

Plan de test
PROJET L2P1
Application ISA

Références du document:		Validé par :	
Version du document:	0.0.1	Validé le :	
Date du document:	09/04/24	Soumis le :	23/04/24
Confidentialité :		UFR Maths-Info / Paris-Cité	

Maîtrise d'ouvrage :	Christophe Gnaho	Chef de projet :	Aymeric Letaconnoux
Date/Signature :		Date/Signature :	<i>Aymeric Letaconnoux</i>

Table des matières

1	Introduction	3
1.1	Documents de références	3
2	Guide de lecture	3
2.1	Maîtrise d'oeuvre	3
2.2	Maîtrise d'ouvrage	4
3	Concepts de base	4
4	Tests fonctionnels	5
4.1	Application mobile	5
4.2	Module d'exploitations des données	18
5	Tests unitaires	26
5.1	Diverses méthodes get() et set(...)	26
5.2	Méthodes communes	27
5.3	Méthodes spécifiques aux classes	34
5.3.1	MainActivity	34
5.3.2	ChoixModeActivity	36
5.3.3	ParametresVolActivity	40
5.3.4	ParametresMesureActivity	44
5.3.5	MesureMultiActivity	53
5.3.6	CompteurChargeMentaleActivity	58
5.3.7	ChoixNomUtiActivity	60
5.3.8	ChargeMentaleSeulActivity	64
5.3.9	InfoVolManager	69
5.3.10	SauvegardeEtEnvoi	71
5.3.11	CodeManager	72
5.4	Tests unitaires du module d'exploitation des données	74
5.4.1	scriptcible.php	74
5.4.2	tableau.php	75
5.4.3	graphes.php	77
6	Tests d'intégration	78
6.1	Tests d'intégration de l'application	79
7	Références et sources	82

1 Introduction

Le plan de test a pour objectif d'organiser les différentes phases de test nécessaires avant la mise en place du produit sur le marché. Le plan de test comporte trois sections majeures : les tests fonctionnels, d'intégration et unitaires.

1.1 Documents de références

Afin de pouvoir comprendre le document, le cahier des charges[1] et le cahier des recettes[2] du produit doivent être connus et lus par les testeurs afin de comprendre les enjeux et objectifs du produit final. Le cahier des charges permet de définir un premier cadre (contraintes liées au produit, fonctionnalités globales, etc.). Le cahier des recettes permet ensuite de définir l'ensemble des fonctionnalités détaillées du produit. Il permet également de définir un certain nombre de scénarios afin de pouvoir concevoir certains tests d'intégration. Le dernier document de référence est la conception détaillée[3] qui permet d'implémenter le produit.

2 Guide de lecture

Afin de pouvoir consulter le document efficacement, le lecteur pourra retrouver les différents tests (fonctionnels, unitaires et d'intégration) dans les différentes sections. Il pourra de plus, pour chaque test, retrouver l'ensemble des conditions nécessaires et la procédure à adopter afin de réaliser les tests de façon efficace.

2.1 Maîtrise d'oeuvre

Le projet sera réalisé par Ramzy Chibani, Vibol Arnaud Sok, Kevin Chen et Aymeric Letaconnoux.

Tous les membres du projet participeront aux tâches de conception et de développement. Pour les tâches de développement, chaque membre travaillera soit de manière individuelle en fonction de la tâche qui lui est attribuée ou en groupe si la tâche demande beaucoup de main d'œuvre.

2.2 Maîtrise d'ouvrage

Le projet est commandé par David Janiszek, directeur de l'UFR Mathématiques et Informatique de l'université Paris-Cité Campus Saint-Germain-des-Prés et Christophe Gnaho, encadrant du projet et professeur à l'université Paris-Cité.

3 Concepts de base

Afin de pouvoir exploiter le document de manière efficace, les notions suivantes devront être comprises et maîtrisées :

- Un *environnement de test*[4] est un espace dédié aux développeurs leur permettant de vérifier le bon fonctionnement du produit en décelant et en corrigeant les problèmes avant sa mise en production.
- Un *test unitaire*[5] consiste à tester une méthode ou fonction individuellement. Ce sont des tests de bas niveau.
- Un *test d'intégration*[6] consiste à tester que les différentes parties du code-source fonctionnent correctement entre elles. Par exemple les interactions entre les différentes *Activités*.
- Un *test fonctionnel*[7] consiste à tester et vérifier qu'une fonctionnalité de l'application ou du module d'exploitation produit les résultats attendus en fonction des spécifications.

4 Tests fonctionnels

4.1 Application mobile

- **Page de démarrage**
 - Identification: MOBILE_DEMARAGE
 - Description: Le but de ce test est de vérifier que lorsque l'utilisateur du téléphone appuie sur le bouton "Démarrer" (voir figure 1) il est redirigé vers la page du choix du mode de l'expérimentation.
 - Contraintes: Il n'y a pas de contrainte liée à ce test.
 - Dépendances: Il n'y a pas de dépendances liées à ce test.
 - Procédure du test: Afin de réaliser le test, le testeur devra suivre les étapes suivantes :
 1. Appuyer sur le bouton "Démarrer".
 2. Vérifier que l'utilisateur est bien redirigé vers la page du choix du mode de l'expérimentation (voir figure 2), le Mode "Vol" ou le mode standard (*ChoixModeActivity*).

Application ISA

Démarrer

**Si vous réalisez
l'expérience dans
un avion,
il est conseillé
d'activer le mode
vol.**

Activer

Refuser



Figure 1: Page de démarrage

Figure 2: Page de choix du mode

- **Choix du mode**

- Identification: MOBILE_MODE
- Description: Le but de ce test est de vérifier que lorsque l'utilisateur du téléphone choisit le mode vol, alors il est redirigé vers la page des paramétrages des informations de vol (voir figure 3) avant la page des paramétrages de l'expérimentation (voir figure 4). S'il ne le choisit pas, alors il est directement redirigé vers la page des paramétrages de l'expérimentation.
- Contraintes: Il n'y a pas de contrainte liée à ce test.
- Dépendances: Le test MOBILE_DEMARAGE devra être effectué avant celui-là pour vérifier l'accès à la page du choix de mode.
- Procédure du test: Afin de réaliser le test, le testeur devra suivre les étapes suivantes :
 1. Appuyer sur le bouton "Activer".
 2. Vérifier que l'utilisateur est bien redirigé vers la page des paramétrages des informations de vol (*ParametresVolActivity*).
 3. Appuyer sur le bouton "Précédent".
 4. Appuyer sur le bouton "Refuser".
 5. Vérifier que l'utilisateur est bien redirigé vers la page des paramétrages de l'expérimentation (*ParametresMesureActivity*).

Durée du vol (heures et minutes)

2	29
3	30
4	31

Aérodrome de départ

Aerodrome
 Charles de Gaulle International Airport

Confirmer

Aérodrome d'arrivée

Aerodrome
 Paris-Orly Airport

Confirmer

Précédent

Suivant

Paramétrages

Intervalle de mesure

minutes	secondes
10	4
0	5
1	6

Échelle de mesure

5

Nombre d'utilisateurs

3

Signal Sonore

Durée de mesure

minutes	secondes
10	59
0	10
1	11

Précédent

Suivant

Figure 3: Paramétrages des informations de vol

Figure 4: Paramétrages des informations de l'expérimentation

- **Choix des noms des utilisateurs**

- Identification: MOBILE_NOM
- Description: Le but de ce test est de vérifier que l'utilisateur puisse bien rentrer le nom de tous les utilisateurs participant à l'expérimentation si cela se déroule avec plusieurs personnes, ou seulement le sien dans le cas contraire. Le nombre d'utilisateurs est défini lors du paramétrage de l'expérimentation (voir figure)3.
- Contraintes: Il n'y a pas de contrainte liée à ce test.
- Dépendances: Le test MOBILE_MODE devra être effectué avant celui-là pour vérifier l'accès à la page du choix des noms des utilisateurs.
- Procédure du test: Afin de réaliser le test, le testeur devra suivre les étapes suivantes :
 1. Commencer à écrire les noms des utilisateurs dans le champ de saisie (voir figure 5)..
 2. Les ajouter grâce au bouton "Ajouter".
 3. Essayer le bouton "Précédent" pour pouvoir changer les noms déjà émis.
 4. Lorsque tous les noms ont été définis, observer si un pop-up apparaît à l'écran (voir figure 6) afin de confirmer ces choix, si c'est le cas alors la fonctionnalité est respectée.

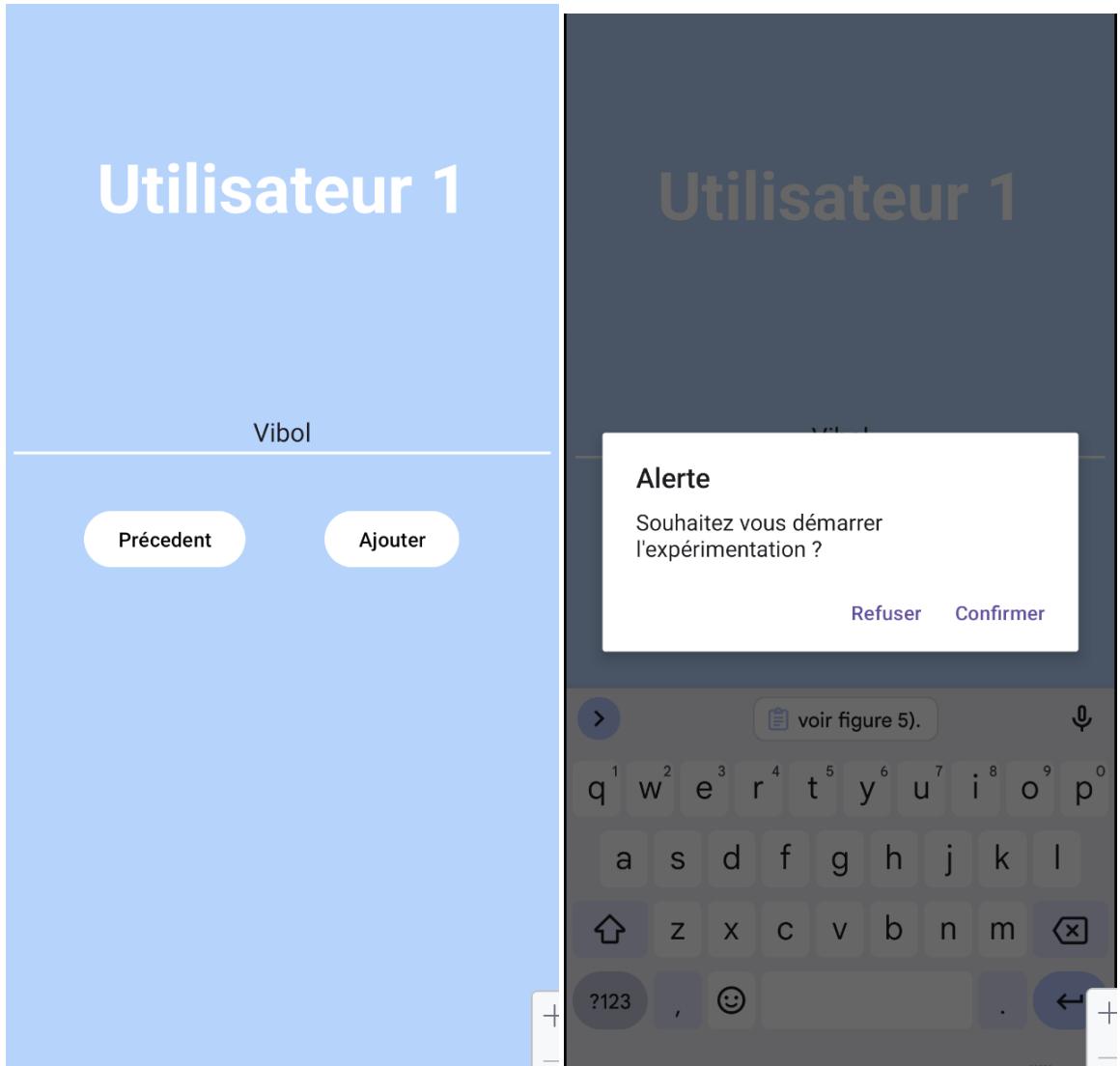


Figure 5: Page du choix des noms d'utilisateurs

Figure 6: Affichage du pop-up de confirmation

- **Conditions d'expérimentation**

- Identification: MOBILE_EXPER
- Description: Le but de ce test est de vérifier que l'expérimentation est bien en accord avec l'ensemble des paramètres définis sur la page des paramétrages (voir figure4) et que les noms des utilisateurs sont bien ceux définis sur la page associée (voir figure5).
- Contraintes: Il n'y a pas de contrainte liée à ce test.
- Dépendances: Le test MOBILE_NOM devra être effectué avant celui-là pour vérifier l'accès à l'écran d'expérimentation.
- Procédure du test: Afin de réaliser le test, le testeur devra suivre les étapes suivantes :
 1. Si le signal sonore a été activé, vérifier que l'écran de signal est bien celui associé (voir figure 7 et 8). (*Ce sont les activités RedSonActivity et SonActivity*).
 2. Vérifier que si l'expérimentation se fait à plusieurs, alors l'utilisateur devra mesurer autant d'utilisateurs que le nombre défini lors de la phase de paramétrage. Il faudra en plus s'assurer que les noms affichés en haut de l'écran correspondent bien à ceux définis précédemment. (*Ce sont les activités ChargeMentaleSeulActivity et MesureMultiActivity*).
 3. Vérifier que la durée de mesure est bien celle paramétrée sur l'écran de paramétrage (voir figure 4).
 4. Vérifier que l'écran du compteur (voir figure 9) qui sera affiché lorsque que toutes les mesures auront été effectuées (une mesure si l'utilisateur est seul ou 1 cycle s'il y a plusieurs utilisateurs) respecte bien la durée définie précédemment. (*C'est l'activité CompteurChargeMentaleActivity*).

