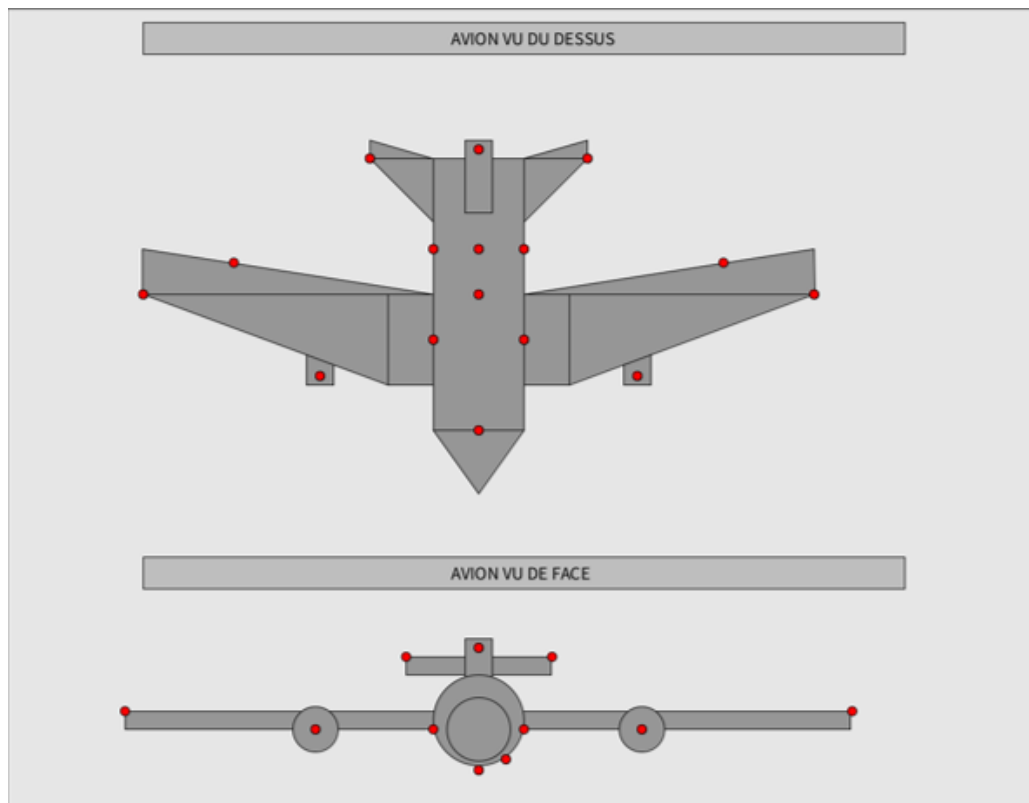


Figure 11 - Partie 1 de l'interface processing 4.

Cette partie est purement esthétique, nous y retrouvons une vue de l'avion du dessus et une vue de l'avion de face. De plus, nous y retrouvons des points rouges qui sont en réalité des leds et celles-ci vont changer de couleur en fonction de l'avancée de la maintenance.

Cette configuration va permettre au technicien d'identifier l'endroit où la maintenance doit être effectuée.

Vous pouvez retrouver la partie du code qui gère cette mise en forme en Annexe.

Deuxième partie de l'interface

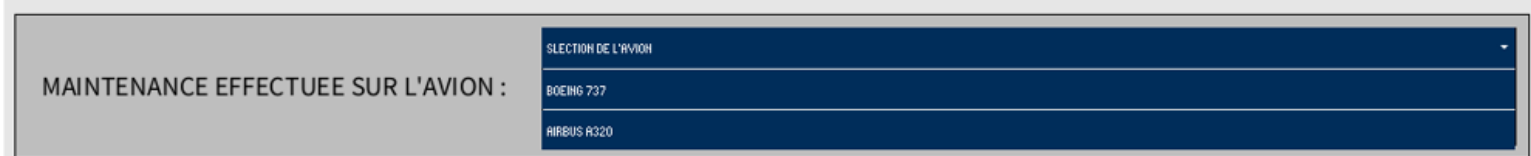


Figure 12 - Partie 2 de l'interface processing 4.

Cette partie va permettre au technicien de sélectionner l'avion sur lequel va être effectuée la maintenance. Pour cela, un grand nombre de nom d'avion a été référencé dans le code.

Voici le code qui permet de mettre en forme la liste déroulante :

```

1 import processing.serial.*; //Importation de la bibliothèque pour la communication série avec Arduino.
2 import controlP5.*; /* Importation de la bibliothèque controlP5, utilisée pour créer des interfaces graphiques
3 comme des listes déroulantes. */
4
5 String selectedPlane = ""; // Variable qui stockera le nom de l'avion sélectionné dans la liste déroulante.
6 ControlP5 cp5; // Déclaration d'un objet ControlP5 pour gérer les éléments d'interface graphique.
7 DropDownList avionList; // Liste déroulante pour afficher les noms d'avions.
8 String[] avions = {"Boeing 737", "Airbus A320", "Concorde", "Cessna 172",
9 "Airbus A220-300", "Airbus A220-100", "Boeing 777", "Boeing 747"}; // Tableau contenant les noms d'avions.
10
11 Serial myPort; // Objet pour la communication série
12
13 void setup() {
14     fullScreen(); // Affiche l'interface en grand écran.
15     textSize(20); // Définit la taille par défaut du texte affiché (nbr pxl).
16     cp5 = new ControlP5(this); // Initialise l'objet cp5 pour gérer les composants de l'interface utilisateur.
17
18     // Création de la liste déroulante :
19     avionList = cp5.addDropDownList("Sélection de l'avion") // Crée une liste déroulante avec le titre "Sélection de l'avion".
20         .setPosition(450, 940) // Positionne la liste déroulante sur l'écran. (x, y)
21         .setSize(720, 100) // Définit les dimensions de la liste déroulante (x, y) pxl.
22         .addItem(avions) // Ajoute les noms d'avions définis dans le tableau avions comme options de la liste.
23         .setValue(0) // Sélectionne l'élément par défaut
24         .plugTo(this, "avionSelect"); /* Connecte la liste déroulante à une méthode qui
25 sera appelée lorsqu'un élément est sélectionné. */
26
27     // Style de la liste déroulante :
28     avionList.setItemHeight(30); //Définit la hauteur de chaque élément dans la liste déroulante à 30 pixels.
29     avionList.setBarHeight(30); // Définit la hauteur de la barre visible lorsque la liste n'est pas déroulée à 30 pixels.
30 }
31
32 // Méthode appelée lorsqu'un avion est sélectionné
33 public void avionSelect(int n) { // Méthode déclenchée lorsqu'un avion est sélectionné dans la liste déroulante.
34     selectedPlane = avions[n]; // Met à jour la variable selectedPlane avec le nom de l'avion correspondant à l'index n.
35     println("Avion sélectionné : " + selectedPlane); // Affiche le nom de l'avion sélectionné dans la console processing.
36 }

```

Figure 13 - Code processing pour la liste déroulante.

Vous pouvez retrouver en commentaires l'explication de chaque ligne de code. Il manque toutefois les lignes de code liées à la mise en forme que l'on peut voir dans la figure précédente (à savoir le rectangle en fond et le texte « MAINTENANCE EFFECTUEE SUR L'AVION : ». Vous retrouverez les lignes concernant cette mise en forme en Annexe ou dans le code directement.

Troisième partie de l'interface

ETAT	NOM DE LA TÂCHE	TEMPS
●	Moteur droit et gauche.	En cours...
●	Train d'atterrissage.	En cours...
●	Ailes (structures et dispositifs de contrôle).	En cours...
●	APU (Auxiliary Power Unit).	En cours...
●	Réservoirs.	En cours...
●	Avionniques.	En cours...
TEMPS TOTAL : 0,00 sec		

Figure 14 - Partie 3 de l'interface processing 4.

Ci-dessous, vous trouverez le code pour obtenir le visuel de la partie précédente (à gauche) ainsi que le rendu (à droite) :

```

1 void setup() {
2   fullscreen(); // pour avoir taille écran max peut importe l'ordinateur
3   textSize(20); //va donner à l'ensemble de l'interface la taille de texte indiquée
4 }
5
6 void draw() {
7   background(255); // Fond blanc
8   //Code pour les zones de texte et le visuel de l'interface :
9   fill(220); // couleur du rect ci-dessous
10  rect(1200, 35, 680, 1010); // rectangle
11  fill(190);
12  rect(1210, 40, 80, 35);
13  fill(0); // couleur du texte ci-dessous
14  text("ETAT", 1230, 65); //zone de texte
15  fill(190);
16  rect(1300, 40, 425, 35);
17  fill(0);
18  text("NOM DE LA TÂCHE", 1435, 65);
19  fill(190);
20  rect(1734, 40, 136, 35);
21  fill(0);
22  text("TEMPS", 1770, 65);
23  fill(230);
24  rect(1210, 90, 80, 600);
25  fill(230);
26  rect(1300, 90, 425, 600);
27  fill(230);
28  rect(1734, 90, 136, 600);
29  fill(230);
30  rect(1210, 700, 660, 330);
31  fill(190);
32  rect(1300, 720, 485, 35);
33  fill(0);
34  text("IDENTIFICATION DE LA PERSONNE", 1390, 745);
35 }

```

Figure 15 - Code de la partie visuelle.

ETAT	NOM DE LA TÂCHE	TEMPS
IDENTIFICATION DE LA PERSONNE		

Figure 16 - Résultat du visuel.