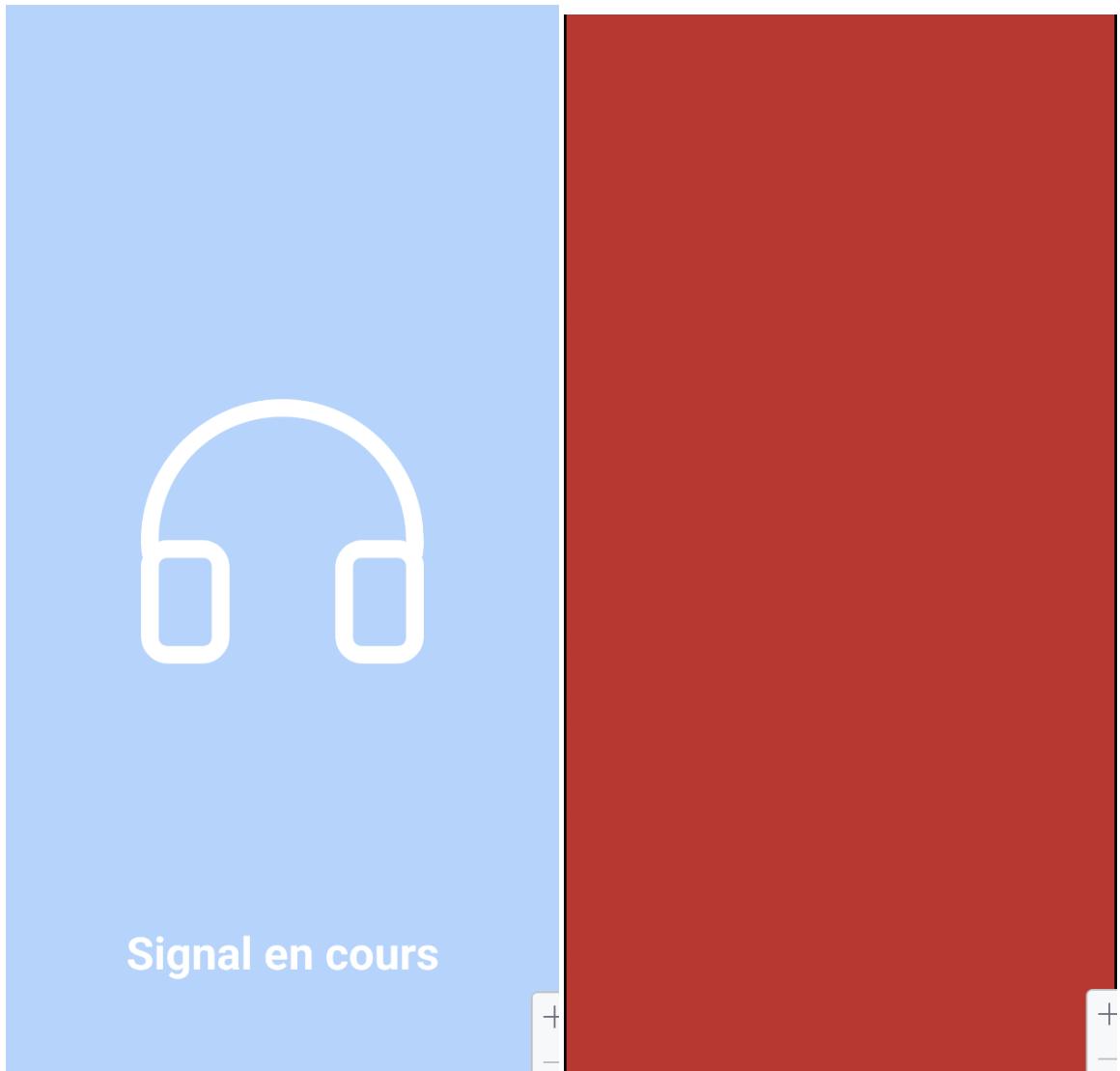


Figure 7: Écran du signal sonore

Figure 8: Écran du signal visuel

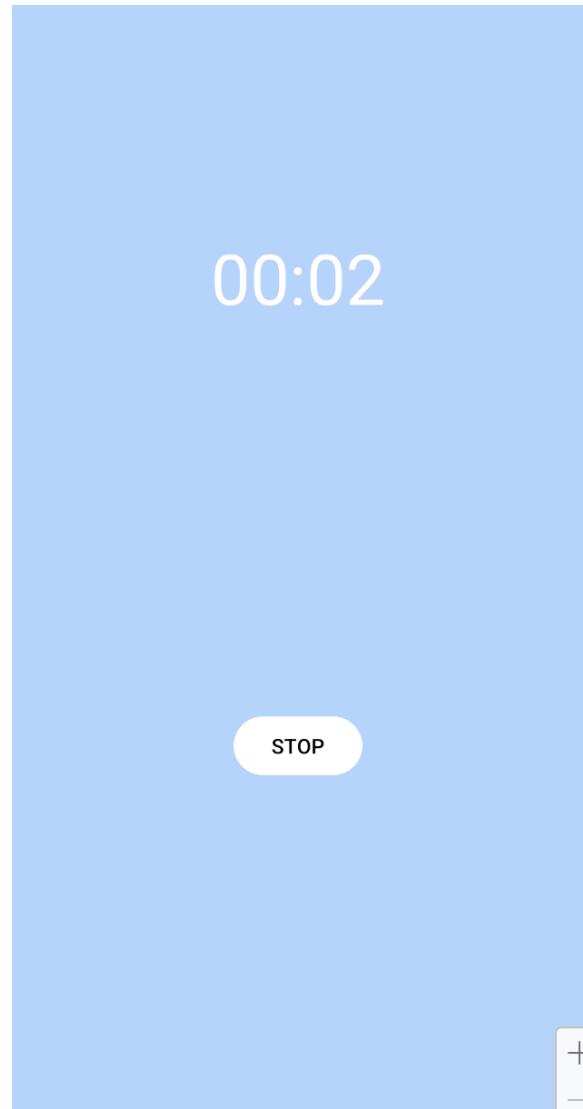


Figure 9: Écran du compteur

- **Cohérences des mesures de l'expérimentation dans l'application**

- Identification: MOBILE_MASUREXPER
- Description: Le but de ce test est de vérifier que les mesures effectuées lors de l'expérimentation sont bien cohérentes avec les mesures récupérées et stockées par la suite avant de les transmettre au module d'exploitation des données.
- Contraintes: Afin de réaliser ce test, il est nécessaire de disposer d'un téléphone Android (8.1 oreo au minimum) avec une entrée son afin de tester la mesure par la voix.
- Dépendances: Le test MOBILE_EXPER devra être effectué avant celui-là pour vérifier que l'utilisateur puisse bien réaliser l'expérimentation dans de bonnes conditions.
- Procédure du test: Afin de réaliser le test, le testeur devra suivre les étapes suivantes :
 1. Dans *CompteurChargeMentaleActivity*, remplacer la ligne suivante:
Intent choixActivite = new Intent(getApplicationContext(), CodeActivity.class)
par celle ci:
Intent choixActivite = new Intent(getApplicationContext(), MesureTestMultiActivity.class)
 2. Réaliser une expérimentation à plusieurs utilisateurs.
 3. Après avoir effectué un nombre de mesures suffisant (3 cycles par exemple), appuyer sur le bouton "Stop" de l'écran du compteur (voir figure 10) et confirmer l'arrêt de l'expérimentation.
 4. Vérifier si les valeurs des mesures correspondent bien à celle montrées à l'écran.

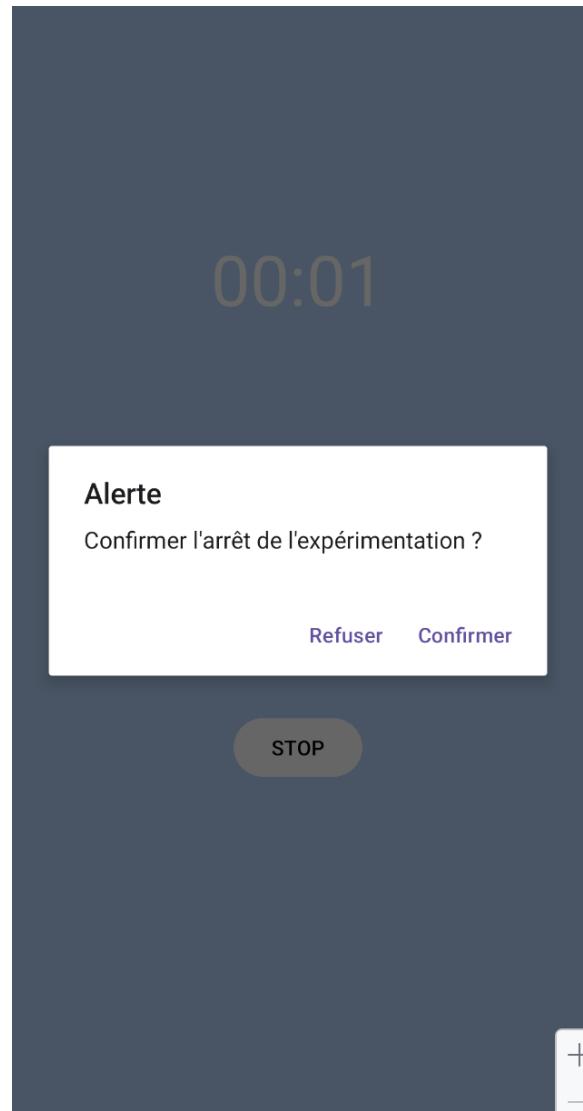


Figure 10: Pop-up affiché à l'écran après avoir appuyé sur le bouton "Stop"

- **Affichage du code d'accès**

- Identification: MOBILE_CODE
- Description: Le but de ce test est de vérifier qu'un code d'accès est bien affiché à l'écran quand l'utilisateur décide d'arrêter l'expérimentation. Ce code permettra à l'utilisateur d'accès au module d'exploitation des données.
- Contraintes: Il n'y a pas de contrainte liée à ce test.
- Dépendances: Le test MOBILE_MESUREXPER devra être effectué avant celui-là pour vérifier que les données qui seront envoyées et récupérées grâce au code d'accès sont bien celles de l'expérimentation.
- Procédure du test: Afin de réaliser le test, le testeur devra suivre les étapes suivantes :
 1. Cliquer sur un le bouton "STOP" de l'écran sur le compteur ou sur l'écran de mesure. (voir figure 10 et figure 11).
 2. Confirmer l'arrêt de l'expérimentation.
 3. Observer si l'écran du code d'accès est bien afficher à la fin.

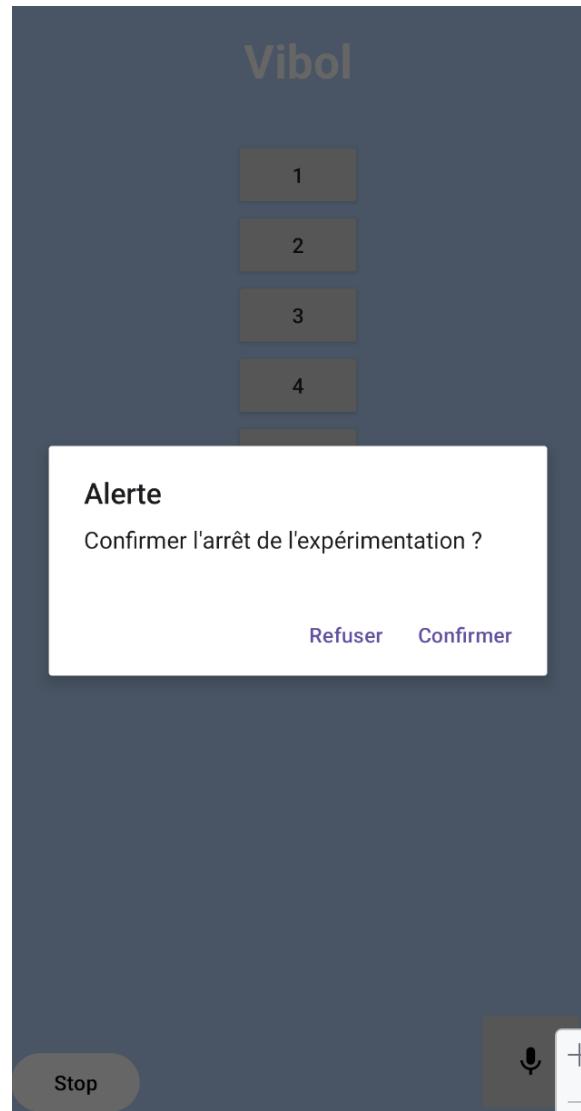


Figure 11: Pop-up affiché à l'écran après avoir appuyé sur le bouton "Stop"

4.2 Module d'exploitations des données

- **Page d'accueil**

- Identification: MODULE_ACCUEIL
- Description: Le but de ce test est de vérifier que lorsque l'utilisateur saisit un code et le valide (figure 12), alors il est redirigé vers la page des informations du vol (figure 13), sinon une erreur s'affiche.
- Contraintes: Il n'y a pas de contrainte liée à ce test.
- Dépendances: Les tests liés à la page cible doivent être concluants afin de garantir une bonne redirection.
- Procédure du test: Afin de réaliser le test, le testeur devra suivre les étapes suivantes :
 1. Ouvrir le module d'exploitation sur un navigateur dont l'url est : <https://aymeric1.fr/>.
 2. Vérifier que lorsque l'utilisateur saisit et valide un code valide alors il accède bien au module. Il pourra utiliser le code *v12341234* qui est un code de test en libre-service.

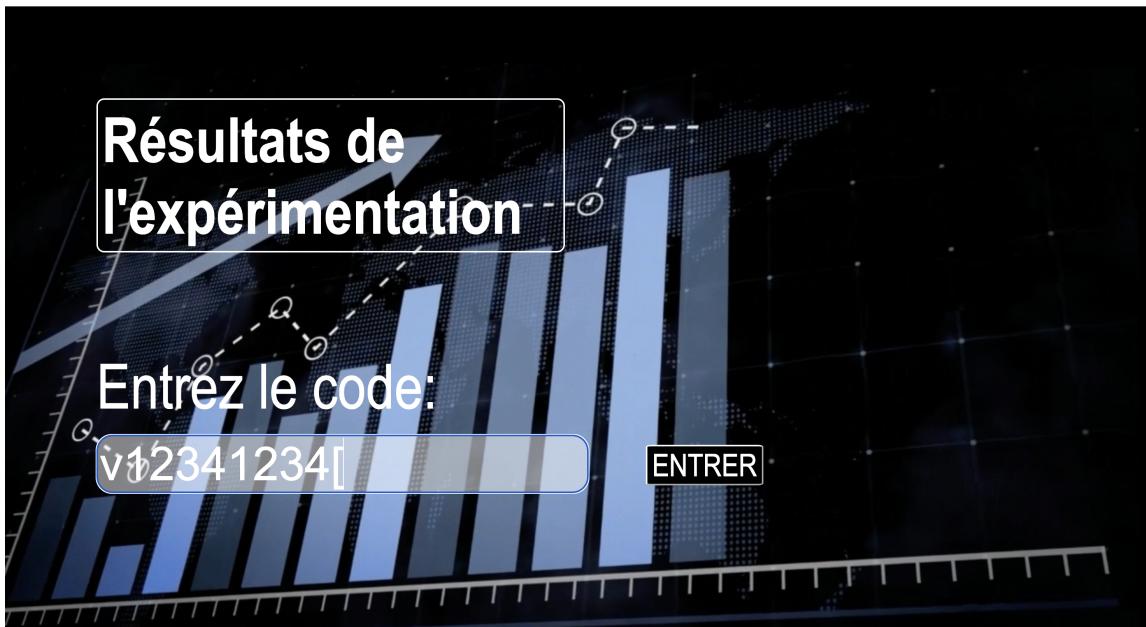


Figure 12: Page d'accueil

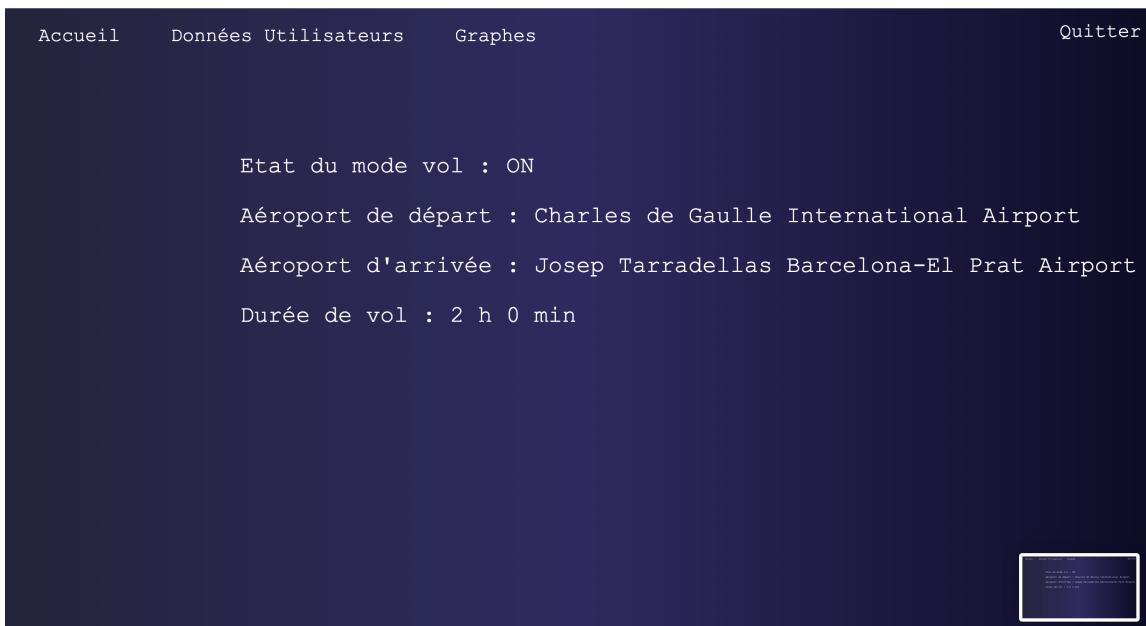


Figure 13: Informations du vol

- **Cohérence des données**

- Identification: MODULE_DONNEES
- Description: Le but de ce test est de vérifier que les données affichées sur le module correspondent au fichier texte sélectionné avec le code saisi sur la page d'accueil.
- Contraintes: Il n'y a pas de contrainte liée à ce test.
- Dépendances: Le test *MODULE_ACCUEIL* doit être concluant afin de garantir que le code permet l'accès au module d'exploitation.
- Procédure du test: Afin de réaliser le test, le testeur devra suivre les étapes suivantes :
 1. Ouvrir le module d'exploitation sur un navigateur dont l'url est : <https://aymeric1.fr/>.
 2. Vérifier que lorsque l'utilisateur saisit et valide un code valide alors il accède bien au module. Il pourra utiliser le code *v12341234* qui est un code de test en libre-service.
 3. Cliquer sur les rubriques *Données Utilisateurs* et *Graphes*.(figure 14 et figure 15)
 4. Vérifier que les valeurs affichées correspondent bien aux valeurs présentes dans le fichier *v12341234.txt*.

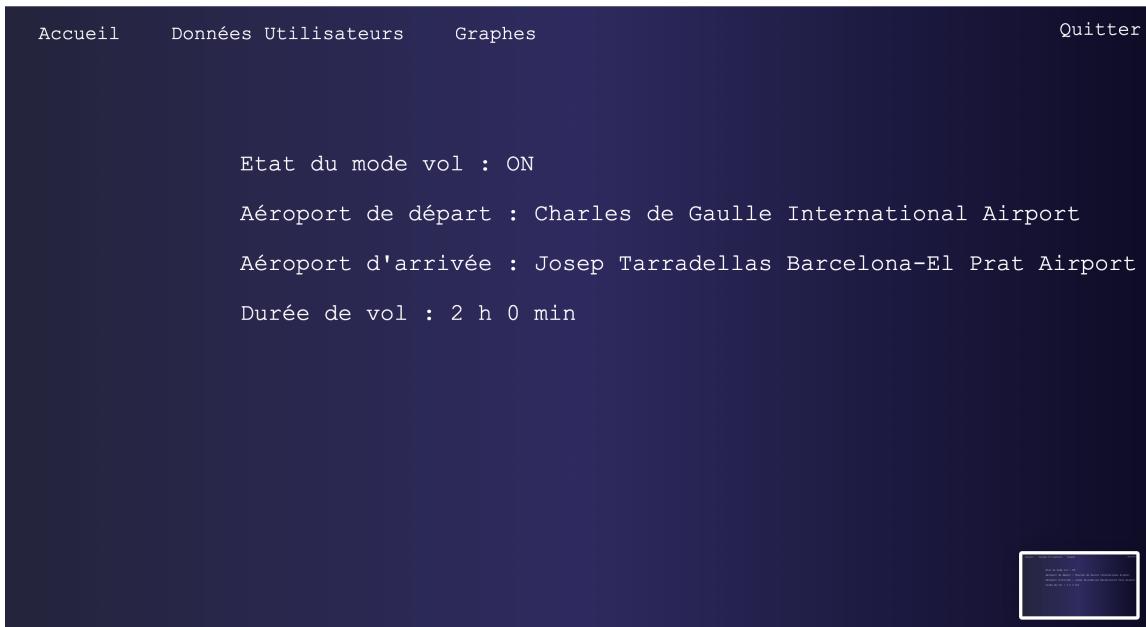


Figure 14: Rubrique *Données Utilisateurs*

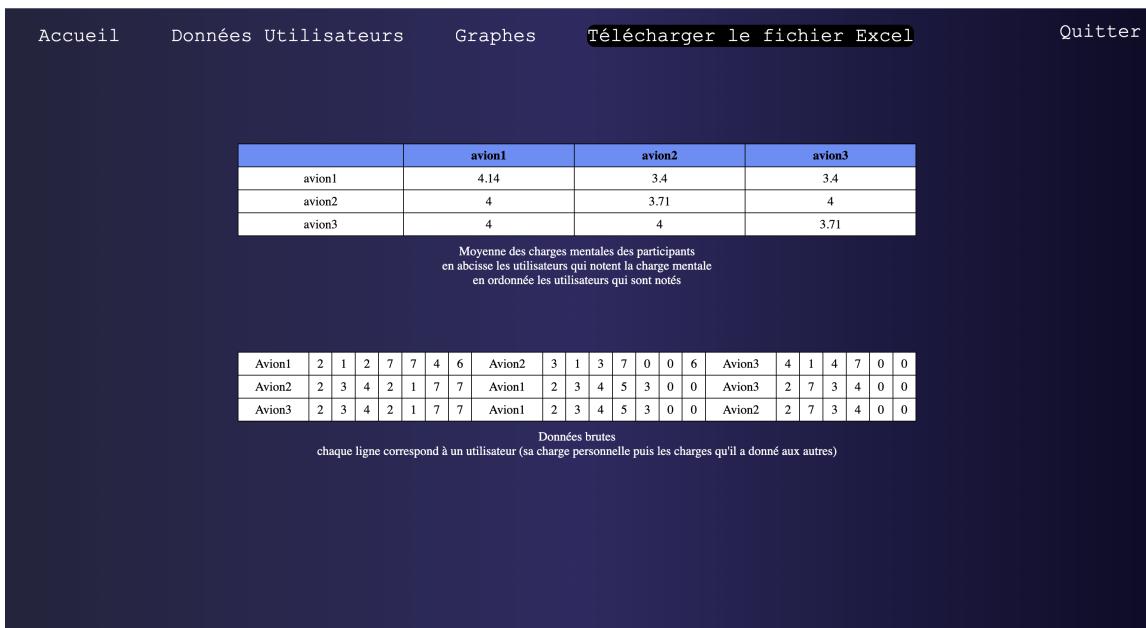


Figure 15: Rubrique *Graphes*

- **Quitter le module**

- Identification: MODULE_QUITTER
- Description: Le but de ce test est de vérifier que lorsque l'utilisateur clique sur le bouton *QUITTER*, alors il est directement redirigé vers la page d'accueil.
- Contraintes: Il n'y a pas de contrainte liée à ce test.
- Dépendances: Le test *MODULE_ACCUEIL* doit être concluant afin de garantir que le code permet l'accès au module d'exploitation.
- Procédure du test: Afin de réaliser le test, le testeur devra suivre les étapes suivantes :
 1. Cliquer sur le bouton *QUITTER* situé en haut à droite.
 2. Lorsque la boîte de dialogue s'affiche (figure 16), cliquer sur *OK*.
 3. Vérifier que l'utilisateur est bien redirigé vers la page d'accueil.
 4. Réitérer le test en appuyant sur le bouton *QUITTER* depuis toutes les rubriques (*Accueil, Données Utilisateurs, etc.*).

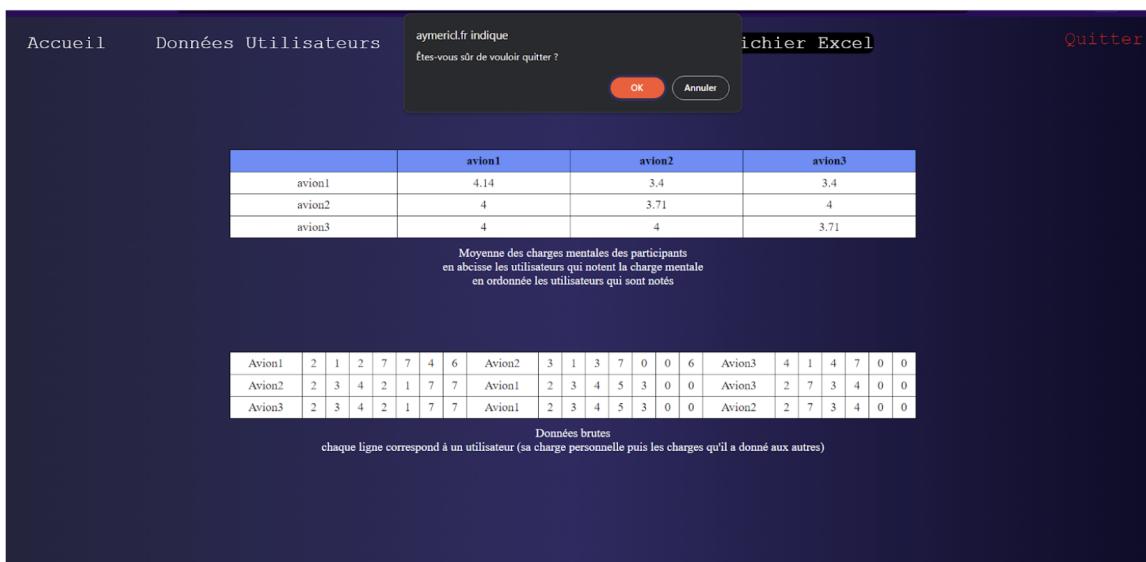


Figure 16: Boîte de dialogue sur la page *Données Utilisateurs*

- **Téléchargement du fichier format EXCEL**

- Identification: MODULE_EXCEL
- Description: Le but de ce test est de vérifier que lorsque l'utilisateur clique sur le bouton “*Télécharger le fichier Excel*”, alors le téléchargement des données en format *excel* (.xlsx) se lance.
- Contraintes: Posséder un logiciel capable d'ouvrir les fichiers au format xlsx.
- Dépendances: Le test *MODULE_DONNEES* doit être concluant afin de la cohérence des données avant qu'elles soient téléchargées.
- Procédure du test: Afin de réaliser le test, le testeur devra suivre les étapes suivantes :
 1. Cliquer sur le bouton “*Télécharger le fichier Excel*”.
 2. Après la fin du téléchargement du fichier, ouvrir le fichier en double-cliquant dessus.
 3. Vérifier la cohérence des données (figure 17 et 18) par rapport à celles présentes sur le module d'exploitation.