Ci-dessous, vous trouverez la suite du code pour la mise en forme des tâches, la mise en place d'un chronomètre mais aussi la mise en place d'un système de led capable de changer de couleur en fonction de l'avancée de la maintenance. Il existe trois états, les voici :

- **Etat rouge** : Etat initial, la led n'a encore reçu aucune information, elle reste rouge.
- **Etat orange** : La led devient orange dès que le technicien va scanner un badge pour lancer une maintenance. De plus, un chronomètre va se lancer de manière à connaître la durée de la maintenance.
- **Etat vert** : La led devient verte dès que le technicien va scanner une deuxième fois un badge pour lancer une maintenance. Cette action va aussi mettre fin au chronométrage et celui-ci sera indiqué à droite de la liste des tâches.


```

10 int engineState = 0; // état du moteur principal (0=rouge, 1=orange, 2=vert)
11
12 Serial myPort; // Objet pour la communication série
13
14 // Tableau d'états pour les tâches secondaires (0 = rouge, 1 = orange, 2 = vert)
15 int[] taskStates = {0, 0, 0, 0, 0, 0}; // Tableau représentant les états de chaque tâche (rouge, orange ou vert donc 0, 1, 2)
16 float[] taskStartTimes = new float[6]; // Tableau pour stocker le moment où la tâche devient orange
17 float[] taskTimes = new float[6]; // Tableau pour stocker le temps écoulé pour chaque tâche
18
19 void setup() {
20   fullscreen(); // pour avoir taille écran max peut importe l'ordinateur
21   myPort = new Serial(this, "COM3", 9600); /* Initialise la communication série via le port COM3
22   avec une vitesse de 9600 bauds */
23   myPort.bufferUntil('\n'); // Lecture ligne complète depuis Arduino
24   textSize(20); //va donner à l'ensemble de l'interface la taille de texte indiquée
25 }

```

Figure 17 - Code pour la liste des tâches + Couleur led + chronomètre (partie 1).

```

27 void draw() {
28   background(255); // Fond blanc
29   drawLEDs(); // Appelle une fonction pour dessiner les led du moteur gauche
30   // Liste des tâches à afficher :
31   String[] phrases = {
32     "Moteur droit et gauche.",
33     "Train d'atterrissage.",
34     "Ailes (structures et dispositifs de contrôle).",
35     "APU (Auxiliary Power Unit).",
36     "Réservoirs.",
37     "Avioniques."
38   };
39
40   float totalTime = 0.0;
41   for (int i = 0; i < taskTimes.length; i++) {
42     if (taskStates[i] == 2) { // Va seulement comptabiliser les tâches terminées
43       totalTime += taskTimes[i]; // Additionne les temps
44     }
45   }
46
47   // Temps total de maintenance (Rectangle noir à droite de l'interface) :
48   fill(0);
49   rect(1300, 630, 425, 50);
50   fill(255);
51   text("TEMPS TOTAL : " + nf(totalTime, 1, 2) + " sec", 1400, 660);
52
53   // Affichage de la liste avec LED à côté et les chronomètres
54   for (int i = 0; i < phrases.length; i++) {
55     fill(0);
56     text(phrases[i], 1320, 120 + (i * 40)); //pour changer emplacement du texte des tâches secondaires
57
58     // Affichage du temps de chaque tâche une fois la LED verte
59     if (taskStates[i] == 2) { // Si la led est verte (tâche terminée)
60       text("Temps: " + nf(taskTimes[i], 0, 2) + " s", 1750, 120 + (i * 40)); // Chrono de la tâche (deux chiffres après virgule)
61     } else {
62       text("En cours...", 1750, 120 + (i * 40)); // Affichage si tâche est en cours et non terminée
63     }
64   }
65 }

```

Figure 18 - Code pour la liste des tâches + Couleur led + chronomètre (partie 2).

```

65 // Détermine la couleur de la LED - en fonction état actuel
66 if (taskStates[i] == 0) {
67   fill(255, 0, 0); // Rouge
68 } else if (taskStates[i] == 1) {
69   fill(255, 165, 0); // Orange
70 } else if (taskStates[i] == 2) {
71   fill(0, 255, 0); // Vert
72 }
73 ellipse(1250, 115 + (i * 40), 20, 20); // Dessin de la led
74 }
75 }
76
77 void drawLEDs() {
78   // Code pour le changement de led en fonction de l'état du badge :
79   if (taskStates[0] == 2) { // Si la tâche "Moteur" est terminée (led verte)
80     engineState = 2; // Moteur led devient vert
81   } else if (taskStates[0] == 1) { // Si la tâche est en cours (led orange)
82     engineState = 1; // Moteur led devient orange
83   } else {
84     engineState = 0; // Si la tâche est non commencée (led rouge)
85   }
86   // LED Moteur gauche
87   fill(engineState == 0 ? color(255, 0, 0) : (engineState == 1 ? color(255, 165, 0) : color(0, 255, 0)));
88   // Ligne du dessus : donne la couleur à la led en fonction de l'état
89   ellipse(390, 830, 10, 10); // Position de la led sur Moteur gauche en VDF
90   ellipse(395, 440, 10, 10); // Position de la led sur Moteur gauche en VDD
91 }

```

Figure 19 - Code pour la liste des tâches + Couleur led + chronomètre (partie 3).

```

94 // Fonction appelée quand une ligne complète est reçue depuis Arduino :
95 void serialEvent(Serial myPort) {
96     String message = myPort.readStringUntil('\n'); // Lecture jusqu'à la fin de la ligne
97     if (message != null) {
98         message = message.trim(); // Retirer les espaces inutiles
99         println("Message reçu: " + message); // Affiche chaque message reçu pour le débogage
100
101         // Gestion des tâches secondaires
102         if (message.startsWith("TASK_")) {
103             // Recherche de la tâche et mise à jour de son état
104             String taskName = message.substring(5); // Ignore "TASK_"
105             for (int i = 0; i < taskStates.length; i++) {
106
107                 if (taskName.equals(getTaskName(i))) {
108                     // Si la tâche est orange, passe à vert et calcule le temps écoulé
109                     if (taskStates[i] == 0) { // Si la LED est rouge (étape non commencée)
110                         taskStates[i] = 1; // Passe à l'état orange (tâche en cours)
111                         taskStartTimes[i] = millis() / 1000.0; // Démarre le chrono à l'instant actuel
112                     } else if (taskStates[i] == 1) { // Si la LED est orange (tâche en cours)
113                         taskStates[i] = 2; // Passe à l'état vert (tâche terminée)
114                         taskTimes[i] = (millis() / 1000.0) - taskStartTimes[i]; // Calcule le temps écoulé pour la tâche
115                     }
116                 }
117             }
118         }
119     }
120 }
121
122 // Fonction pour obtenir le nom de la tâche en fonction de l'index pour validation des messages
123 String getTaskName(int index) {
124     String[] tasks = {
125         "Avionniques",
126         "APU",
127         "Moteurs",
128         "Train atterrissages",
129         "Réservoirs",
130         "Ailes"
131     };
132     return tasks[index];
133 }

```

Figure 20 - Code pour la liste des tâches + Couleur led + chronomètre (partie 4).

En ajoutant l'ensemble du code des 4 dernières figures que l'on retrouve ci-dessus, nous obtenons la mise en forme suivante :



Figure 21 - Résultat de l'ajout du code précédent.

Le code va donc nous permettre d'afficher un système de led (au niveau du schéma de l'avion et au niveau de la liste des tâches) qui va changer de couleur en fonction de l'état de la maintenance, mais aussi la liste des tâches à effectuer sur l'aéronef et pour finir, le chronométrage de chacune des tâches ainsi que le temps total pour l'ensemble de la maintenance.

Concernant cette partie du code :

```
// Code pour le changement de led en fonction de l'état du badge :
if (taskStates[0] == 2) { // Si la tâche "Moteur" est terminée (led verte)
engineState = 2; // Moteur led devient vert
} else if (taskStates[0] == 1) { // Si la tâche est en cours (led orange)
engineState = 1; // Moteur led devient orange
} else {
engineState = 0; // Si la tâche est non commencée (led rouge)
}
// LED Moteur gauche
fill(engineState == 0 ? color(255, 0, 0) : (engineState == 1 ? color(255, 165, 0) : color(0, 255, 0)));
// Ligne du dessus : donne la couleur à la led en fonction de l'état
ellipse(390, 830, 10, 10); // Position de la led sur Moteur gauche en VDF
ellipse(395, 440, 10, 10); // Position de la led sur Moteur gauche en VDD
}
```

Figure 22 - Partie du code processing (gestion des états des leds par partie).

Je n'ai donné qu'un seul exemple, celui de la partie motrice de l'aéronef. Cependant, il faut ajouter le même type de code pour chaque composant sur lequel on va effectuer une maintenance. Donc dans notre cas 6 composants (voir liste des tâches). Pour ce faire, il suffit simplement de déclarer une nouvelle variable en début de code au niveau du « int engineState = 0; » en y ajoutant un nouveau nom de variable par exemple « int aileState = 0; » pour la gestion de la maintenance des ailes.

Puis, il suffit de modifier le code suivant de la manière suivante :

- Remplacer engineState par aileState à chaque fois dans le code.
- Remplacer le numéro de la taskStates[0] par un autre chiffre, par exemple : taskStates[1] et ainsi de suite à chaque nouvelle variable.

Quatrième partie de l'interface

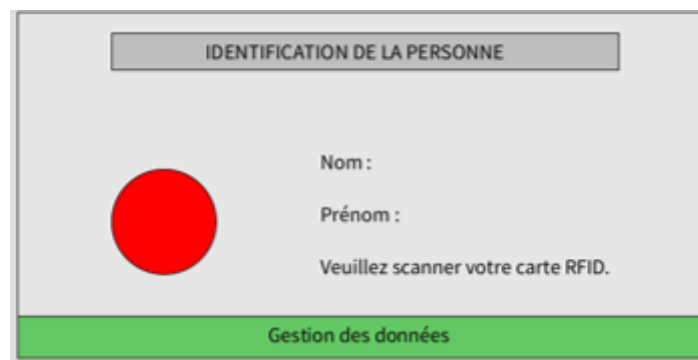


Figure 23 - Partie 4 de l'interface processing 4.

Cette partie va permettre d'identifier la personne qui effectue la maintenance de l'aéronef. Pour ce faire, nous avons une led qui va rester de couleur rouge lorsqu'aucun badge n'a été détecté. Un message « Veuillez scanner votre carte RFID. » va aussi s'afficher par la même occasion.

Dès qu'une personne va scanner sa carte RFID, le nom et le prénom de cette personne va venir s'afficher ainsi que la phrase « utilisateur reconnu. ». La led quant à elle deviendra de couleur verte.

Pour finir, nous avons dans cette partie un bouton « Gestion des données » qui va permettre à l'utilisateur de pouvoir accéder en un clic à un tableau Excel avec de nombreuses informations sur les anciennes maintenances de manière à pouvoir prévoir les futures maintenances possibles et d'en estimer le délai.