	A	B	C	D	E
1	avion1	avion1			
2	avion1	4,14	3,4	3,4	
3	avion2	4	3,71	4	
4	avion3	4	4	3,71	
5					
6					
7					
8					
9					
10					
11					
12					
13					
14					
15					
16					
17					
18					
19					
20					
21					
22					
23					

Figure 17: Données sous format xlsx 1/2

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
1	Avion1	2	1	2	7	4	6	Avion2	3	1	3	7	0	0	6	Avion3	4	1	4	7	0	0	
2	Avion2	2	3	4	2	1	7	7	Avion1	2	3	4	5	3	0	0	Avion3	2	7	3	4	0	0
3	Avion3	2	3	4	2	1	7	7	Avion1	2	3	4	5	3	0	0	Avion2	2	7	3	4	0	0
4																							
5																							
6																							
7																							
8																							
9																							
10																							
11																							
12																							
13																							
14																							
15																							
16																							
17																							
18																							
19																							
20																							
21																							
22																							
23																							

Figure 18: Données sous format xlsx 2/2

5 Tests unitaires

Du côté de l'application mobile, les tests des méthodes seront regroupés selon les classes où elles se trouvent. Des méthodes récurrentes communes à de nombreuses classes seront regroupés dans la section *Méthodes communes* afin de ne pas surcharger le document et de ne pas réaliser des tests répétitifs.

5.1 Diverses méthodes get() et set(...)

Certaines classes (*la majorité les classes se terminant par Manager*) possèdent des méthodes get et set (*mutateurs et assesseurs*), ces méthodes réalisent toutes la même chose, elles assignent, pour les sets, une valeur à un attribut de classe ou bien, pour les gets, elles renvoient un attribut de classe.

Les tests sont alors les mêmes:

La procédure est la suivante :

1. Créer une instance de la classe.
2. Pour les méthodes get, regarder si la valeur retournée est bien celle de l'attribut de classe.
3. Pour les méthodes set, il faut vérifier que l'attribut de classe a bien la valeur passée en paramètre du set.

Cette vérification peut se faire avec un get par exemple ou une méthode affiche() qui affichera dans la console les différents attributs de la classe.

5.2 Méthodes communes

- *private void activiteSuivante()*
 - Identifiant: COMMUNE_ACTIVITESUIVANTE
 - Description: Ce test vise à vérifier que lorsque la méthode est appelée, alors elle permet la redirection vers l'activité voulue grâce au clic effectué sur le bouton dédié et stock de manière définitive le choix de l'utilisateur. La méthode est commune à la classe *ParametresVolActivity* et *ParametresMesureActivity*.
L'environnement de test est le suivant:
 - * Le matériel nécessaire est un téléphone Android muni de la version 8.1 au minimum ou une machine virtuelle Android.
 - Contraintes: L'existence d'un bouton afin de pouvoir appeler la méthode *setOnClickListener*, et l'existence d'une activité cible pour créer l'objet *Intent*.
Les classes associées aux objets nécessaires au fonctionnement de la méthode doivent également être importées.
 - Dépendances: Il n'y a pas de dépendances associées à cette méthode.
 - Procédure du test: Afin de réaliser le test, le testeur devra suivre les étapes suivantes :
 1. Créer un objet **boutonTest** de classe *Button*.
 2. Créer une activité **arrivee**.
 3. Créer un objet **local** d'une des classes Manager selon le besoin.
 4. Créer un objet **data** de classe *DataManager*.
 5. Placer le bouton créé précédemment dans la méthode comme cela:
boutonTest.setOnClickListener(new View.OnClickListener(){...})
 6. Placer l'activité créée précédemment avec *.class* à la fin comme cela:
Intent nextActivity = new Intent(getApplicationContext(), arrivee.class);
 7. Placer l'objet **choixMode** créé précédemment dans la méthode souhaitée comme cela:
local.setNomMethode(...);

8. Placer l'objet ***data*** crée précédemment dans la méthode comme cela:
data.setNomMethodeFinal(...);
9. Compiler, exécuter puis cliquer sur le bouton et vérifier si le testeur est bien redirigé vers l'activité ***Arrivee***.
10. Dans l'activité ***Arrivee***, réitirer l'étape 3. et 4. et afficher la valeur.

- *private void activitePrecedante()*
 - Identifiant: COMMUNE_ACTIVITEPRECEDANTE
 - Description: Ce test vise à vérifier que lorsque la méthode est appelée, alors elle permet la redirection vers l'activité précédente (*ChoixModeActivity*) grâce au clic effectué sur le bouton dédié.
La méthode est commune à la classe *ParametresVolActivity* et *ParametresMesureActivity*.
L'environnement de test est le suivant:
 - * Le matériel nécessaire est un téléphone Android muni de la version 8.1 au minimum ou une machine virtuelle Android.
 - Contraintes: L'existence d'un bouton afin de pouvoir appeler la méthode *setOnClickListener*, et l'existence d'une activité cible pour créer l'objet *Intent*.
Les classes associées aux objets nécessaires au fonctionnement de la méthode doivent également être importées.
 - Dépendances: Il n'y a pas de dépendances associées à cette méthode.
 - Procédure du test: Afin de réaliser le test, le testeur devra suivre les étapes suivantes :
 1. Créer un objet **boutonTest** de classe *Button*.
 2. Créer une activité **retour**.
 3. Placer le bouton créé précédemment dans la méthode comme cela:
`boutonTest.setOnClickListener(new View.OnClickListener(){...})`
 4. Placer l'activité créée précédemment avec *.class* à la fin comme cela:
`Intent nextActivity = new Intent(getApplicationContext(), retour.class);`
 5. Compiler, exécuter puis cliquer sur le bouton et vérifier si le testeur est bien redirigé vers l'activité **retour**.

- *private void passerALaProchaineActivite()*
 - Identifiant: COMMUNE_PASSER
 - Description: Ce test vise à vérifier que lorsque la méthode est appelée alors, le testeur est redirigé selon une des deux activités de mesure. Cette méthode est présente dans les classes *RedActivity* et *SonActivity*.
L'environnement de test est le suivant:
 - * Le matériel nécessaire est un téléphone Android muni de la version 8.1 au minimum ou une machine virtuelle Android.
 - Contraintes: Il n'y a aucune contrainte.
Les classes associées aux objets nécessaires au fonctionnement de la méthode doivent également être importées.
 - Dépendances: Les méthodes de la classe *UserManager* devront être testées.
 - Procédure du test: Afin de réaliser le test, le testeur devra suivre les étapes suivantes :
 1. Créer deux objets **cible1** et **cible2** de classe *Intent*.
 2. Appeler les méthodes de *UserManager* comme ceci:
UserManager.getInstance().getUserNames().size().
 3. Vérifier que la redirection est bien celle qui est attendue.

- *private void boutonStop()*
 - Identifiant: COMMUNE_BOUTON_STOP
 - Description: Ce test vise à vérifier que lorsqu'un utilisateur appuie sur le bouton stop de l'activité, une fenêtre de dialogue fonctionnelle invitant l'utilisateur à confirmer son choix de quitter l'expérimentation apparaît. Ici la confirmation de quitter l'expérimentation a pour conséquence de renvoyer l'utilisateur vers une dernière activité (*CodeActivity*) qui affichera un code pour se connecter sur le module d'expérimentation.
 - La méthode est commune aux classes *ChargeMentaleSeulActivity*, *Mesure-MultiActivity* et *CompteurChargeMentaleActivity*.

L'environnement de test est le suivant:

- * Le matériel nécessaire est un téléphone Android muni de la version 8.1 au minimum ou une machine virtuelle Android.
- Contraintes: Il n'y a aucune contrainte pour ce test.
- Dépendances: Il n'y a aucune dépendance pour ce test.
- Procédure du test: Afin de réaliser le test, le testeur devra suivre les étapes suivantes :
 1. Créer les attributs suivants :
 - * Un objet **boutonStop** de type *Button*.
 - * Un objet **mCountDownTimer** de type *CountDownTimer*.
 - * Un objet **builder** de type *AlertDialog.Builder*.
 2. Créer une activité **test** qui devra apparaître lorsque l'on confirme l'arrêt de l'expérimentation.
 3. Placer les attributs dans la méthode ainsi que l'activité test de cette manière :
 - **bouton_stop.setOnClickListener(new View.OnClickListener())**;
 - **mCountDownTimer.cancel();**
 - **builder.setTitle("Alerte")**
 4. Lancer l'application et appuyer sur le bouton stop, appuyer sur refuser pour confirmer que l'expérimentation ne s'arrête pas dans ce cas, puis re-appuyer sur le bouton stop et appuyer sur confirmer, si l'utilisateur est renvoyé vers l'activité test alors le test est concluant.

- *private void boutonVoix()*
 - Identifiant: COMMUNE_BOUTON_VOIX
 - Description: Ce test vise à vérifier que lorsqu'un utilisateur appuie sur le bouton micro dans l'activité, l'utilisateur peut donner une valeur par la voix, et cette valeur sera enregistrée.
La méthode est commune à la classe *ChargeMentaleSeulActivity* et *Mesure-MultiActivity*.
- L'environnement de test est le suivant:
- * Le matériel nécessaire est un téléphone Android muni de la version 8.1 au minimum ou une machine virtuelle Android. Le téléphone ou la machine virtuelle devra disposer d'une entrée audio.
 - Contraintes: Le testeur devra énoncer certaines valeurs à l'oral pour vérifier le fonctionnement de la méthode.
 - Dépendances: Pour ce test nous utiliserons la méthode *reconnaissanceVocale()* qui nous servira à interpréter les données reçues par le micro. Il faudra également vérifier le fonctionnement des méthodes *sstartListening* et *stopListening*.
 - Procédure du test: Afin de réaliser le test, le testeur devra suivre les étapes suivantes :
1. créer les attributs suivants :
 - * Un objet ***boutonMicro*** de type *Button*.
 - * Un objet ***vocalAction*** de type *Intent*.
 - * Un objet ***reco Vocale*** de type *SpeechRecognizer*.
 - * Un objet ***text*** de type *TextView*.
 2. Placer les trois premiers attributs dans la méthode *boutonVoix()*.
 3. Remplacer dans la méthode *reconnaissanceVocale()* la ligne *startActivity();* par ***text.setText(valeurChargeMentale);***.
 4. Lancer l'application et appuyer sur le bouton micro, donner une valeur de cette manière "valeur un" si un "un" s'affiche dans l'activité le test est concluant.

- *public void getInstance()*
 - Identifiant: SINGLETON_GET_INSTANCE
 - Description: Ce test permet de vérifier qu'une seule instance de *UserManager* (resp. *DataManager*) peut-être créée, la classe étant un singleton. Cette méthode est commune à la classe *DataManager* et *UserManager*. L'environnement de test est le suivant:
 - * Le matériel nécessaire est un téléphone Android muni de la version 8.1 au minimum ou une machine virtuelle Android.
 - Contraintes: Il n'y a pas de contrainte pour tester cette méthode.
 - Dépendances: Les méthodes de la classe *Utilisateur* et de toutes les classes se terminant par *Manager* doivent être testées.
 - Procédure du test: Afin de réaliser le test, le testeur devra suivre les étapes suivantes :
 1. Tenter de créer une instance de *UserManager* (resp. *DataManager*) en appelant le constructeur avec *new UserManager()*. (resp. *DataManager*)
 2. Compiler le code.
 3. Si le code ne compile pas alors la classe est bien un singleton.
 - Maintenant il faut vérifier qu'il est possible d'assigner des valeurs à l'unique instance de *UserManager* (resp. *DataManager*) :
 1. Appeler l'instance de *UserManager* (resp. *DataManager*) avec la méthode *getInstance()*.
 2. Initialiser les attributs de la classe.
 3. Appeler une nouvelle fois la méthode *getInstance()*.
 4. À l'aide des méthodes get de la classe, vérifier que les attributs sont les mêmes qu'au départ.

5.3 Méthodes spécifiques aux classes

5.3.1 MainActivity

- *private void demarrer()*
 - Identifiant: MAINACTIVITY_DEMARRER
 - Description: Ce test vise à vérifier que lorsque la méthode est appelée, alors elle permet la redirection vers l'activité voulue grâce au clic effectué sur le bouton dédié.
L'environnement de test est le suivant:
 - * Le matériel nécessaire est un téléphone Android muni de la version 8.1 au minimum ou une machine virtuelle Android.
 - Contraintes: L'existence d'un bouton afin de pouvoir appeler la méthode *setOnClickListener*, et l'existence d'une activité cible pour créer l'objet *Intent*.
Les classes associées aux objets nécessaires au fonctionnement de la méthode doivent également être importées.
 - Dépendances: Il n'y a pas de dépendances associées à cette méthode.
 - Procédure du test: Afin de réaliser le test, le testeur devra suivre les étapes suivantes :
 1. Créer un objet **boutonTest** de classe *Button*.
 2. Créer une activité **arrivee**.
 3. Placer le bouton créé précédemment dans la méthode comme cela:
boutonTest.setOnClickListener(new View.OnClickListener()...)
 4. Placer l'activité créée précédemment avec *.class* à la fin comme cela:
Intent nextActivity = new Intent(getApplicationContext(), arrivee.class);
 5. Compiler, exécuter puis cliquer sur le bouton et vérifier si le testeur est bien redirigé vers l'activité **arrivee**.

- `public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[] grantResults)`

– Identifiant: MAINACTIVITY_ONREQUEST

– Description: Ce test vise à vérifier que lorsque la méthode est appelée, un pop-up demandant à l'utilisateur les droits liés à la reconnaissance vocale est affichée l'écran.

L'environnement de test est le suivant:

* Le matériel nécessaire est un téléphone Android muni de la version 8.1 au minimum avec une entrée audio ou une machine virtuelle Android compatible avec une entrée audio.

– Contraintes: *Les classes associées aux objets nécessaires au fonctionnement de la méthode doivent également être importées.*

– Dépendances: Il n'y a pas de dépendances associées à cette méthode.

– Procédure du test: Afin de réaliser le test, le testeur devra suivre les étapes suivantes :

1. Lancer l'application.
2. Appuyer sur "Ne pas autoriser".
3. Vérifier que le micro n'est pas activé dès lors que l'application est lancée puis fermer l'application.
4. Relancer l'application puis appuyer sur "Uniquement cette fois-ci".
5. Vérifier que le micro est activé et ne l'est plus lorsque l'application est relancé.
6. Appuyer sur "Lorsque vous utilisez l'appli".
7. Vérifier que le micro est activé dès lors que l'application est lancée.

5.3.2 ChoixModeActivity

- *private void activiteModeVol()*
 - Identifiant: CHOIXMODEACTIVITY_ACTIVITEVOL
 - Description: Ce test vise à vérifier que lorsque la méthode est appelée, alors elle permet la redirection vers l'activité voulue grâce au clic effectué sur le bouton dédié et stock de manière définitive le choix de l'utilisateur. L'environnement de test est le suivant:
 - * Le matériel nécessaire est un téléphone Android muni de la version 8.1 au minimum ou une machine virtuelle Android.
 - Contraintes: L'existence d'un bouton afin de pouvoir appeler la méthode *setOnClickListener*, et l'existence d'une activité cible pour créer l'objet *Intent*.
Les classes associées aux objets nécessaires au fonctionnement de la méthode doivent également être importées.
 - Dépendances: Les méthodes de la classe *DataManager* et de la classe *ChoixModeManager* devront être testées et validées pour veiller au bon fonctionnement de cette méthode.
 - Procédure du test: Afin de réaliser le test, le testeur devra suivre les étapes suivantes :
 1. Créer un objet **boutonTest** de classe *Button*.
 2. Créer une activité **arrivee**.
 3. Créer un objet **choixMode** de classe *ChoixModeManager*.
 4. Créer un objet **data** de classe *DataManager*.
 5. Placer le bouton créé précédemment dans la méthode comme cela:
boutonTest.setOnClickListener(new View.OnClickListener()...
 6. Placer l'activité créée précédemment avec *.class* à la fin comme cela:
Intent nextActivity = new Intent(getApplicationContext(), arrivee.class);
 7. Placer l'objet **choixMode** créé précédemment dans la méthode comme cela:
choixMode.setChoixMode(true);

8. Placer l'objet ***data*** crée précédemment dans la méthode comme cela:
data.setChoixModeFinal(choixMode);
9. Compiler, exécuter puis cliquer sur le bouton et vérifier si le testeur est bien redirigé vers l'activité ***Arrivee***.
10. Dans l'activité ***Arrivee***, réitirer l'étape 3. et 4. et afficher la valeur.
Dans notre test, la valeur doit être "true".

- *private void activiteParaMesure()*
 - Identifiant: CHOIXMODEACTIVITY_ACTIVITEPARAMESURE
 - Description: Ce test vise à vérifier que lorsque la méthode est appelée, alors elle permet la redirection vers l'activité voulue grâce au clic effectué sur le bouton dédié et stock de manière définitive le choix de l'utilisateur. L'environnement de test est le suivant:
 - * Le matériel nécessaire est un téléphone Android muni de la version 8.1 au minimum ou une machine virtuelle Android.
 - Contraintes: L'existence d'un bouton afin de pouvoir appeler la méthode *setOnClickListener*, et l'existence d'une activité cible pour créer l'objet *Intent*.
Les classes associées aux objets nécessaires au fonctionnement de la méthode doivent également être importées.
 - Dépendances: Les méthodes de la classe *DataManager* et de la classe *ChoixModeManager* devront être testées et validées pour veiller au bon fonctionnement de cette méthode.
 - Procédure du test: Afin de réaliser le test, le testeur devra suivre les étapes suivantes :
 1. Créer un objet **boutonTest** de classe *Button*.
 2. Créer une activité *arrivee*.
 3. Créer un objet **choixMode** de classe *ChoixModeManager*.
 4. Créer un objet **data** de classe *DataManager*.
 5. Placer le bouton créé précédemment dans la méthode comme cela:
`boutonTest.setOnClickListener(new View.OnClickListener(){...})`
 6. Placer l'activité créée précédemment avec *.class* à la fin comme cela:
`Intent nextActivity = new Intent(getApplicationContext(), Arrivee.class);`
 7. Placer l'objet **choixMode** créé précédemment dans la méthode comme cela:
`choixMode.setChoixMode(false);`

8. Placer l'objet ***data*** crée précédemment dans la méthode comme cela:
data.setChoixModeFinal(choixMode);
9. Compiler, exécuter puis cliquer sur le bouton et vérifier si le testeur est bien redirigé vers l'activité ***arrivee***.
10. Dans l'activité ***Arrivee***, réitirer l'étape 3. et 4. et afficher la valeur.
Dans notre test, la valeur doit être "false".

5.3.3 ParametresVolActivity

- *private void dureeHeures()*

– Identifiant: PARAMETRESVOL_DUREEHEURES

– Description: Ce test vise à vérifier que lorsque la méthode est appelée, la durée en heures des informations liées au vol sont bien enregistrées dans l'attribut de classe *InfoVolManager*.

L'environnement de test est le suivant:

* Le matériel nécessaire est un téléphone Android muni de la version 8.1 au minimum ou une machine virtuelle Android.

– Contraintes: L'existence d'une roulette (*NumberPicker*) afin de pouvoir appeler la méthode *setOnValueChangeListener*.

Les classes associées aux objets nécessaires au fonctionnement de la méthode doivent également être importées.

– Dépendances: Les méthodes de la classe *InfoVolManager* devront être testées et validées pour veiller au bon fonctionnement de cette méthode.

– Procédure du test: Afin de réaliser le test, le testeur devra suivre les étapes suivantes :

1. Créer un objet **pickerTest** de classe *NumberPicker*.
2. Créer un objet **info** de classe *InfoVolManager*.
3. Placer la roue créée précédemment dans la méthode comme cela:
pickerTest.setOnValueChangeListener(new NumberPicker.OnValueChangeListener(){...})
4. Placer l'objet **info** créé précédemment dans la méthode comme cela:
info.setMinutes(newVal, oldVal);
5. Interagir avec la roue et la placer sur une valeur définie au hasard.
6. Afficher la valeur stockée dans *info* grâce à la méthode *getHeures* comme ceci : **info.getHeures()**.

- *private void dureeMinutes()*

- Identifiant: PARAMETRESVOL_DUREEMINUTES
- Description: Ce test vise à vérifier que lorsque la méthode est appelée, la durée en minutes des informations liées au vol sont bien enregistrées dans l'attribut de classe *InfoVolManager*.
L'environnement de test est le suivant:
 - * Le matériel nécessaire est un téléphone Android muni de la version 8.1 au minimum ou une machine virtuelle Android.
- Contraintes: L'existence d'un bouton (*NumberPicker*) afin de pouvoir appeler la méthode *setOnValueChangeListener*.
Les classes associées aux objets nécessaires au fonctionnement de la méthode doivent également être importées.
- Dépendances: Les méthodes de la classe *InfoVolManager* devront être testées et validées pour veiller au bon fonctionnement de cette méthode.
- Procédure du test: Afin de réaliser le test, le testeur devra suivre les étapes suivantes :
 1. Créer un objet ***pickerTest*** de classe *NumberPicker*.
 2. Créer un objet ***info*** de classe *InfoVolManager*.
 3. Placer la roue créée précédemment dans la méthode comme cela:
`pickerTest.setOnValueChangeListener(new NumberPicker.OnValueChangeListener(){...}`
 4. Placer l'objet ***info*** créé précédemment dans la méthode comme cela:
`info.setMinutes(newVal, oldVal);`
 5. Interagir avec la roue et la placer sur une valeur définie au hasard.
 6. Afficher la valeur stockée dans *info* grâce à la méthode *getMinutes* comme ceci : ***info.getMinutes()***.

- *private void boutonValiderDepart()*
 - Identifiant: PARAMETRESVOL_VALIDERDEPART
 - Description: Ce test vise à vérifier que lorsque la méthode est appelée, la chaîne de caractère de l'aéroport de départ sera bien stockée dans l'attribut de classe *InfoVolManager*.

L'environnement de test est le suivant:

 - * Le matériel nécessaire est un téléphone Android muni de la version 8.1 au minimum ou une machine virtuelle Android.
 - Contraintes: L'existence d'un bouton afin de pouvoir appeler la méthode *setOnClickListener*.

Les classes associées aux objets nécessaires au fonctionnement de la méthode doivent également être importées.

 - Dépendances: Les méthodes de la classe *InfoVolManager* devront être testées et validées pour veiller au bon fonctionnement de cette méthode.
 - Procédure du test: Afin de réaliser le test, le testeur devra suivre les étapes suivantes :
 1. Créer un objet **boutonTest** de classe *Button*.
 2. Créer un objet **info** de classe *InfoVolManager*.
 3. Placer le bouton crée précédemment dans la méthode comme cela:
boutonTest.setOnClickListener(new View.OnClickListener(){...})
 4. Placer l'objet **info** crée précédemment dans la méthode comme cela:
info.setAeroportDepart(aerodromes_correspondants.getText().toString());
 5. Appuyer sur le bouton.
 6. Afficher la valeur stockée dans **info** grâce à la méthode *getAeroportDepart* comme ceci : **info.getAeroportDepart()**.

- *private void boutonValiderArrivee()*
 - Identifiant: PARAMETRESVOL_VALIDERDARRIVEE
 - Description: Ce test vise à vérifier que lorsque la méthode est appelée, la chaîne de caractère de l'aéroport d'arrivée sera bien stockée dans l'attribut de classe *InfoVolManager*.
L'environnement de test est le suivant:
 - * Le matériel nécessaire est un téléphone Android muni de la version 8.1 au minimum ou une machine virtuelle Android.
 - Contraintes: L'existence d'un bouton afin de pouvoir appeler la méthode *setOnClickListener*.
Les classes associées aux objets nécessaires au fonctionnement de la méthode doivent également être importées.
 - Dépendances: Les méthodes de la classe *InfoVolManager* devront être testées et validées pour veiller au bon fonctionnement de cette méthode.
 - Procédure du test: Afin de réaliser le test, le testeur devra suivre les étapes suivantes :
 1. Créer un objet **boutonTest** de classe *Button*.
 2. Créer un objet **info** de classe *InfoVolManager*.
 3. Placer le bouton crée précédemment dans la méthode comme cela:
boutonTest.setOnClickListener(new View.OnClickListener(){...})
 4. Placer l'objet **info** crée précédemment dans la méthode comme cela:
info.setAeroportArrivee(aerodromes_correspondantsArrivee.getText().toString());
 5. Appuyer sur le bouton.
 6. Afficher la valeur stockée dans **info** grâce à la méthode *getAeroportArrivee* comme ceci : **info.getAeroportArrivee()**.

5.3.4 ParametresMesureActivity

- *private void setMinutes()*
 - Identifiant: PARAMETRESMESURE_MINUTES
 - Description: Ce test vise à vérifier que lorsque la méthode est appelée, la durée en minutes entre chaque mesure est bien enregistrée dans l'attribut de classe *TimeManager*.
L'environnement de test est le suivant:
 - * Le matériel nécessaire est un téléphone Android muni de la version 8.1 au minimum ou une machine virtuelle Android.
 - Contraintes: L'existence d'une roue (*NumberPicker*) afin de pouvoir appeler la méthode *setOnValueChangeListener*.
Les classes associées aux objets nécessaires au fonctionnement de la méthode doivent également être importées.
 - Dépendances: Les méthodes de la classe *TimeManager* devront être testées et validées pour veiller au bon fonctionnement de cette méthode.
 - Procédure du test: Afin de réaliser le test, le testeur devra suivre les étapes suivantes :
 1. Créer un objet **pickerTest** de classe *NumberPicker*.
 2. Créer un objet **time** de classe *TimeManager*.
 3. Placer la roue créée précédemment dans la méthode comme cela:
pickerTest.setOnValueChangeListener(new NumberPicker.OnValueChangeListener(){...})
 4. Placer l'objet **time** créé précédemment dans la méthode comme cela:
time.setMinutes(newVal);
 5. Interagir avec la roue et la placer sur une valeur définie au hasard.
 6. Afficher la valeur stockée dans *time* grâce à la méthode *getMinutes* comme ceci : **time.getMinutes()**.
- *private void setSecondes()*

- Identifiant: PARAMETRESMESURE_SECONDES
- Description: Ce test vise à vérifier que lorsque la méthode est appelée, la durée en secondes entre chaque mesure est bien enregistrée dans l'attribut de classe *TimeManager*.
L'environnement de test est le suivant:
 - * Le matériel nécessaire est un téléphone Android muni de la version 8.1 au minimum ou une machine virtuelle Android.
- Contraintes: L'existence d'une roue (*NumberPicker*) afin de pouvoir appeler la méthode *setOnValueChangeListener*.
Les classes associées aux objets nécessaires au fonctionnement de la méthode doivent également être importées.
- Dépendances: Les méthodes de la classe *TimeManager* devront être testées et validées pour veiller au bon fonctionnement de cette méthode.
- Procédure du test: Afin de réaliser le test, le testeur devra suivre les étapes suivantes :
 1. Créer un objet **pickerTest** de classe *NumberPicker*.
 2. Créer un objet **time** de classe *TimeManager*.
 3. Placer la roue créée précédemment dans la méthode comme cela:
`pickerTest.setOnValueChangeListener(new NumberPicker.OnValueChangeListener(){...})`
 4. Placer l'objet **time** créé précédemment dans la méthode comme cela:
`time.setSecondes(newVal);`
 5. Interagir avec la roue et la placer sur une valeur définie au hasard.
 6. Afficher la valeur stockée dans *time* grâce à la méthode *getSecondes* comme ceci : **time.getSecondes()**.