Voici le code utilisé pour effectuer cette partie :

```

10 // Chemin vers le fichier Excel pour le traitement de données
11 String filePath = "C:\\Users\\coral\\OneDrive\\Bureau\\Projet IOT Terminé\\Traitements.xlsx";
12
13 //int pour effectuer le dimensionnement du bouton
14 int buttonX;
15 int buttonY;
16 int buttonW;
17 int buttonH;
18
19 boolean cardScanned = false; // État de la carte scannée
20 boolean cardUnknown = false; // État pour carte non reconnue
21 String userName = ""; // Nom de l'utilisateur
22 String userFullName = ""; // Prénom de l'utilisateur
23 String texte = "";
24
25 // Variable pour empêcher la LED de l'utilisateur de revenir à rouge ou orange une fois verte
26 boolean userLedGreen = false; // Bloque la LED de l'utilisateur une fois qu'elle devient verte

```

Figure 24 - Code pour la partie identification de l'utilisateur + Gestion des données (partie 1).

```

52 void draw() {
53     background(255); // Fond blanc
54
55     // Affiche la LED et les informations de l'utilisateur (à gauche de l'interface)
56     if (cardScanned && !userLedGreen) {
57         fill(0, 255, 0); // LED verte si carte scannée et LED pas encore verte
58     } else if (userLedGreen) {
59         fill(0, 255, 0); // LED verte définitive (ne peut plus revenir à rouge ou orange)
60     } else {
61         fill(255, 0, 0); // LED rouge sinon
62     }
63     ellipse(1350, 900, 100, 100); // Dessin de la LED
64     // Va afficher le nom et prénom de la personne qui à badge :
65     fill(0);
66     text("Nom : " + userName, 1500, 850);
67     text("Prénom : " + userFullName, 1500, 900);
68
69     // N'affiche pas "Veuillez scanner votre RFID" si la LED de l'utilisateur est verte
70     if (cardScanned && !userLedGreen) {
71         text("Carte scannée avec succès.", 1500, 950);
72     } else if (userLedGreen) {
73         text("Utilisateur reconnu.", 1500, 950);
74     } else {
75         text("Veuillez scanner votre carte RFID.", 1500, 950);
76     }
77
78     // Dessine le bouton pour l'ouverture de l'excel de traitement de données :
79     buttonX = 1210;
80     buttonY = 990;
81     buttonW = 660;
82     buttonH = 40;
83     fill(100, 200, 100); // la couleur
84     rect(buttonX, buttonY, buttonW, buttonH); //forme rectangle avec variables ci-dessus
85     fill(0);
86     text("Gestion des données", 1450, 1015); // texte

```

Visuel du bouton de gestion des données.

Figure 25 - Code pour la partie identification de l'utilisateur + Gestion des données (partie 2).

```

88 void keyPressed() {
89     texte += key; // Ajoute le caractère saisi
90     // Appuyer sur 'S' pour sauvegarder
91     if (key == 's' || key == 'S') {
92         saveMaintenanceData();
93     }
94 }

```

Figure 26 - Code pour sauvegarder les données de la maintenance.

```

96 // Fonction appelée quand une ligne complète est reçue depuis Arduino :
97 void serialEvent(Serial myPort) {
98   String message = myPort.readStringUntil('\n'); // Lecture jusqu'à la fin de la ligne
99   if (message != null) {
100     message = message.trim(); // Retirer les espaces inutiles
101     println("Message reçu: " + message); // Affiche chaque message reçu pour le débogage
102     // Réinitialise les états
103     cardScanned = false;
104     cardUnknown = false;
105
106     // Gestion des utilisateurs principaux
107     if (message.startsWith("USER_")) {
108       String[] data = split(message.substring(5), ':'); // Ignore "USER_"
109       if (data.length == 2) {
110         userName = data[1]; // Nom après ":"
111         userFullName = data[0]; // Prénom avant ":"
112         cardScanned = true; // Met à jour l'état
113
114         // Bloque la LED de l'utilisateur en vert si elle n'est pas déjà verte
115         if (!userLedGreen) {
116           userLedGreen = true; // La LED devient verte et reste verte
117         }
118       }
119     }
120     // Carte inconnue
121     else if (message.equals("UNKNOWN_CARD")) {
122       cardUnknown = true;
123     }
124   }
125 }

```

Gestion de l'état de la led en fonction de l'IUD de l'utilisateur.

Figure 27 - Code pour la partie identification de l'utilisateur + Gestion des données (partie 3).

```

127 void saveMaintenanceData() {
128   // dossier CSV pour renseigner les données obtenues depuis l'interface
129   String fileName = "C:\\Users\\coral\\OneDrive\\Bureau\\Projet IOT Terminé\\taches.csv";
130   try {
131     File file = new File(fileName);
132     boolean fileExists = file.exists();
133     FileWriter csvWriter = new FileWriter(fileName, true);
134     if (!fileExists) {
135       csvWriter.append("Date,Utilisateur,Prénom,Nom de l'avion,Tâche,Temps (s),État\n");
136     }
137     String currentDate = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").format(new Date());
138     for (int i = 0; i < taskStates.length; i++) {
139       String taskState = "";
140       switch(taskStates[i]) {
141         case 0: taskState = "Non commencé"; break;
142         case 1: taskState = "En cours"; break;
143         case 2: taskState = "Terminé"; break;
144       }
145       if (taskStates[i] > 0) {
146         csvWriter.append(String.format("%s,%s,%s,%s,%s,%s,.2f,%s\n",
147           currentDate,
148           userFullName,
149           userName,
150           selectedPlane, // Utilisez selectedPlane ici
151           getTaskName(i),
152           taskTimes[i],
153           taskState
154         ));
155       }
156     }
157     csvWriter.close();
158     println("Données de maintenance ajoutées au fichier : " + fileName);
159   } catch (IOException e) {
160     println("Erreur lors de l'enregistrement du fichier CSV : " + e.getMessage());
161   }
162 }

```

Figure 28 - Code pour la partie identification de l'utilisateur + Gestion des données (partie 4).

```

166 void mousePressed() {
167     // Vérifier si le clic est sur le bouton
168     if (mouseX > boutonX && mouseX < boutonX + boutonW &&
169         mouseY > boutonY && mouseY < boutonY + boutonH) {
170         ouvrirFichierExcel(filePath);
171     }
172 }
173
174 void ouvrirFichierExcel(String cheminFichier) {
175     try {
176         File fichier = new File(cheminFichier);
177         if (!fichier.exists()) {
178             println("Fichier introuvable : " + cheminFichier);
179             return;
180         }
181         Desktop.getDesktop().open(fichier);
182         println("Fichier ouvert : " + cheminFichier);
183     } catch (Exception e) {
184         e.printStackTrace();
185         println("Erreur lors de l'ouverture du fichier.");
186     }
187 }
188

```

Clic de souris a activé le bouton ou non.

Ouverture du fichier Excel si conditions OK.

Figure 29 - Code pour la partie identification de l'utilisateur + Gestion des données (partie 5).

En assemblant les 5 parties de codes précédentes nous obtenons le résultat suivant :

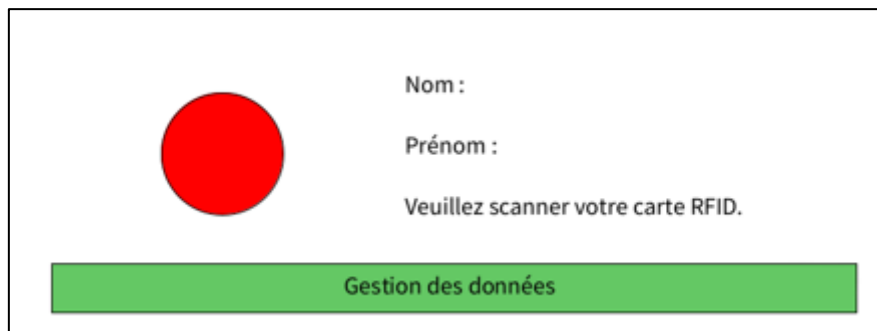


Figure 30 - Résultat du visuel obtenu.

Pour finir sur cette partie, si nous ajoutons l'ensemble des codes précédents, nous obtenons l'interface vue en début de partie, à savoir une interface interactive, dynamique et très intuitive pour son utilisateur.

c) Partie traitement des données et interprétations de ces données

Dans cette partie, le schéma de fonctionnement est le suivant :

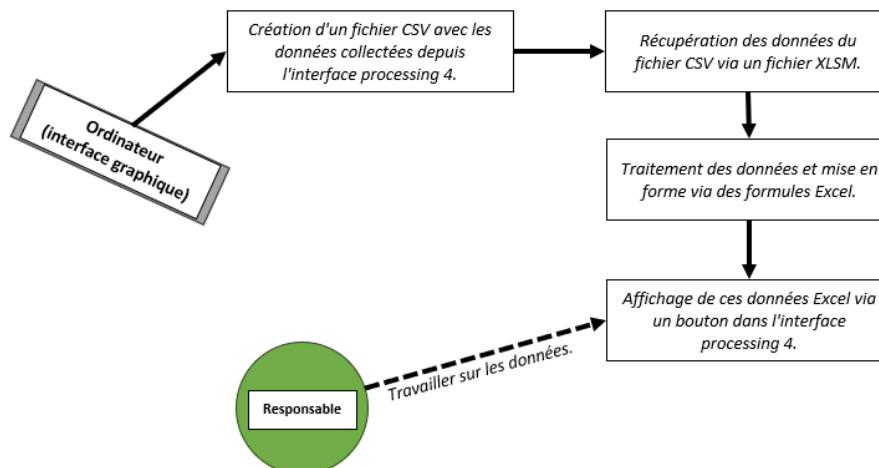


Figure 31 - Schéma de fonctionnement (Partie traitement des données).

Le responsable va avoir à sa disposition le matériel suivant :

- Une interface (processing 4) située sur un ordinateur.
- Un document Excel avec l'ensemble des données concernant la maintenance des aéronefs.

Procédure d'utilisation :

- 1 Le responsable va venir appuyer sur un bouton « Gestion des données » depuis l'interface (processing 4).
- 2 Le responsable va venir travailler sur les données concernant la maintenance des aéronefs.

Concernant la partie logiciel, l'ensemble de la partie processing a été abordée dans les parties précédentes. Au niveau de la partie Excel, il y a seulement des formules simples de manière à pouvoir passer d'une quantité de données importée depuis un fichier CSV à une feuille Excel avec des graphiques et d'autres outils de manière à mettre en forme ces données.

Le responsable aura accès aux deux feuilles Excel suivantes :

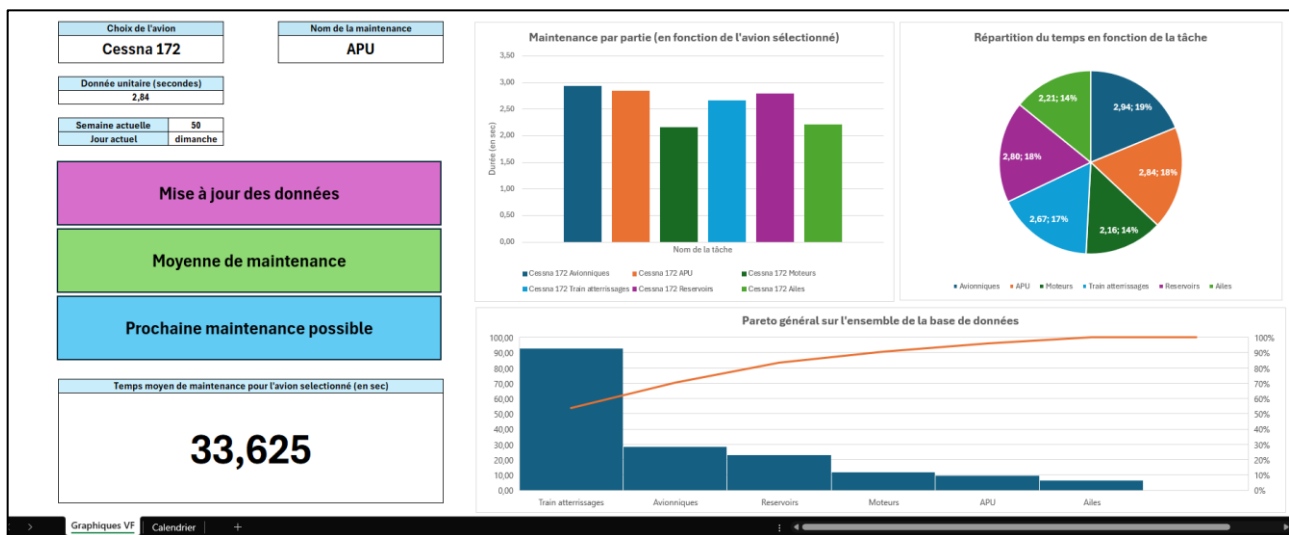


Figure 32 - Feuille Excel de données 1.

Depuis la feuille Excel « Graphiques VF », le responsable pourra :

- Sélectionner un aéronef.
- Sélectionner le nom d'une tâche.
- Mettre à jour les données.
- Connaître le temps moyen de maintenance pour l'aéronef sélectionné.
- Connaître la prochaine disponibilité pour effectuer la maintenance de l'aéronef.

Pour ce faire, le responsable devra suivre la procédure suivante :

- Sélectionner un aéronef via la liste déroulante.
- Sélectionner le nom d'une tâche via la liste déroulante.
- Appuyer sur le bouton « Mettre à jour des données ».
- Appuyer sur le bouton « Moyenne de maintenance ».
- Appuyer sur le bouton « Prochaine maintenance possible ».

A la suite de cela, le responsable aura accès au temps moyen de maintenance pour l'aéronef sélectionné ainsi que deux graphiques (précis concernant le nom de l'aéronef) et un diagramme Pareto (concernant la base de données totale avec tous les noms des aéronefs).

De manière à connaître les disponibilités pour effectuer la prochaine maintenance, le responsable va devoir mettre à jour le plus souvent possible le calendrier situé en feuille 2 de l'Excel (feuille « Calendrier »). Pour ce faire, il va venir indiquer si le jour est disponible (DISP) ou indisponible (INDISP).

Voici le calendrier :

semaine	jour	Disponibilité
semaine 50	Lundi	INDISP
semaine 50	Mardi	INDISP
semaine 50	Mercredi	INDISP
semaine 50	Jeudi	INDISP
semaine 50	Vendredi	INDISP
semaine 51	Lundi	DISP
semaine 51	Mardi	INDISP
semaine 51	Mercredi	INDISP
semaine 51	Jeudi	DISP
semaine 51	Vendredi	INDISP
semaine 52	Lundi	INDISP
semaine 52	Mardi	DISP
semaine 52	Mercredi	INDISP
semaine 52	Jeudi	INDISP
semaine 52	Vendredi	DISP
semaine 1	Lundi	DISP
semaine 1	Mardi	INDISP
semaine 1	Mercredi	INDISP
semaine 1	Jeudi	INDISP
semaine 1	Vendredi	INDISP

< > Graphiques VF Calendrier +

Figure 33 - Calendrier des disponibilités de maintenance.

Concernant l'interprétation de ces données, nous avons les graphiques suivants :

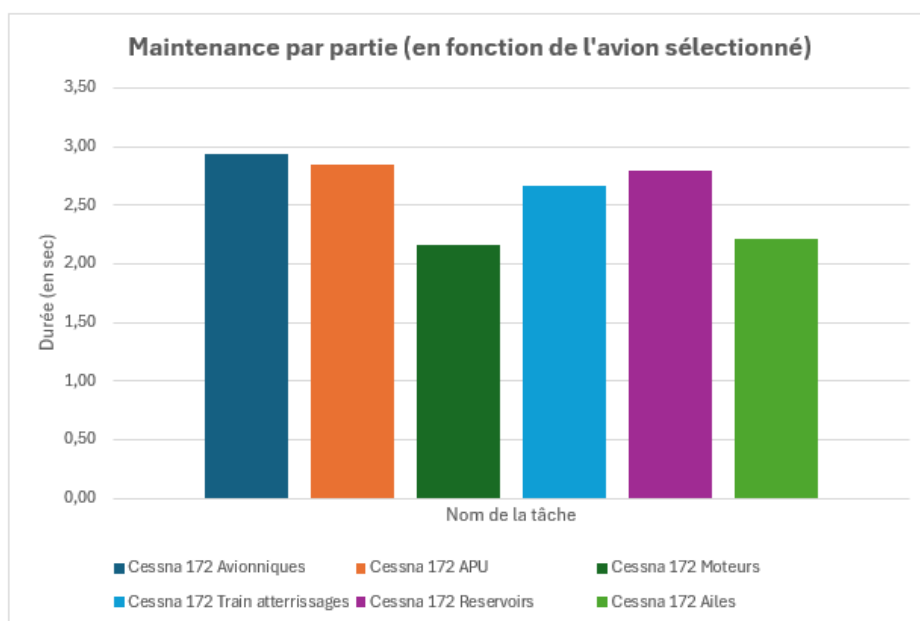


Figure 34 - Graphique 1 (Maintenance par partie (en fonction de l'avion sélectionné)).

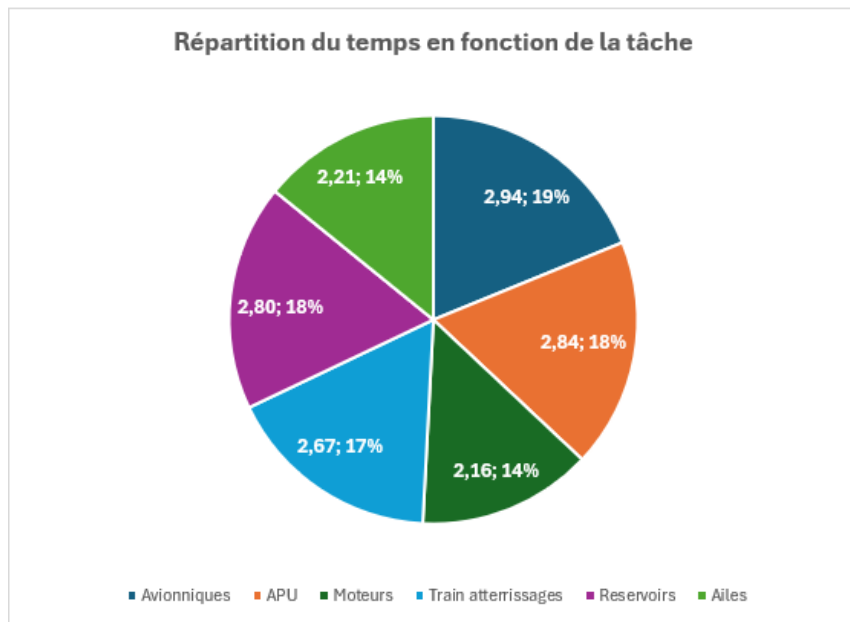


Figure 35 - Graphique 2 (Répartition du temps en fonction de la tâche).

Les graphiques précédents vont indiquer le temps de maintenance pour chaque partie de l'aéronef (les 6 tâches que l'on a identifiées dans notre code processing).

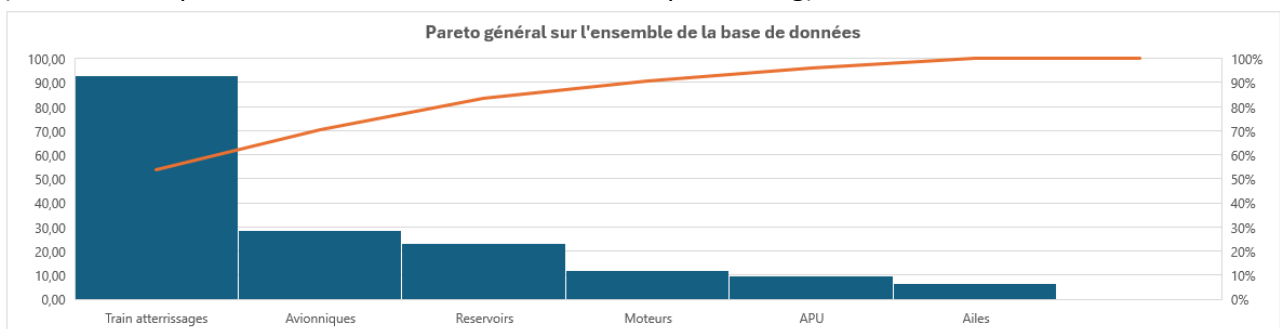


Figure 36 - Graphique 3 (Pareto général sur l'ensemble de la base de données).

Le diagramme ci-dessus va venir indiquer le temps de maintenance total effectué sur chaque type de maintenance. Donc à savoir le temps total passé sur chaque partie parmi les 6 que l'on a identifié dans notre processing.

d) Conclusion

Cette partie présentation du fonctionnement de notre système touche donc à sa fin. Nous avons donc à disposition un système fonctionnel, interactif et intuitif permettant à quiconque de pouvoir l'utiliser sans problème.

Cependant, il y a toutefois des améliorations à faire mais nous aborderons cela dans la partie « Pistes d'améliorations » de notre rapport.

PROTOTYPE

Dans cette partie, nous allons aborder l'ensemble des étapes réalisées pour obtenir notre prototype.

Pour ce faire, nous avons procédé de la manière suivante :

- Identification des besoins et du possible fonctionnement.
- Modélisation du prototype sous SolidWorks.
- Création du prototype.
- Essais sur le prototype.

Concernant l'étape d'identification des besoins et du possible fonctionnement du prototype, ce fût une étape très rapide car nous avons seulement un composant à incorporer (à savoir le module RFID).