- *private void setMinutesDureeMesure()*
 - Identifiant: PARAMETRESMESURE_DUREEMIN
 - Description: Ce test vise à vérifier que lorsque la méthode est appelée, la durée en minutes d'une mesure est bien enregistrée dans l'attribut de classe *DureeMesureManager*.
L'environnement de test est le suivant:
 - * Le matériel nécessaire est un téléphone Android muni de la version 8.1 au minimum ou une machine virtuelle Android.
 - Contraintes: L'existence d'une roue (*NumberPicker*) afin de pouvoir appeler la méthode *setOnValueChangeListener*.
Les classes associées aux objets nécessaires au fonctionnement de la méthode doivent également être importées.
 - Dépendances: Les méthodes de la classe *DureeMesureManager* devront être testées et validées pour veiller au bon fonctionnement de cette méthode.
 - Procédure du test: Afin de réaliser le test, le testeur devra suivre les étapes suivantes :
 1. Créer un objet ***pickerTest*** de classe *NumberPicker*.
 2. Créer un objet ***time*** de classe *DureeMesureManager*.
 3. Placer la roue créée précédemment dans la méthode comme cela:
`pickerTest.setOnValueChangeListener(new NumberPicker.OnValueChangeListener()`...
 4. Placer l'objet ***time*** créé précédemment dans la méthode comme cela:
`time.setMinutes(newVal);`
 5. Interagir avec la roue et la placer sur une valeur définie au hasard.
 6. Afficher la valeur stockée dans *time* grâce à la méthode *getMinutes* comme ceci : ***time.getMinutes()***.

- *private void setSecondesDureeMesure()*
 - Identifiant: PARAMETRESMESURE_DUREESEC
 - Description: Ce test vise à vérifier que lorsque la méthode est appelée, la durée en secondes d'une mesure est bien enregistrée dans l'attribut de classe *DureeMesureManager*.

L'environnement de test est le suivant:

 - * Le matériel nécessaire est un téléphone Android muni de la version 8.1 au minimum ou une machine virtuelle Android.
 - Contraintes: L'existence d'une roue (*NumberPicker*) afin de pouvoir appeler la méthode *setOnValueChangeListener*.

Les classes associées aux objets nécessaires au fonctionnement de la méthode doivent également être importées.

- Dépendances: Les méthodes de la classe *DureeMesureManager* devront être testées et validées pour veiller au bon fonctionnement de cette méthode.
- Procédure du test: Afin de réaliser le test, le testeur devra suivre les étapes suivantes :
 1. Créer un objet **pickerTest** de classe *NumberPicker*.
 2. Créer un objet **time** de classe *DureeMesureManager*.
 3. Placer la roue créée précédemment dans la méthode comme cela:
`pickerTest.setOnValueChangeListener(new NumberPicker.OnValueChangeListener(){...})`
 4. Placer l'objet **time** créé précédemment dans la méthode comme cela:
`time.setSecondes(newVal);`
 5. Interagir avec la roue et la placer sur une valeur définie au hasard.
 6. Afficher la valeur stockée dans *time* grâce à la méthode *getSecondes* comme ceci : `time.getSecondes()`.

- *private void setMinMax()*
 - Identifiant: PARAMETRESMESURE_MINMAX
 - Description: Ce test vise à vérifier que lorsque la méthode est appelée, les valeurs minimum et maximum des objets de classe *NumberPicker* sont bien limitées.

L'environnement de test est le suivant:

 - * Le matériel nécessaire est un téléphone Android muni de la version 8.1 au minimum ou une machine virtuelle Android.
 - Contraintes: L'existence de roues de classe (*NumberPicker*) afin de pouvoir appeler la méthode *setMaxValue* et *setMinValue*.
Les classes associées aux objets nécessaires au fonctionnement de la méthode doivent également être importées.
 - Dépendances: Aucune dépendance à signaler.
 - Procédure du test: Afin de réaliser le test, le testeur devra suivre les étapes suivantes :
 1. Créer 4 objet ***pickerTest[1-4]*** de classe *NumberPicker*.
 2. Appeler sur chaque objet les méthodes *setMaxValue* et *setMinValue* avec les valeurs choisies.
 3. Interagir avec les roue et vérifier la cohérence des valeurs.

- *private void setSwitchSignal()*
 - Identifiant: PARAMETRESMESURE_SIGNAL
 - Description: Ce test vise à vérifier que lorsque la méthode est appelée, le signal choisie est bien enregistrée dans l'attribut de classe *SignalSonoreManager*.

L'environnement de test est le suivant:

 - * Le matériel nécessaire est un téléphone Android muni de la version 8.1 au minimum ou une machine virtuelle Android.
 - Contraintes: L'existence d'un interrupteur (*Switch*) afin de pouvoir appeler la méthode *setOnCheckedChangeListener*.

Les classes associées aux objets nécessaires au fonctionnement de la méthode doivent également être importées.

 - Dépendances: Les méthodes de la classe *SignalSonoreManager* devront être testées et validées pour veiller au bon fonctionnement de cette méthode.
 - Procédure du test: Afin de réaliser le test, le testeur devra suivre les étapes suivantes :
 1. Créer un objet **switchTest** de classe *Switch*.
 2. Créer un objet **signal** de classe *SignalSonoreManager*.
 3. Placer l'interrupteur crée précédemment dans la méthode comme cela:
`switchTest.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener(){...})`
 4. Placer l'objet **signal** crée précédemment dans la méthode comme cela:
`signal.setSignal(isChecked);`
 5. Intéragir avec le switch (*activer ou désactiver*).
 6. Afficher la valeur stockée dans **signal** grâce à la méthode *isSignal* comme ceci : `signal.isSignal()`.

- *private void majEtatBoutonEchelle(int valeur)*
private void majEtatBoutonUti(int valeur)
 - Identifiant: PARAMETRESMESURE_MAJECTELLEUTI
 - Description: Ce test vise à vérifier que lorsque la méthode est appelée, le bouton défini est bien désactivé lorsque que *valeur* est strictement inférieur à 1.
 L'environnement de test est le suivant:
 - * Le matériel nécessaire est un téléphone Android muni de la version 8.1 au minimum ou une machine virtuelle Android.
 - Contraintes: L'existence d'un bouton (*Button*) afin de pouvoir appeler la méthode *setEnabled*.
Les classes associées aux objets nécessaires au fonctionnement de la méthode doivent également être importées.
 - Dépendances: Il n'y a aucune dépendance.
 - Procédure du test: Afin de réaliser le test, le testeur devra suivre les étapes suivantes (à réitérer pour tester l'autre méthode) :
 1. Créer un objet **boutonTest** de classe *Button*.
 2. Définir une valeur pour *valeur*.
 3. Placer le bouton crée précédemment dans la méthode comme cela:
boutonTest.setEnabled(true)(resp.false).
 4. Intéragir avec le bouton.
 5. Vérifier qu'il est impossible de descendre en dessous de 1.

- *private void incrementUtilisateur()*
private void incrementEchelle()
 - Identifiant: PARAMETRESMESURE_INCREMUTIECHELLE
 - Description: Ce test vise à vérifier que lorsque la méthode est appelée, le nombre d'utilisateur (resp. echelle) est bien mise à jour à chaque interaction (incrémantation) avec le bouton désigné.
 L'environnement de test est le suivant:
 - * Le matériel nécessaire est un téléphone Android muni de la version 8.1 au minimum ou une machine virtuelle Android.
 - Contraintes: L'existence d'un bouton (*Button*) afin de pouvoir appeler la méthode *setOnClickListener*.
Les classes associées aux objets nécessaires au fonctionnement de la méthode doivent également être importées.
 - Dépendances: Les méthodes de la classe *ChoixModeManager* (resp. *echelleManager*) et de la classe *NbUserManager* devront être testées et validées pour veiller au bon fonctionnement de cette méthode.
 - Procédure du test: Afin de réaliser le test, le testeur devra suivre les étapes suivantes (à réitérer pour tester l'autre méthode) :
 1. Créer un objet **boutonTest** de classe *Button*.
 2. Créer un objet **donnees** de classe *ChoixModeManager* (resp. *echelleManager*).
 3. Créer un objet **text** de la classe *TextView*
 4. Placer le bouton crée précédemment dans la méthode comme cela:
boutonTest.setOnClickListener(...).
 5. Appeler la méthode *get* des classes souhaitées comme cela:
donnees.getIsChoixMode()(resp. getNombreBouton());
 6. Interagir avec le bouton.
 7. Vérifier le bon fonctionnement de la méthode *majEtatBoutonUti* (resp . *majEtatBoutonEchelle*).

8. Vérifier le bon affichage de ***text*** après chaque interaction avec ***boutonTest***.
- *private void decrementUtilisateur()*
private void decrementEchelle()
 - Identifiant: PARAMETRESMESURE_DECREMUTIECHELLE
 - Description: Ce test vise à vérifier que lorsque la méthode est appelée, le nombre d'utilisateur (resp. echelle) est bien mise à jour à chaque interaction (décrémentation) avec le bouton désigné.
 L'environnement de test est le suivant:
 - * Le matériel nécessaire est un téléphone Android muni de la version 8.1 au minimum ou une machine virtuelle Android.
 - Contraintes: L'existence d'un bouton (*Button*) afin de pouvoir appeler la méthode *setOnClickListener*.
Les classes associées aux objets nécessaires au fonctionnement de la méthode doivent également être importées.
 - Dépendances: Les méthodes de la classe *ChoixModeManager* (resp. *echelleManager*) et de la classe *NbUserManager* devront être testées et validées pour veiller au bon fonctionnement de cette méthode.
 - Procédure du test: Afin de réaliser le test, le testeur devra suivre les étapes suivantes (à réitérer pour tester l'autre méthode) :
 1. Créer un objet ***boutonTest*** de classe *Button*.
 2. Créer un objet ***text*** de la classe *TextView*
 3. Placer le bouton crée précédemment dans la méthode comme cela:
boutonTest.setOnClickListener(...).
 4. Placer les éléments nécessaires dans la méthode.
 5. Intéragir avec le bouton.
 6. Vérifier le bon fonctionnement de la méthode *majEtatBoutonUti* (resp. *.majEtatBoutonEchelle*).
 7. Vérifier le bon affichage de ***text*** après chaque interaction avec ***boutonTest***.

5.3.5 MesureMultiActivity

- *private void startTimer(int index)*
 - Identifiant: MESUREMULTI_TIMER
 - Description: Ce test vise à vérifier que lorsque la méthode est appelée, le compteur a bien la durée préalablement définie et que la charge mentale a bien été ajoutée dans le vecteur de l'utilisateur associé.
L'environnement de test est le suivant:
 - * Le matériel nécessaire est un téléphone Android muni de la version 8.1 au minimum ou une machine virtuelle Android.
 - Contraintes: L'existence d'une activité cible afin de vérifier la redirection à la fin du compteur.
Les classes associées aux objets nécessaires au fonctionnement de la méthode doivent également être importées.
 - Dépendances: Les méthodes de la classe *Utilisateur* devront être testées et validées pour veiller au bon fonctionnement de cette méthode.
 - Procédure du test: Afin de réaliser le test, le testeur devra suivre les étapes suivantes :
 1. Créer une variable **temps** et **temps_restant** de type *Long*.
 2. Créer un objet **uti** de classe *ArrayList* (de *Utilisateur*)
 3. Créer un objet **cible** de classe *Intent*.
 4. Assigner des valeurs au hasard pour ses variable.
 5. Placer les variables créées précédemment dans la méthode comme cela:
*mCountDownTimer = new CountDownTimer(temps_restant, 1000)
temps_restant = millisUntilFinished;*
 6. Vérifier qu'à la fin du chronomètre, le testeur est bien redirigé vers **cible**.
 7. Afficher les valeurs de **uti** et vérifier la cohérence.

- *private void startListening()*
 - Identifiant: MESUREMULTI_STARTLI
 - Description: Ce test vise à vérifier que lorsque la méthode est appelée, la reconnaissance vocale est bien activée.
L'environnement de test est le suivant:
 - * Le matériel nécessaire est un téléphone Android muni de la version 8.1 au minimum ou une machine virtuelle Android. Le téléphone ou la machine virtuelle devra disposer d'une entrée audio.
 - Contraintes: Il n'y a aucune contrainte.
Les classes associées aux objets nécessaires au fonctionnement de la méthode doivent également être importées.
 - Dépendances: La méthode qui permet d'autoriser la reconnaissance vocale doit être testée (*onRequestPermissionsResult*).
 - Procédure du test: Afin de réaliser le test, le testeur devra suivre les étapes suivantes :
 1. Créer un objet **cible** de classe *Intent*.
 2. Créer un objet **reco Vocale** de la classe *SpeechRecognizer*.
 3. Placer les objets créés précédemment dans la méthode comme cela:
reco Vocale.startListening(cible);
 4. Vérifier que quand la méthode est appelée, alors la reconnaissance vocale s'active.