Ensuite, nous avons réalisé la conception de notre prototype avec SolidWorks et nous obtenons le prototype suivant :

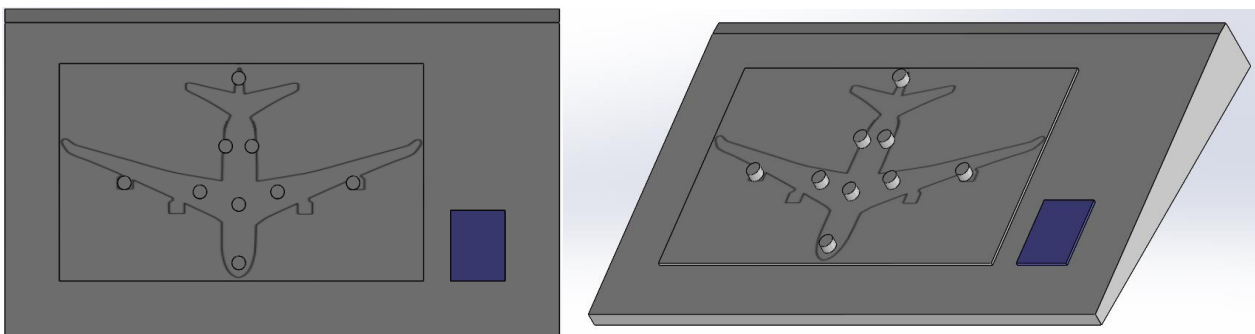


Figure 37 - CAO du prototype.

A la suite de cela, nous sommes allés chercher le matériel nécessaire pour la réalisation de ce prototype (voir « 2) Matériel qu'on a utilisé pour le projet »).

Au total, il nous a fallu 10 minutes pour concevoir cette CAO du prototype et près d'une demi-heure pour le concevoir avec les outils cités en début de rapport.

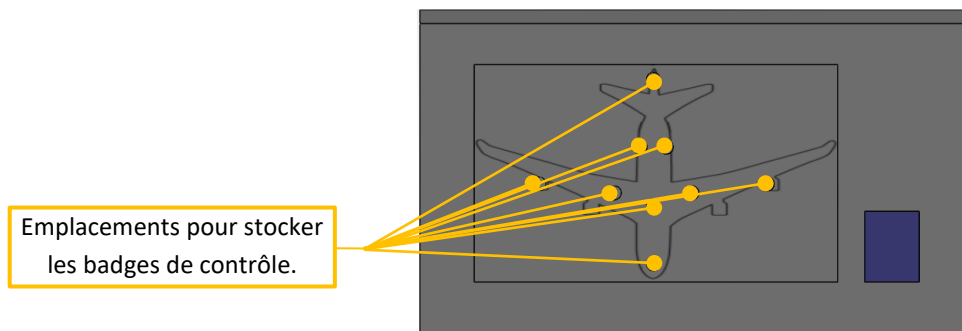


Figure 38 - Informations supplémentaires sur la CAO.

ETUDE DE COUTS

Pour effectuer cette partie, nous allons reprendre les données de coûts identifiés dans la partie « tarification du prototype » et allons ajouter à cela le coût de la main d'œuvre et de son possible aménagement en milieu professionnel.

Coût de l'électronique :

Nom du composant	Prix (en €)
Carte Arduino UNO	27.64 €
Module RFID + Badges + Cartes	4.85 €
Câbles	1.76 €
TOTAL	34.25 €

Figure 39 - Coût de l'électronique.

Coût de la main-d'œuvre :

Nous soumettons les hypothèses suivantes :

- Temps passé sur le projet : 80 heures.
- Salaire d'un ingénieur : 20€/h.
- Nombre de personne dans le groupe : 7.

Nombre de personne dans le groupe	Temps de travail (en h)	Salaire/heure	Total
7	80	20	1 600 €

Figure 40 - Coût de la main-d'œuvre.

Coût du pupitre amovible (amovible de manière à pouvoir le déplacer facilement dans l'atelier) :

Nous avons choisi le pupitre en aluminium suivant :



Prix du pupitre : 118.68 €



Prix du pupitre : dépend du devis.

Figure 41 - Pupitre amovible en aluminium.

Pupitre amovible trilogic (proposition).

Dans le cas où le prix n'est pas un problème, nous aurions pu choisir une solution « trilogic » (ci-dessus à droite) car c'est une entreprise spécialisée dans l'équipement de l'aviation.

Pour conclure, pour mettre en place ce système dans un milieu professionnel, la solution aurait un coût estimé de : **1 752.93 €** pouvant varier vers le positif si nous achetons de l'électronique de meilleure qualité et avec une meilleure performance.

REFLEXIONS SUR LE PROJET

Ce projet fut assez complexe à la vue de nos connaissances dans la matière mais aussi par manque de temps car le délai fut trop court pour nous. Mais nous avons su nous aider de vidéos, vlogs, IA pour y aboutir. Les résultats sont concluants mais il reste toutefois de nombreuses parties, fonctions à améliorer afin d'obtenir un projet efficace et complet.

Il nous a fallu près de 80 heures de travail pour obtenir le résultat actuel.

De ce fait, nous avons dû faire des choix que ce soit au niveau de la conception ou au niveau du code.

Et donc d'aller au plus simple de manière à avoir un code simple mais lisible par tous et surtout capable de répondre à notre cahier des charges.

C'est pour cela que la longueur du code est assez importante et qu'il y a des parties du code qui se ressemblent beaucoup (car nous avons utilisé beaucoup de copier-coller au lieu de créer une implémentation avec un `i++`).

LES PISTES D'AMÉLIORATIONS

Dans cette partie, nous allons aborder les pistes d'améliorations sur notre projet.

Dans un premier temps, il nous est possible d'améliorer notre code en le réduisant, ce qui va permettre à l'utilisateur de mieux comprendre le code, son fonctionnement et de pouvoir le modifier plus rapidement.

Ensuite, au niveau de l'interface (processing 4), nous pouvons améliorer notre système de liste déroulante. Pour rappel, nous avons une liste déroulante qui va permettre de sélectionner le nom de notre aéronef pendant la maintenance de celui-ci. Cependant, nos noms d'aéronefs sont directement renseignés dans le code source processing, donc cela reste accessible pour le modifier, mais si nous souhaitons faire cela, il y aura de nombreuses lignes de code à modifier pour y aboutir sans bug.

Pour lutter contre cela, nous pouvons créer un fichier Excel dans lequel nous renseignerons l'ensemble des noms de nos aéronefs. Cet Excel pourra varier et au niveau de notre code, nous gagnerons en place mais nous obtiendrons une liste déroulante optimisée car elle sera constamment à jour et pourra évoluer plus facilement dans le temps.

Nous pourrions aussi faire cela avec la liste des tâches de maintenances.

Nous pouvons aussi améliorer la partie accès aux données de maintenance, à savoir la partie bouton « Gestion des données ». Actuellement, tout le monde peut y accéder et cela peut possiblement poser problèmes en fonction de la situation. Pour faire face à cela, nous pourrions y bloquer l'accès par certaines personnes en fonction de leur IUD.

Nous pouvons aussi changer de type de base de données. Actuellement, nous utilisons des fichiers Excel, donc CSV et XSLM. Mais nous pouvons optimiser cela en utilisant du SQL (pour une meilleure fiabilité et le traitement plus efficace des données complexes).

De manière visuelle mais aussi audio, nous pourrions ajouter des hauts parleurs de manière à pouvoir énoncer la procédure à l'utilisateur. Nous pourrions aussi y intégrer un système de choix de langue et un système de bip ou de led afin d'avoir en plus du changement de couleur des led de l'interface, avoir quelque chose de visuel en temps réel.

Pour finir, nous pourrions intégrer une liste de procédure qui va venir indiquer au technicien les outils nécessaires pour la maintenance, une photo de la partie à contrôler et pour finir une petite gamme qui va permettre de guider la personne pendant la maintenance.

CONCLUSION

Nous avons répondu à notre cahier des charges en obtenant un prototype fonctionnel et capable d'aider un technicien lors de la maintenance d'un aéronef et un responsable dans sa gestion des maintenances.

Le projet fut très intéressant au niveau de la réflexion pour dans un premier identifié ce qui est possible de faire ou non et ensuite de la logique à avoir pour obtenir un résultat concluant.

Notre solution est donc viable, tant sur le plan de son coût que sur celui de sa mise en place et de son fonctionnement.

Cependant, il serait nécessaire de sélectionner des composants plus performants et mieux adaptés à la situation afin d'assurer sa durabilité à long terme.

Pour finir, ce fut une belle expérience malgré un temps de travail assez conséquent.

TABLE DES FIGURES

Figure 1 - Description du fonctionnement.	4
Figure 2 - Câblage du module RFID avec la carte Arduino Uno.	7
Figure 3 - Consommation électrique des composants.	8
Figure 4 - Schéma de fonctionnement (Partie maintenance et prise d'informations).	10
Figure 5 - Code Arduino (Partie pupitre).	11
Figure 6 - Code Arduino (Partie pupitre). (2)	11
Figure 7 - Code Arduino (Partie pupitre). (3)	12
Figure 8 - Code Arduino (Partie pupitre). (4)	12
Figure 9 - Schéma de fonctionnement (Partie visuelle et utilisation de l'interface (processing 4)). .	13
Figure 10 - Interface processing 4.	14
Figure 11 - Partie 1 de l'interface processing 4.	14
Figure 12 - Partie 2 de l'interface processing 4.	15
Figure 13 - Code processing pour la liste déroulante.	15
Figure 14 - Partie 3 de l'interface processing 4.	16
Figure 15 - Code de la partie visuelle.	16
Figure 16 - Résultat du visuel.	16
Figure 17 - Code pour la liste des tâches + Couleur led + chronomètre (partie 1).	17
Figure 18 - Code pour la liste des tâches + Couleur led + chronomètre (partie 2).	17
Figure 19 - Code pour la liste des tâches + Couleur led + chronomètre (partie 3).	17
Figure 20 - Code pour la liste des tâches + Couleur led + chronomètre (partie 4).	18
Figure 21 - Résultat de l'ajout du code précédent.	18
Figure 22 - Partie du code processing (gestion des états des leds par partie).	19
Figure 23 - Partie 4 de l'interface processing 4.	19
Figure 24 - Code pour la partie identification de l'utilisateur + Gestion des données (partie 1).	20
Figure 25 - Code pour la partie identification de l'utilisateur + Gestion des données (partie 2).	20
Figure 26 - Code pour sauvegarder les données de la maintenance.	20
Figure 27 - Code pour la partie identification de l'utilisateur + Gestion des données (partie 3).	21
Figure 28 - Code pour la partie identification de l'utilisateur + Gestion des données (partie 4).	21
Figure 29 - Code pour la partie identification de l'utilisateur + Gestion des données (partie 5).	22
Figure 30 - Résultat du visuel obtenu.	22
Figure 31 - Schéma de fonctionnement (Partie traitement des données).	22
Figure 32 - Feuille Excel de données 1.	23

<i>Figure 33 - Calendrier des disponibilités de maintenance.</i>	24
<i>Figure 34 - Graphique 1 (Maintenance par partie (en fonction de l'avion sélectionné)).</i>	24
<i>Figure 35 - Graphique 2 (Répartition du temps en fonction de la tâche).</i>	25
<i>Figure 36 - Graphique 3 (Pareto général sur l'ensemble de la base de données).</i>	25
<i>Figure 37 - CAO du prototype.</i>	26
<i>Figure 38 - Informations supplémentaires sur la CAO.</i>	26
<i>Figure 39 - Coût de l'électronique.</i>	27
<i>Figure 40 - Coût de la main-d'œuvre.</i>	27
<i>Figure 41 - Pupitre amovible en aluminium. Pupitre amovible trilogic (proposition).</i>	27

TABLE DES TABLEAUX

<i>Tableau 1 - Composants (partie électronique).</i>	<i>5</i>
<i>Tableau 2 - Composants (partie prototype).....</i>	<i>5</i>
<i>Tableau 3 - Logiciels.....</i>	<i>6</i>
<i>Tableau 4 - Tarification des composants.</i>	<i>6</i>

ANNEXES

Code Arduino et Processing.

Code Arduino :

CodearduinoiotREV3.ino

```

1  #include <SPI.h>
2  #include <MFRC522.h>
3  #define SS_PIN 10
4  #define RST_PIN 9
5  #define LED_PIN 7 // Pin de la LED physique
6
7  MFRC522 mfrc522(SS_PIN, RST_PIN);
8
9  // Structure pour associer UID et noms
10 struct User {
11     byte uid[4];
12     char name[16];
13 };
14
15 // Liste des utilisateurs autorisés (UID principaux)
16 User users[] = {
17     {{0xBB, 0x4E, 0xC6, 0x50}, "Alex:MARTIN"}, // Format : "Prénom:Nom"
18     {{0x64, 0xF2, 0x36, 0x5B}, "Julie:RESP"},
19 };
20
21 const int numUsers = sizeof(users) / sizeof(users[0]);
22
23 // Liste des badges secondaires avec leurs tâches spécifiques
24 struct Task {
25     byte uid[4];
26     char taskName[20]; // Tâche associée
27 };
28
29 // Référence des badges secondaires (UID secondaires)
30 Task secondaryTasks[] = {
31     {{0x21, 0xFC, 0x66, 0x1E}, "Avionniques"},
32     {{0x41, 0x86, 0x67, 0x1E}, "APU"},
33     {{0x39, 0x96, 0xAB, 0xE2}, "Moteurs"},
34     {{0x1C, 0x69, 0xB5, 0x6D}, "Train atterrissages"},
35     {{0xD1, 0x07, 0x67, 0x1E}, "Reservoirs"},
36     {{0xB1, 0xF6, 0x66, 0x1E}, "Ailes"},
37 };
38
39 const int numTasks = sizeof(secondaryTasks) / sizeof(secondaryTasks[0]);
40

```

```

41 void setup() {
42     Serial.begin(9600);
43     SPI.begin();
44     mfrc522.PCD_Init();
45     pinMode(LED_PIN, OUTPUT);
46     digitalWrite(LED_PIN, LOW);
47     Serial.println("Approche ta carte...");
48 }
49

```

```

50 void loop() {
51     // Attente de la détection d'une nouvelle carte
52     if (!mfrc522.PICC_IsNewCardPresent() || !mfrc522.PICC_ReadCardSerial()) {
53         return;
54     }
55
56     byte* uid = mfrc522.uid.uidByte;
57     bool userFound = false;
58
59     // Vérifie si l'UID correspond à un utilisateur principal enregistré
60     for (int i = 0; i < numUsers; i++) {
61         if (compareUID(uid, users[i].uid)) {
62             Serial.print("USER_");
63             Serial.println(users[i].name); // Envoie "USER_Prenom:Nom" à Processing
64             digitalWrite(LED_PIN, HIGH); // Allume la LED physique pour succès
65             delay(2000); // Maintient la LED allumée pendant 2 secondes
66             digitalWrite(LED_PIN, LOW);
67             userFound = true;
68             break;
69         }
70     }
71
72     // Vérifie si l'UID correspond à un badge secondaire pour les tâches
73     if (!userFound) {
74         for (int i = 0; i < numTasks; i++) {
75             if (compareUID(uid, secondaryTasks[i].uid)) {
76                 Serial.print("TASK_");
77                 Serial.println(secondaryTasks[i].taskName); // Envoie le nom de la tâche à Processing
78                 digitalWrite(LED_PIN, HIGH); // Allume la LED pour indiquer une tâche secondaire reconnue
79                 delay(2000);
80                 digitalWrite(LED_PIN, LOW);
81                 userFound = true;
82                 break;
83             }
84         }
85     }
86

```

```

87     // Si aucun utilisateur ou tâche n'a été trouvée avec cet UID
88     if (!userFound) {
89         Serial.println("UNKNOWN_CARD"); // Envoie le message pour carte inconnue
90         digitalWrite(LED_PIN, LOW); // Assure que la LED reste éteinte
91     }
92
93     mfrc522.PICC_HaltA(); // Arrête la communication avec la carte
94 }
95
96 // Fonction pour comparer deux UID
97 bool compareUID(byte* uid1, byte* uid2) {
98     for (byte i = 0; i < 4; i++) {
99         if (uid1[i] != uid2[i]) return false;
100     }
101     return true;
102 }
103

```

Code processing :

```

InterfaceFinaleIOTProject
1 import processing.serial.*;
2 import java.io.FileWriter;
3 import java.io.IOException;
4 import java.text.SimpleDateFormat;
5 import java.util.Date;
6 import controlP5.*;
7 import java.awt.Desktop;
8 import java.io.File;
9
10 String filePath = "C:\\Users\\coral\\OneDrive\\Bureau\\Projet IOT Terminé\\Traitements.xlsx"; // Chemin vers le fichier Excel pour le traitement de données
11
12 //int pour effectuer le dimensionnement du bouton
13 int buttonX;
14 int buttonY;
15 int buttonW;
16 int buttonH;
17
18 //Pour faire la liste déroulante pour le choix de l'avion
19 ControlP5 cp5;
20 DropDownList avionList;
21 String[] avions = {"Boeing 737", "Airbus A320", "Concorde", "Cessna 172", "Airbus A220-300", "Airbus A220-100", "Boeing 777", "Boeing 747"};
22
23 Serial myPort; // Objet pour la communication série
24
25 boolean cardScanned = false; // État de la carte scannée
26 boolean cardUnknown = false; // État pour carte non reconnue
27 String userName = ""; // Nom de l'utilisateur
28 String userFullName = ""; // Prénom de l'utilisateur
29 String texte = "";
30 boolean actif = false;
31
32 int wingLeftState = 0;
33 int wingRightState = 0;
34 int engineState = 0;
35 int GouverneState = 0;
36 int CockpitState = 0;
37 int EclairageState = 0;
38 int ReserveState = 0;
39
40 // Tableau d'états pour les tâches secondaires (0 = rouge, 1 = orange, 2 = vert)
41 int[] taskStates = {0, 0, 0, 0, 0, 0};
42 float[] taskStartTimes = new float[6]; // Tableau pour stocker le moment où la tâche devient orange
43 float[] taskTimes = new float[6]; // Tableau pour stocker le temps écoulé pour chaque tâche
44

```

```

InterfaceFinaleIOTProject
46 // Variable pour empêcher la LED de l'utilisateur de revenir à rouge ou orange une fois verte
47 boolean userLedGreen = false; // Bloque la LED de l'utilisateur une fois qu'elle devient verte
48
49 String selectedPlane = "";
50
51 void setup() {
52   fullScreen(); // pour avoir taille écran max peut importe l'ordinateur
53   myPort = new Serial(this, "COM3", 9600);
54   myPort.bufferUntil('\n'); // Lecture ligne complète depuis Arduino
55   textSize(20); //va donner à l'ensemble de l'interface la taille de texte indiquée
56   // Créer une instance de ControlP5
57   cp5 = new ControlP5(this);
58   // Créer la liste déroulante et ajouter les éléments
59   avionList = cp5.addDropDownList("Sélection de l'avion")
60     .setPosition(450, 940)
61     .setSize(720, 100)
62     .addItem(avions)
63     .setValue(0) // Sélectionne l'élément par défaut
64     .plugTo(this, "avionSelect"); // code pour connecter l'évènement
65   // Style optionnel pour améliorer l'affichage
66   avionList.setItemHeight(30);
67   avionList.setBarHeight(30); //Pour épaisseur de la barre de la liste déroulante
68 }
69
70 // Méthode appelée lorsqu'un élément est sélectionné
71 public void avionSelect(int n) {
72   selectedPlane = avions[n];
73   println("Avion sélectionné : " + selectedPlane);
74 }
75

```

InterfaceFinaleIOTProject

```

76 void draw() {
77     background(255); // Fond blanc
78     drawPlane();
79     drawLEDs();
80     drawProgressBar();
81     //Code pour les zones de texte et le visuel de l'interface :
82     fill(220); // couleur du rect ci-dessous
83     rect(1200, 35, 680, 1010); // rectangle forme
84     fill(190);
85     rect(1210, 40, 80, 35);
86     fill(0); // couleur du texte ci-dessous
87     text("ETAT", 1230, 65);
88     fill(190);
89     rect(1300, 40, 425, 35);
90     fill(0);
91     text("NOM DE LA TÂCHE", 1435, 65);
92     fill(190);
93     rect(1734, 40, 136, 35);
94     fill(0);
95     text("TEMPS", 1770, 65);
96     fill(230);
97     rect(1210, 90, 80, 600);
98     fill(230);
99     rect(1300, 90, 425, 600);
100    fill(230);
101    rect(1734, 90, 136, 600);
102    fill(230);
103    rect(1210, 700, 660, 330);
104    fill(190);
105    rect(1300, 720, 485, 35);
106    fill(0);
107    text("IDENTIFICATION DE LA PERSONNE", 1390, 745);
108
109    //Plaque des noms
110    fill(190); //la couleur du rect ci-dessous
111    rect(200, 50, 840, 35);
112    fill(0); //la couleur du texte ci-dessous
113    text("AVION VU DU DESSUS", 540, 75);
114    fill(190);
115    rect(200, 640, 840, 35);
116    fill(0);
117    text("AVION VU DE FACE", 540, 665);
118    //plaque d'idd de l'avion (en bs à gauche interfC)
119    fill(190);
120    rect(60, 930, 1120, 106);
121    fill(0);