- *private void stopListening()*
 - Identifiant: MESUREMULTI_STOPLI
 - Description: Ce test vise à vérifier que lorsque la méthode est appelée, la reconnaissance vocale se désactive.
L'environnement de test est le suivant:
 - * Le matériel nécessaire est un téléphone Android muni de la version 8.1 au minimum ou une machine virtuelle Android. Le téléphone ou la machine virtuelle devra disposer d'une entrée audio.
 - Contraintes: Il n'y a aucune contrainte.
Les classes associées aux objets nécessaires au fonctionnement de la méthode doivent également être importées.
 - Dépendances: La méthode qui permet d'autoriser la reconnaissance vocale doit être testée (*onRequestPermissionsResult*) et la méthode *startListening* doit également être testée afin d'activer la reconnaissance vocale dans un premier temps.
 - Procédure du test: Afin de réaliser le test, le testeur devra suivre les étapes suivantes :
 1. Créer un objet **cible** de classe *Intent*.
 2. Créer un objet **reco Vocale** de la classe *SpeechRecognizer*.
 3. Placer les objets créés précédemment dans la méthode comme cela:
reco Vocale.stopListening();
 4. Vérifier que quand la méthode est appelée, alors la reconnaissance vocale se désactive.

- *private void reconnaissanceVocale()*
 - Identifiant: MESUREMULTI_RECVOCALE
 - Description: Ce test vise à vérifier le bon fonctionnement du traitement des données en entrée (à l'oral).
L'environnement de test est le suivant:
 - * Le matériel nécessaire est un téléphone Android muni de la version 8.1 au minimum ou une machine virtuelle Android. Le téléphone ou la machine virtuelle devra disposer d'une entrée audio.
 - Contraintes: Il n'y a aucune contrainte.
Les classes associées aux objets nécessaires au fonctionnement de la méthode doivent également être importées.
 - Dépendances: La méthode qui permet d'autoriser la reconnaissance vocale doit être testée (*onRequestPermissionsResult*). Les méthodes *startListening* et *stopListening* devront également être testées pour pouvoir lancer et arrêter la reconnaissance vocale. Enfin, l'ensemble des méthodes de la classe *Utilisateur* doivent également être testées.
 - Procédure du test: Afin de réaliser le test, le testeur devra suivre les étapes suivantes :
 1. Définir les variables comme nécessaires au fonctionnement de la méthode. (Voir méthode)
 2. Énoncer la valeur souhaitez à l'oral sous cette forme : *Valeur [1-10]*
 3. Vérifier si l'application nous permet de passer au prochain utilisateur à noter.
 4. Énoncer une valeur à l'oral qui est hors échelle de mesure et vérifier si le message affiché à l'écran correspond bien à l'erreur.
 5. Faire un tour de test, et vérifier que lorsque tous les utilisateurs ont été notés, alors l'application affiche *CompteurChargeMentaleActivity*.
 6. Afficher à la fin des tests, les valeurs stockées dans le *ArrayList* *d'Utilisateur*.

- *private void mesurerChargeMentale(Utilisateur utilisateur)*
 - Identifiant: MESUREMULTI_MESURECHARGE
 - Description: Ce test vise à vérifier que la méthode s'appelle bien elle-même tant que la condition d'arrêt n'est pas remplie (récursivité).
L'environnement de test est le suivant:
 - * Le matériel nécessaire est un téléphone Android muni de la version 8.1 au minimum ou une machine virtuelle Android.
 - Contraintes: Il n'y a aucune contrainte.
Les classes associées aux objets nécessaires au fonctionnement de la méthode doivent également être importées.
 - Dépendances: Les méthodes de la classe *Utilisateur* devront être testées au préalable.
 - Procédure du test: Afin de réaliser le test, le testeur devra suivre les étapes suivantes :
 1. Créer un objet ***Text*** de classe *TextView*.
 2. Vérifier la création des boutons de manière dynamique dans la méthode.
 3. Appeler sur les boutons la méthode *setOnClickListener*.
 4. Effectuer plusieurs mesures et vérifier à chaque fois que ***Text*** se met à jour.
 5. Quand tous les utilisateurs ont été mesurés, vérifier que la méthode permet de basculer sur *CompteurChargeMentaleActivity*.
 6. Quand l'ensemble des mesures a été effectué, afficher à la fin des tests, les valeurs stockées dans le *ArrayList* d'*Utilisateur*.

5.3.6 CompteurChargeMentaleActivity

- *private void startTimer()*
 - Identifiant: COMPTEUR_TIMER
 - Description: Ce test vise à vérifier que lorsque la méthode est appelée alors un décompte est lancé et permet à la fin du compteur de passer sur une autre activité.
L'environnement de test est le suivant:
 - * Le matériel nécessaire est un téléphone Android muni de la version 8.1 au minimum ou une machine virtuelle Android.
 - Contraintes: Il n'y a aucune contrainte.
Les classes associées aux objets nécessaires au fonctionnement de la méthode doivent également être importées.
 - Dépendances: Il n'y a pas de dépendance.
 - Procédure du test: Afin de réaliser le test, le testeur devra suivre les étapes suivantes :
 1. Créer une variable **temps** et **temps_restant** de type *Long*.
 2. Créer un objet **cible** de classe *Intent*.
 3. Assigner des valeurs au hasard pour ses variables.
 4. Placer les variables créées précédemment dans la méthode comme cela:
mCountDownTimer = new CountDownTimer(temps_restant, 1000)
***temps_restant** = millisUntilFinished;*
 5. Vérifier qu'à la fin du chronomètre, le testeur est bien redirigé vers **cible**.

- *private void updateCountDownText()*
 - Identifiant: COMPTEUR_UPDATE
 - Description: Ce test vise à vérifier qu'à chaque mise à jour du chronomètre, alors le texte associé à ce compteur se met lui aussi à jour.
L'environnement de test est le suivant:
 - * Le matériel nécessaire est un téléphone Android muni de la version 8.1 au minimum ou une machine virtuelle Android.
 - Contraintes: Il n'y a aucune contrainte.
Les classes associées aux objets nécessaires au fonctionnement de la méthode doivent également être importées.
 - Dépendances: Il n'y a pas de dépendance.
 - Procédure du test: Afin de réaliser le test, le testeur devra suivre les étapes suivantes :
 1. Créer une variable **temps** et **temps_restant** de type *Long*.
 2. Initialiser les variables de la méthode avec les variables **temps** et **temps_restant**.
 3. Créer un objet **text** de classe *TextView*
 4. Exécuter la méthode et vérifier que **text** se met bien à jour à chaque tic.

5.3.7 ChoixNomUtiActivity

- *private void updateCurrentUserNumber()*
 - Identifiant: CHOIXNOM_UPDATE
 - Description: Ce test vise à vérifier que le texte affiché à l'écran se met bien à jour à chaque fois que la méthode est appelée.
L'environnement de test est le suivant:
 - * Le matériel nécessaire est un téléphone Android muni de la version 8.1 au minimum ou une machine virtuelle Android.
 - Contraintes: Il n'y a aucune contrainte.
Les classes associées aux objets nécessaires au fonctionnement de la méthode doivent également être importées.
 - Dépendances: Il n'y a pas de dépendance.
 - Procédure du test: Afin de réaliser le test, le testeur devra suivre les étapes suivantes :
 1. Créer un objet **text** de classe *TextView*.
 2. Créer une variable **valeur** de type *int*.
 3. Placer **text** comme cela :
this.text.setText("Utilisateur " + (valeur+1));
 4. Incrémenter de 1 **valeur** et rappeler la méthode.
 5. Vérifier que la mise à jour de **text** a bien eu lieu avec la bonne valeur.

- *private void clearText()*
 - Identifiant: CHOIXNOM_UPDATE
 - Description: Ce test vise à vérifier que le texte "Nom Utilisateur" dans la barre d'écriture s'enlève bien à chaque interaction par le testeur.
L'environnement de test est le suivant:
 - * Le matériel nécessaire est un téléphone Android muni de la version 8.1 au minimum ou une machine virtuelle Android.
 - Contraintes: Il n'y a aucune contrainte.
Les classes associées aux objets nécessaires au fonctionnement de la méthode doivent également être importées.
 - Dépendances: Il n'y a pas de dépendance.
 - Procédure du test: Afin de réaliser le test, le testeur devra suivre les étapes suivantes :
 1. Créer un objet **text** de classe *TextView*.
 2. Placer **text** comme cela :
`text.setOnFocusChangeListener(new View.OnFocusChangeListener(){
 text.setHint(" ");
 text.setText("Nom d'utilisateur");});`
 3. Vérifier que la mise à jour de **text** a bien eu lieu à chaque interaction.

- *private void nextUser()*
 - Identifiant: CHOIXNOM_NEXT
 - Description: Ce test vise à vérifier que lorsque le testeur appuie sur le bouton déterminé, alors le nom est bien ajouté de manière temporaire, puis de manière définitive lorsque le testeur le confirme.
L'environnement de test est le suivant:
 - * Le matériel nécessaire est un téléphone Android muni de la version 8.1 au minimum ou une machine virtuelle Android.
 - Contraintes: Il n'y a aucune contrainte.
Les classes associées aux objets nécessaires au fonctionnement de la méthode doivent également être importées.
 - Dépendances: Les méthodes de la classe *UserManager*, *DataManager* et la méthode *updateCurrentUserNumber* devront être testées.
 - Procédure du test: Afin de réaliser le test, le testeur devra suivre les étapes suivantes :
 1. Créer un objet **cible** de classe *Intent*.
 2. Définir une taille pour *UserManager.getInstance().getUserNames().size()*
 3. Ajouter plusieurs nombres d'utilisateurs et vérifier que lorsque nombre d'utilisateurs atteint le maximum (cf. étape 2) alors un pop-up apparaît.
 4. Vérifier que lorsque le testeur appuie sur confirmer, alors le testeur est bien redirigé vers **cible**.
 5. Sur l'activité **cible**, afficher les noms qui doivent être affichés dans le *ArrayList* de *String* de la classe *UserManager*. Il faut également vérifier que les utilisateurs associés aux noms sont également créés.

- *private void pastUser()*
 - Identifiant: CHOIXNOM_PAST
 - Description: Ce test vise à vérifier que lorsque le testeur appuie sur le bouton déterminé, alors le nom d'utilisateur courant est supprimé et il a la possibilité de le ressaisir.

L'environnement de test est le suivant:

 - * Le matériel nécessaire est un téléphone Android muni de la version 8.1 au minimum ou une machine virtuelle Android.
 - Contraintes: Il n'y a aucune contrainte.
Les classes associées aux objets nécessaires au fonctionnement de la méthode doivent également être importées.
 - Dépendances: Les méthodes de la classe *UserManager* et la méthode *updateCurrentUserNumber* devront être testées.
 - Procédure du test: Afin de réaliser le test, le testeur devra suivre les étapes suivantes :
 1. Créer un objet **boutonTest** de classe *Button*.
 2. Créer un objet **text**
 3. Placer **boutonTest** comme cela:
`boutonTest.setOnClickListener(new View.OnClickListener()`
 4. Ajouter un nom d'utilisateur.
 5. Appuyer sur le bouton et vérifier que **text** s'actualise bien.
 6. Vérifier que dans le *ArrayList* de *String* de la classe *UserManager*, le nom a bien été supprimé.

5.3.8 ChargeMentaleSeulActivity

- *public void boutonMesure()*
 - Identifiant: CHARGE_SEUL_BOUTON_MESURE
 - Description: Ce test permet de vérifier que la méthode crée bien un nombre donné de boutons fonctionnels.
L'environnement de test est le suivant:
 - * Le matériel nécessaire est un téléphone Android muni de la version 8.1 au minimum ou une machine virtuelle Android.
 - Contraintes: Il n'y a pas de contrainte pour cette méthode.
 - Dépendances: Les méthodes de la classe *Utilisateur* doivent être testées pour vérifier le bon fonctionnement de cette méthode.
 - Procédure du test: Afin de réaliser le test, le testeur devra suivre les étapes suivantes :
 1. Créer les attributs suivants :
 - * Un entier ***nbBoutons*** correspondant au nombre de boutons.
 - * Un objet ***countDownTimer*** de type *CountDownTimer* pour le compteur de temps.
 - * Une liste d'objets utilisateurs de type utilisateurs. (*ArrayList*).
 2. Créer une activité ***test*** qui nous servira à vérifier que la redirection fonctionne.
 3. Placer les attributs initialisés dans la méthode et lancer l'application.
Si les boutons s'affichent et qu'ils redirigent bien tous vers l'activité ***test*** alors les boutons fonctionnent.
 - Maintenant il faut vérifier que les valeurs sont bien enregistrées :
 1. Reprendre les étapes précédentes et mettre en commentaire la ligne *startActivity()* dans la méthode.
 2. Créer un objet *TextView* et le relier de manière à ce qu'il se mette à jour lorsqu'un bouton est cliqué, on pourra dans l'interface *onClick()* de la

méthode rajouter un appel de la méthode `setText(clickedButton.getId())`, appelée par le bouton qui mettra à jour le texte du `Textview`. Cette méthode affichera l'id du bouton, qui correspond à son placement dans l'activité (le bouton le plus haut a l'id 1, le deuxième l'id 2 etc).

3. Lancer l'application, appuyer sur chaque boutons et confirmer qu'à chaque fois qu'un bouton est sélectionné son id s'affiche bien à l'écran.