```

```

122 text("MAINTENANCE EFFECTUEE SUR L'AVION :", 80, 990);
123 fill(230);
124 rect(450, 940, 720, 86);
125
126 // Affiche la LED et les informations de l'utilisateur (à gauche de l'interface)
127 if (cardScanned && !userLedGreen) {
128   fill(0, 255, 0); // LED verte si carte scannée et LED pas encore verte
129 } else if (userLedGreen) {
130   fill(0, 255, 0); // LED verte définitive (ne peut plus revenir à rouge ou orange)
131 } else {
132   fill(255, 0, 0); // LED rouge sinon
133 }
134 ellipse(1350, 900, 100, 100); // Dessin de la LED
135 // Va afficher le nom et prénom de la personne qui à badge :
136 fill(0);
137 text("Nom : " + userName, 1500, 850);
138 text("Prénom : " + userFullName, 1500, 900);
139
140 // N'affiche pas "Veuillez scanner votre RFID" si la LED de l'utilisateur est verte
141 if (cardScanned && !userLedGreen) {
142   text("Carte scannée avec succès.", 1500, 950);
143 } else if (userLedGreen) {
144   text("Utilisateur reconnu.", 1500, 950);
145 } else {
146   text("Veuillez scanner votre carte RFID.", 1500, 950);
147 }
148
149 // Dessiner le bouton pour l'ouverture de l'excel de traitement de données :
150 buttonX = 1210;
151 buttonY = 990;
152 buttonW = 660;
153 buttonH = 40;
154 fill(100, 200, 100); // la couleur
155 rect(buttonX, buttonY, buttonW, buttonH);
156 fill(0);
157 text("Gestion des données", 1450, 1015);
158
159 // Liste de phrases à afficher (à droite de l'interface au niveau de la liste de tâches) :
160 String[] phrases = {
161   "Moteur droit et gauche.",
162   "Train d'atterrissage.",
163   "Ailes (structures et dispositifs de contrôle).",
164   "APU (Auxiliary Power Unit).",
165   "Réservoirs.",
166   "Avioniques."
167 };

```

```

169         float totalTime = 0.0;
170         for (int i = 0; i < taskTimes.length; i++) {
171             if (taskStates[i] == 2) { // Va seulement comptabiliser les tâches terminées
172                 totalTime += taskTimes[i]; // Additionne les temps
173             }
174         }
175
176         // Temps total de maintenance (Rectangle noir à droite de l'interface) :
177         fill(0);
178         rect(1300, 630, 425, 50);
179         fill(255);
180         text("TEMPS TOTAL : " + nf(totalTime, 1, 2) + " min", 1400, 660);
181
182         // Affichage de la liste avec LED à côté et des chronomètres
183         for (int i = 0; i < phrases.length; i++) {
184             fill(0);
185             text(phrases[i], 1320, 120 + (i * 40)); //pour changer emplacement du texte des tâches secondaires
186
187             // Affichage du temps de chaque tâche une fois la LED verte
188             if (taskStates[i] == 2) { // Si la LED est verte (tâche terminée)
189                 text(nf(taskTimes[i], 0, 2) + " " + "min", 1750, 120 + (i * 40)); // Chrono de la tâche (deux chiffres apr virgule)
190             } else {
191                 text("En cours...", 1750, 120 + (i * 40)); // Affichage si tâche est en cours
192             }
193
194             // Détermine la couleur de la LED - en fonction état actuel
195             if (taskStates[i] == 0) {
196                 fill(255, 0, 0); // Rouge
197             } else if (taskStates[i] == 1) {
198                 fill(255, 165, 0); // Orange
199             } else if (taskStates[i] == 2) {
200                 fill(0, 255, 0); // Vert
201             }
202             ellipse(1250, 115 + (i * 40), 20, 20); // Dessin de la LED
203         }
204     }
205
206     void keyPressed() {
207         texte += key; // Ajoute le caractère saisi
208         // Par exemple, appuyer sur 'S' pour sauvegarder
209         if (key == 's' || key == 'S') {
210             saveMaintenanceData();
211         }
212     }
213
214 void drawPlane() { //schéma de l'avion st à gch de la struct
215     fill(230); //couleur de la forme ci-dessous
216     rect(50, 35, 1140, 1010); // forme et dimensions de la forme mais aussi position : rect(x, y, x2, y2)
217     // Avion vu du dessus :
218     fill(150);
219     rect(380, 410, 30, 40); //Moteur gauche
220     rect(730, 410, 30, 40); //Moteur droit
221     rect(520, 200, 100, 300); // Corps de l'avion
222     rect(470, 350, 50, 100); // Aile sup gauche
223     rect(620, 350, 50, 100); // Aile sup droite
224     triangle(470, 350, 470, 450, 200, 350); // Aile gauche
225     triangle(670, 350, 670, 450, 940, 350); // Aile droite
226     triangle(520, 200, 520, 270, 450, 200); // Aileron gauche
227     triangle(620, 200, 620, 270, 690, 200); // Aileron droit
228     triangle(450, 180, 450, 200, 519, 200); // Aileron gauche pour faire beau
229     triangle(690, 180, 620, 200, 690, 200); // Aileron droit pour faire beau
230     rect(555, 180, 30, 80); //Aileron central (gouverne)
231     triangle(620, 500, 570, 570, 520, 500); // Avant de l'avion (cockpit)
232     triangle(620, 350, 941, 350, 940, 300); // le triangle pour faire beau en arrière de l'aile (flaps...) du VDD droite
233     triangle(520, 350, 199, 350, 199, 300); // le triangle pour faire beau en arrière de l'aile (flaps...) du VDD gauche
234     //Avion vu de face :
235     rect(570, 750, 80, 20); // Aileron droit
236     rect(490, 750, 80, 20); // Aileron gauche
237     rect(555, 730, 30, 60); // Aileron central (gouvernail)
238     rect(180, 810, 800, 20); // Ailes
239     ellipse(390, 830, 50, 50); // Moteur gauche
240     ellipse(750, 830, 50, 50); // Moteur droit
241     ellipse(570, 820, 100, 100); // Cabines de l'avion
242     ellipse(570, 830, 70, 70); // Cabines de l'avion p2
243 }
244

```

```

245 void drawLEDs() {
246
247 // LED Moteur gauche
248 fill(engineState == 0 ? color(255, 0, 0) : (engineState == 1 ? color(255, 165, 0) : color(0, 255, 0)));
249 ellipse(390, 830, 10, 10); // Position de la LED sur Moteur gauche en VDF
250 ellipse(395, 440, 10, 10); // Position de la LED sur Moteur gauche en VDD
251
252 // LED Moteur droit
253 fill(engineState == 0 ? color(255, 0, 0) : (engineState == 1 ? color(255, 165, 0) : color(0, 255, 0)));
254 ellipse(750, 830, 10, 10); // Position de la LED sur Moteur droit en VDF
255 ellipse(745, 440, 10, 10); // Position de la LED sur Moteur droit en VDD
256
257 // LED APU
258 fill(GouverneState == 0 ? color(255, 0, 0) : (GouverneState == 1 ? color(255, 165, 0) : color(0, 255, 0)));
259 ellipse(570, 190, 10, 10); // Position de la LED sur Aileron gauche en VDD
260 ellipse(570, 740, 10, 10); // Position de la LED sur Aileron gauche en VDF
261
262 // LED Train d'atterrissages
263 fill(CockpitState == 0 ? color(255, 0, 0) : (CockpitState == 1 ? color(255, 165, 0) : color(0, 255, 0)));
264 ellipse(570, 350, 10, 10); // Position de la LED sur Aileron gauche en VDD
265 ellipse(570, 500, 10, 10); // Position de la LED sur Aileron gauche en VDD
266 ellipse(570, 875, 10, 10); // Position de la LED sur Aileron gauche en VDF
267
268 // Eclairages
269 fill(EclairageState == 0 ? color(255, 0, 0) : (EclairageState == 1 ? color(255, 165, 0) : color(0, 255, 0)));
270 ellipse(200, 350, 10, 10); // Position de la LED sur Aileron gauche en VDD celui au niveau des winglets gauche
271 ellipse(940, 350, 10, 10); // Position de la LED sur Aileron gauche en VDD celui au niveau des winglets droit
272 ellipse(520, 400, 10, 10); // Position de la LED sur Aileron gauche en VDD au niveau des cabines gauche
273 ellipse(620, 400, 10, 10); // Position de la LED sur Aileron gauche en VDD au niveau des cabines droit
274 ellipse(520, 300, 10, 10); // Position de la LED sur Aileron gauche en VDD au niveau des cabines gauche
275 ellipse(620, 300, 10, 10); // Position de la LED sur Aileron gauche en VDD au niveau des cabines droit
276 ellipse(450, 200, 10, 10); // Position de la LED sur Aileron gauche en VDD au niveau des ailerons gauche
277 ellipse(690, 200, 10, 10); // Position de la LED sur Aileron gauche en VDD au niveau des ailerons droit
278 ellipse(490, 750, 10, 10);
279 ellipse(651, 750, 10, 10);
280 ellipse(180, 810, 10, 10);
281 ellipse(982, 810, 10, 10);
282
283 ellipse(520, 830, 10, 10); // Position de la LED sur Aileron gauche en VDD au niveau des cabines gauche
284 ellipse(620, 830, 10, 10); // Position de la LED sur Aileron gauche en VDD au niveau des cabines droit
285
286 // Réservoirs
287 fill(ReserveState == 0 ? color(255, 0, 0) : (ReserveState == 1 ? color(255, 165, 0) : color(0, 255, 0)));
288 ellipse(570, 300, 10, 10); // Position de la LED gauche en VDD
289 ellipse(600, 863, 10, 10); // Position de la LED gauche en VDF
290

```



```

291 // Code pour le changement de led en fonction de l'état du badge (si badge 1 ou 2 fois):
292 if (taskStates[0] == 2) { // Si la tâche "Moteur droit et gauche" est terminée (LED verte)
293     engineState = 2; // Moteur devient vert
294 } else if (taskStates[0] == 1) { // Si la tâche est en cours (LED orange)
295     engineState = 1; // Moteur devient orange
296 } else {
297     engineState = 0; // Si la tâche est non commencée (LED rouge)
298 }
299 // LED Moteur gauche
300 fill(engineState == 0 ? color(255, 0, 0) : (engineState == 1 ? color(255, 165, 0) : color(0, 255, 0)));
301 ellipse(390, 830, 10, 10); // Position de la LED sur Moteur gauche en VDF
302 ellipse(395, 440, 10, 10); // Position de la LED sur Moteur gauche en VDD
303
304 // Moteur droit :
305 if (taskStates[0] == 2) { // Si la tâche est terminée
306     engineState = 2;
307 } else if (taskStates[0] == 1) { // Si la tâche est en cours
308     engineState = 1;
309 } else {
310     engineState = 0; // Si la tâche est non commencée
311 }
312 // LED Moteur droit
313 fill(engineState == 0 ? color(255, 0, 0) : (engineState == 1 ? color(255, 165, 0) : color(0, 255, 0)));
314 ellipse(750, 830, 10, 10); // Position de la LED sur Moteur droit en VDF
315 ellipse(745, 440, 10, 10); // Position de la LED sur Moteur droit en VDD
316
317 // Train atterrissage :
318 if (taskStates[1] == 2) {
319     CockpitState = 2; // Ailes deviennent vertes
320     CockpitState = 2;
321 } else if (taskStates[1] == 1) {
322     CockpitState = 1; // Ailes deviennent oranges
323     CockpitState = 1;
324 } else {
325     CockpitState = 0; // Ailes deviennent rouges
326     CockpitState = 0;
327 }
328 fill(CockpitState == 0 ? color(255, 0, 0) : (CockpitState == 1 ? color(255, 165, 0) : color(0, 255, 0)));
329 ellipse(570, 350, 10, 10); // Position de la LED sur Aileron gauche en VDD
330 ellipse(570, 500, 10, 10); // Position de la LED sur Aileron gauche en VDD
331 ellipse(570, 875, 10, 10); // Position de la LED sur Aileron gauche en VDF
332
333 // APU :
334 if (taskStates[3] == 2) {
335     wingLeftState = 2; // Ailes deviennent vertes
336 } else if (taskStates[3] == 1) {
337     wingLeftState = 1; // Ailes deviennent oranges
338 } else {
339     wingLeftState = 0; // Ailes deviennent rouges
340 }
341 // LED Ailes gauche
342 fill(wingLeftState == 0 ? color(255, 0, 0) : (wingLeftState == 1 ? color(255, 165, 0) : color(0, 255, 0)));
343 ellipse(300, 315, 10, 10); // Position de la LED sur Aile Gauche
344 ellipse(840, 315, 10, 10); // Position de la LED sur Aile Droite
345
346 // Gouverne :
347 if (taskStates[2] == 2) {
348     GouverneState = 2; // Led ailes deviennent vertes
349 } else if (taskStates[2] == 1) {
350     GouverneState = 1; // Led ailes deviennent oranges
351 } else {
352     GouverneState = 0; // Led ailes deviennent rouges
353 }
354 fill(GouverneState == 0 ? color(255, 0, 0) : (GouverneState == 1 ? color(255, 165, 0) : color(0, 255, 0)));
355 ellipse(570, 190, 10, 10); // Position de la LED sur Aileron gauche en VDD
356 ellipse(570, 740, 10, 10); // Position de la LED sur Aileron gauche en VDF
357
358 // Eclairage :
359 if (taskStates[5] == 2) {
360     EclairageState = 2; // Ailes deviennent vertes
361 } else if (taskStates[5] == 1) {
362     EclairageState = 1; // Ailes deviennent oranges
363 } else {
364     EclairageState = 0; // Ailes deviennent rouges
365 }
366 fill(EclairageState == 0 ? color(255, 0, 0) : (EclairageState == 1 ? color(255, 165, 0) : color(0, 255, 0)));
367 ellipse(200, 350, 10, 10); // Position de la LED sur Aileron gauche en VDD celui au niveau des winglets gauche
368 ellipse(940, 350, 10, 10); // Position de la LED sur Aileron gauche en VDD celui au niveau des winglets droit
369 ellipse(520, 400, 10, 10); // Position de la LED sur Aileron gauche en VDD au niveau des cabines gauche
370 ellipse(620, 400, 10, 10); // Position de la LED sur Aileron gauche en VDD au niveau des cabines droit
371 ellipse(520, 300, 10, 10); // Position de la LED sur Aileron gauche en VDD au niveau des cabines gauche
372 ellipse(620, 300, 10, 10); // Position de la LED sur Aileron gauche en VDD au niveau des cabines droit
373 ellipse(450, 200, 10, 10); // Position de la LED sur Aileron gauche en VDD au niveau des ailerons gauche
374 ellipse(690, 200, 10, 10); // Position de la LED sur Aileron gauche en VDD au niveau des ailerons droit
375

```

```

376 // Réservoirs :
377 if (taskStates[4] == 2) {
378   ReserveState = 2; // Ailes deviennent vertes
379 } else if (taskStates[4] == 1) {
380   ReserveState = 1; // Ailes deviennent oranges
381 } else {
382   ReserveState = 0; // Ailes deviennent rouges
383 }
384 fill(ReserveState == 0 ? color(255, 0, 0) : (ReserveState == 1 ? color(255, 165, 0) : color(0, 255, 0)));
385 ellipse(570, 300, 10, 10); // Position de la LED gauche en VDD
386 ellipse(600, 863, 10, 10); // Position de la LED gauche en VDF
387 }
388
389 // Fonction appelée quand une ligne complète est reçue depuis Arduino :
390 void serialEvent(Serial myPort) {
391   String message = myPort.readStringUntil('\n'); // Lecture jusqu'à la fin de la ligne
392   if (message != null) {
393     message = message.trim(); // Retirer les espaces inutiles
394     println("Message reçu: " + message); // Affiche chaque message reçu pour le débogage
395     // Réinitialise les états
396     cardScanned = false;
397     cardUnknown = false;
398
399     // Gestion des utilisateurs principaux
400     if (message.startsWith("USER_")) {
401       String[] data = split(message.substring(5), ':'); // Ignore "USER_"
402       if (data.length == 2) {
403         userName = data[1]; // Nom après ":"
404         userFullName = data[0]; // Prénom avant ":"
405         cardScanned = true; // Met à jour l'état
406
407         // Bloque la LED de l'utilisateur en vert si elle n'est pas déjà verte
408         if (!userLedGreen) {
409           userLedGreen = true; // La LED devient verte et reste verte
410         }
411       }
412     }
413     // Gestion des tâches secondaires
414     else if (message.startsWith("TASK_")) {
415       // Recherche de la tâche et mise à jour de son état
416       String taskName = message.substring(5); // Ignore "TASK_"
417       for (int i = 0; i < taskStates.length; i++) {
418         if (taskName.equals(getTaskName(i))) {
419           if (taskStates[i] == 0) { // Si la LED est rouge (étape non commencée)
420             taskStates[i] = 1; // Passe à l'état orange (tâche en cours)
421             taskStartTimes[i] = millis() / 1000.0; // Démarre le chrono à l'instant actuel
422           } else if (taskStates[i] == 1) { // Si la LED est orange (tâche en cours)
423             taskStates[i] = 2; // Passe à l'état vert (tâche terminée)
424             taskTimes[i] = (millis() / 1000.0) - taskStartTimes[i]; // Calcule le temps écoulé pour la tâche
425           }
426         }
427       }
428     }
429     // Carte inconnue
430     else if (message.equals("UNKNOWN_CARD")) {
431       cardUnknown = true;
432     }
433   }
434 }

```

```

436 void saveMaintenanceData() {
437     String fileName = "C:\\Users\\coral\\OneDrive\\Bureau\\Projet IOT Terminé\\taches.csv"; // dossier CSV pour renseigner les données obtenues depuis l'interface
438     try {
439         File file = new File(fileName);
440         boolean fileExists = file.exists();
441         FileWriter csvWriter = new FileWriter(fileName, true);
442         if (!fileExists) {
443             csvWriter.append("Date,Utilisateur,Prénom,Nom de l'avion,Tâche,Temps (s),État\n");
444         }
445         String currentDate = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").format(new Date());
446         for (int i = 0; i < taskStates.length; i++) {
447             String taskState = "";
448             switch(taskStates[i]) {
449                 case 0: taskState = "Non commencé"; break;
450                 case 1: taskState = "En cours"; break;
451                 case 2: taskState = "Terminé"; break;
452             }
453             if (taskStates[i] > 0) {
454                 csvWriter.append(String.format("%s,%s,%s,%s,%s,%s,%.2f,%s\n",
455                     currentDate,
456                     userFullName,
457                     userName,
458                     selectedPlane, // Utilisez selectedPlane ici
459                     getTaskName(i),
460                     taskTimes[i],
461                     taskState
462                 ));
463             }
464         }
465         csvWriter.close();
466         println("Données de maintenance ajoutées au fichier : " + fileName);
467     } catch (IOException e) {
468         println("Erreur lors de l'enregistrement du fichier CSV : " + e.getMessage());
469     }
470 }
471
472 void mousePressed() {
473     // Vérifier si le clic est sur le bouton
474     if (mouseX > boutonX && mouseX < boutonX + boutonW &&
475         mouseY > boutonY && mouseY < boutonY + boutonH) {
476         ouvrirFichierExcel(filePath);
477     }
478 }
479

```

```

480 void ouvrirFichierExcel(String cheminFichier) {
481     try {
482         File fichier = new File(cheminFichier);
483         if (!fichier.exists()) {
484             println("Fichier introuvable : " + cheminFichier);
485             return;
486         }
487         Desktop.getDesktop().open(fichier);
488         println("Fichier ouvert : " + cheminFichier);
489     } catch (Exception e) {
490         e.printStackTrace();
491         println("Erreur lors de l'ouverture du fichier.");
492     }
493 }
494
495 void drawProgressBar() {
496     float percentage = getPercentage(); // Obtenir le pourcentage de LEDs vertes
497     // Afficher la barre de progression :
498     fill(100);
499     rect(50, height - 30, 1830, 20); // Fond de la barre de progression
500     fill(0, 255, 0); // Couleur verte pour la progression
501     rect(50, height - 30, map(percentage, 0, 100, 0, 1830), 20); // Barre de progression
502 }
503

```

```

495 void drawProgressBar() {
496     float percentage = getPercentage(); // Obtenir le pourcentage de LEDs vertes
497     // Afficher la barre de progression :
498     fill(100);
499     rect(50, height - 30, 1830, 20); // Fond de la barre de progression
500     fill(0, 255, 0); // Couleur verte pour la progression
501     rect(50, height - 30, map(percentage, 0, 100, 0, 1830), 20); // Barre de progression
502 }
503
504 float getPercentage() {
505     int totalLEDs = 6; // Compter toutes les LEDs surveillées
506     int greenLEDs = 0;
507     // Liste de tous les états des composants :
508     int[] states = {engineState, wingLeftState, wingRightState,
509                     GouverneState, EclairageState, ReserveState, CockpitState};
510     // Compter les LEDs vertes
511     for (int state : states) {
512         if (state == 2) greenLEDs++;
513     }
514     return (greenLEDs * 100.0) / totalLEDs; // Calculer le pourcentage
515 }
516
517 // Fonction pour obtenir le nom de la tâche en fonction de l'index
518 String getTaskName(int index) {
519     String[] tasks = {
520         "Avionniques",
521         "APU",
522         "Moteurs",
523         "Train atterrissages",
524         "Reservoirs",
525         "Ailes"
526     };
527     return tasks[index];
}

```