- *private void reconnaissanceVocale()*
 - Identifiant: CHARGE_SEUL_RECO_VOCALE
 - Description: Ce test permet de vérifier que la méthode enregistre bien les valeurs transmises par la voix par l'utilisateur.

L'environnement de test est le suivant:

- * Le matériel nécessaire est un téléphone Android muni de la version 8.1 au minimum ou une machine virtuelle Android. Le téléphone ou la machine virtuelle devra disposer d'une entrée audio.
- Contraintes: L'existence d'une activité cible afin de vérifier la redirection à la fin du compteur.
- Dépendances: La méthode *startListening()* doit être testée afin de vérifier que la reconnaissance vocale s'active bien. Les méthodes de la classe *Utilisateur* doivent également être testées afin de vérifier que les charges mentales sont bien enregistrées.
- Procédure du test: Afin de réaliser le test, le testeur devra suivre les étapes suivantes :
 1. Créer les attributs suivants :
 - * Un objet ***Reco Vocale*** de type *SpeechRecognizer*.
 - * Un entier ***valeurChargeMentale***.
 - * un objet ***context*** de type *Context*.
 - * un objet ***utilisateurs*** de *ArrayList* d'objet de type *Utilisateur*.
 - * un entier ***echelle_valeur***.
 - * un objet ***mCountDownTimer*** de type *CountDownTimer*.
 2. Créer une activité test qui nous servira à vérifier que la redirection fonctionne.
 3. Placer les attributs initialisés dans la méthode, lancer l'application et tester la méthode en rentrant une valeur par la voix, si après avoir donné la valeur l'application nous redirige vers l'activité test alors la redirection fonctionne.
- Maintenant il faut vérifier que les valeurs sont bien enregistrées :

1. Reprendre les étapes précédentes et mettre en commentaire la ligne `startActivity()` dans la méthode.
2. Créer un objet `TextView` affiché dans l'activité et le relier de manière à ce qu'il se mette à jour lorsqu'une valeur est donnée par la voix, on pourra dans la méthode rajouter une méthode `setText(valeurChargeMentale)`. Cette méthode affichera la valeur donnée par l'utilisateur.
3. Lancer l'application, donner une valeur par la voix et confirmer que cette valeur s'afficher à l'écran.

- *private void startTimer()*
 - Identifiant: SEULE_TIMER
 - Description: Ce test vise à vérifier que lorsque la méthode est appelée alors un décompte est lancé et permet à la fin du compteur de passer sur une autre activité. Elle doit en plus ajouter la valeur maximale à l'utilisateur. L'environnement de test est le suivant:
 - * Le matériel nécessaire est un téléphone Android muni de la version 8.1 au minimum ou une machine virtuelle Android.
 - Contraintes: Il n'y a aucune contrainte.
Les classes associées aux objets nécessaires au fonctionnement de la méthode doivent également être importées.
 - Dépendances: Les méthodes de la classe *Utilisateur* et *UserManager* doivent être testées afin de pouvoir ajouter les charges mentales.
 - Procédure du test: Afin de réaliser le test, le testeur devra suivre les étapes suivantes :
 1. Créer une variable ***temps*** et ***temps_restant*** de type *Long*.
 2. Créer un objet ***cible*** de classe *Intent*.
 3. Créer un objet ***uti*** qui est un *ArrayList* d'*Utilisateur*.
 4. Assigner des valeurs au hasard pour ses variables.
 5. Placer les variables créées précédemment dans la méthode comme cela:
`mCountDownTimer = new CountDownTimer(temps_restant, 1000)
temp_restant = millisUntilFinished;`
 6. Vérifier qu'à la fin du chronomètre, le testeur est bien redirigé vers ***cible***.
 7. Vérifier sur ***cible*** que les charges mentales ont bien été rentrées grâce aux méthodes de *UserManager*.

5.3.9 InfoVolManager

- *public void setHeures()*
 - Identifiant: INFO_VOL_MANAGER_HEURES
 - Description: Ce test permet de vérifier que la conversion des heures en minutes fonctionne bien et que les heures sont bien ajoutées à la durée totale du vol. L'environnement de test est le suivant:
 - * Le matériel nécessaire est un téléphone Android muni de la version 8.1 au minimum ou une machine virtuelle Android.
 - Contraintes: Il n'y a aucune contrainte.
 - Dépendances: Il n'y a aucune dépendance.
 - Procédure du test: Afin de réaliser le test, le testeur devra suivre les étapes suivantes :
 1. Créer une instance de *InfoVolManager*.
 2. Appeler la méthode *setHeures()* et lui donner en paramètres un entier non nul est un entier égal à 0.
 3. Appeler un deuxième fois la méthode cette fois si en rentrant un entier non nul qui sera les heures que l'on cherche à récupérer et l'entier non nul précédemment rentré lors du premier appel de la méthode.
 4. Appeler la méthode *getDureeVol()* pour récupérer le temps rentré et afficher le résultat.
 5. Après l'exécution du test, si la valeur affichée correspond aux heures que l'on cherche à récupérer alors le test est concluant.

- *public void setMinutes()*
 - Identifiant: INFO_VOL_MANAGER_MINUTES
 - Description: Ce test permet de vérifier que les minutes sont bien ajoutées à la durée total du vol.
L'environnement de test est le suivant:
 - * Le matériel nécessaire est un téléphone Android muni de la version 8.1 au minimum ou une machine virtuelle Android.
 - Contraintes: Il n'y a aucune contrainte.
 - Dépendances: Il n'y a aucune dépendance.
 - Procédure du test: Afin de réaliser le test, le testeur devra suivre les étapes suivantes :
 1. Créer une instance de *InfoVolManager*.
 2. Appeler la méthode *setMinutes()* et lui donner en paramètres un entier non nul est un entier égal à 0.
 3. Appeler un deuxième fois la méthode cette fois si en rentrant un entier non nul qui sera les minutes que l'on cherche à récupérer et l'entier non nul précédemment rentré lors du premier appel de la méthode.
 4. Appeler la méthode *getDureeVol()* pour récupérer le temps rentré et afficher le résultat.
 5. Après l'exécution du test, si la valeur affichée correspond aux minutes que l'on cherche à récupérer alors le test est concluant.

5.3.10 SauvegardeEtEnvoi

- public void envoyerRequetePost()
 - Identifiant: SAUVEGARDE_ET_ENVOI_ENVOYER_REQUETE_POST
 - Description: Ce test permet de vérifier qu'une requête a bien été envoyée.
L'environnement de test est le suivant:
 - * Le matériel nécessaire est un téléphone Android muni de la version 8.1 au minimum ou une machine virtuelle Android.
 - Contraintes: Une connexion internet et une url cible valide, c'est-à-dire acceptant les requêtes HTTP, sont indispensables. Il faudra aussi avoir un moyen de s'assurer que la cible a bien récupéré la requête.
 - Dépendances: Cette classe nécessite d'être jumelée à une activité pour fonctionner correctement.
 - Procédure du test: Afin de réaliser le test, le testeur devra suivre les étapes suivantes :
 1. Créer une instance de la classe dans une activité.
 2. Fournir une url cible valide dans le constructeur de la classe.
 3. Appeler la méthode *envoyerRequetePost()*.
 4. Compiler, et exécuter. Pour savoir si les bonnes données ont été envoyées il faut se rendre sur le serveur web hébergeant la cible, la cible (un script php) pourra par exemple créer un fichier texte avec le contenu de la requête à l'intérieur.

5.3.11 CodeManager

- *public void creationMdpAvion()*
 - Identifiant: CODEMANAGER_MDP_AVION
 - Description: Ce test permet de vérifier que le code pour visualiser les résultats sur le module d'expérimentation si le mode vol est activé est bien créé. La méthode prend en entrée des entiers et des chaînes de caractères et réalise une suite de concaténations et d'opérations sur ces paramètres et renvoie une chaîne de caractères commençant par la lettre v. L'environnement de test est le suivant:
 - * Le matériel nécessaire est un téléphone Android muni de la version 8.1 au minimum ou une machine virtuelle Android.
 - Contraintes: Il n'y a pas de contrainte pour réaliser ce test.
 - Dépendances: La méthode *getMotDePasse()* afin de pouvoir récupérer le mot de passe créé.
 - Procédure du test: Afin de réaliser le test, le testeur devra suivre les étapes suivantes :
 1. Créer une classe principale.
 2. Créer une instance de la classe *CodeManager*.
 3. Appeler la méthode *creationMdpAvion* et lui donner les paramètres suivant dans cet ordre : un entier, une chaîne de caractères, une chaîne de caractères, un entier, un entier, un entier.
 4. Afficher le code en appelant la méthode *getMotDePasse*.
 5. Compiler, exécuter et vérifier si le code affiché correspond aux paramètres donnés et qu'il commence par la lettre v.

- *public void creationMdpNormal()*
 - Identifiant: CODEMANAGER_MDP_NORMAL
 - Description: Ce test permet de vérifier que le code pour visualiser les résultats sur le module d'expérimentation si le mode vol n'est pas activé est bien créé. La méthode prend en entrée des entiers et des chaînes de caractères et réalise une suite de concaténations et d'opérations sur ces paramètres et renvoie une chaîne de caractères. L'environnement de test est le suivant:
 - * Le matériel nécessaire est un téléphone Android muni de la version 8.1 au minimum ou une machine virtuelle Android.
 - Contraintes: Il n'y a pas de contrainte pour réaliser le test.
 - Dépendances: La méthode *getMotDePasse()* afin de pouvoir récupérer le mot de passe créé.
 - Procédure du test: Afin de réaliser le test, le testeur devra suivre les étapes suivantes :
 1. Créer une classe principale.
 2. Créer une instance de la classe *CodeManager*.
 3. Appeler la méthode *creationMdpAvion* et lui donner les paramètres suivant dans cet ordre : un entier, une chaîne de caractères, une chaîne de caractères, un entier, un entier, un entier.
 4. Afficher le code en appelant la méthode *getMotDePasse*.
 5. Compiler, exécuter et vérifier si le code affiché correspond aux paramètres donnés.

5.4 Tests unitaires du module d'exploitation des données

5.4.1 scriptcible.php

- *Test complet du script*

- Identifiant : SCRIPT_CIBLE
- Description : Test permettant de vérifier que ce script, si il reçoit une requête HTTP POST comportant des données cohérentes (c'est-à-dire des données issues d'une expérimentation sur l'application), stocke bien ces données dans un fichier texte.
- Contraintes :
- Dépendances :
- Procédure du test :
 1. Créer un objet JSON de cette forme "clé" : "valeur", la clé doit correspondre à celle présente sur le serveur web, la valeur est une chaîne de caractère contenant les données que l'on souhaite envoyer au serveur.
 2. La valeur de l'objet doit avoir cette forme "Avion1.2.1.2.7.7.4.6-Avion2.3.1.3.7.0.0.6:Avion2.4.1.4.7.0.0-Avion1.2.1.2.7.7.4.6" par exemple. Chaque ":" permet de séparer les données des différents utilisateurs et les tirets permettent de séparer pour un même utilisateur, ces données personnelles et les données qu'il a attribué aux autres. Les données en elles mêmes sont séparées par des ".".
 3. Effectuer une requête HTTP de méthode POST contenant cet objet JSON vers le script PHP.
 4. Se rendre sur le serveur web, si dans le répertoire php se trouve un fichier texte portant pour nom le code de l'expérimentation, et si ce fichier contient les informations liées à l'expérimentation alors le test est concluant

5.4.2 tableau.php

- *initialisation*

- Identifiant : TABLEAU_INIT
- Description : Test permettant de vérifier que cette fonction lis avec succès dans un fichier texte donné, puis que cette fonction récupère les données du texte avec succès aussi.
- Contraintes :
- Dépendances : Un fichier texte contenant des informations cohérentes, voir l'exemple ci-dessous.

```
2.0.Charles de Gaulle International Airport.Josep Tarradellas Barcelona-El Prat Airport
Avion1.2.1.2.7.7.4.6-Avion2.3.1.3.7.0.0.6-Avion3.4.1.4.7.0.0
Avion2.2.3.4.2.1.7.7-Avion1.2.3.4.5.3.0.0-Avion3.2.7.3.4.0.0
Avion3.2.3.4.2.1.7.7-Avion1.2.3.4.5.3.0.0-Avion2.2.7.3.4.0.0
```

Figure 19: Exemple de fichier texte qui doit être utilisé pour le bon fonctionnement de la fonction

- Procédure du test :

1. Créer deux variables une correspondant au nom du fichier, et l'autre à son chemin d'accès relatif.
2. Créer un tableau d'indice qui recevra le résultat de la fonction.
3. Assigner au tableau le résultat de la fonction.
4. Afficher le tableau et vérifier que les informations sont les bonnes. Si on se réfère à la figure 1, dans la case 0 du tableau nous devrions avoir la deuxième ligne, dans la case 1 la troisième et dans la case 2 la dernière ligne.

- *noms*

- Identifiant : TABLEAU_NOMS
- Description : Test permettant de vérifier que cette fonction, à partir d'un tableau d'indices donnés, récupère les bonnes informations de ce même tableau.
- Contraintes :

- Dépendances :
- Procédure du test :
 1. Créer un tableau d'indices comportant pour chaque case les données nécessaires. Le tableau devrait être celui que l'on obtiendrait après avoir exécuté la fonction initialisation.
 2. Placer ce tableau dans les paramètres de l'appel de la fonction.
 3. Créer un autre tableau d'indices vide.
 4. Assigner au tableau vide le résultat de la fonction.
 5. Afficher le tableau et vérifier que dans chaque case se trouve un nom, qui est le premier mot de chaque case du tableau d'entrée.
- *creationTableauTriHTML*
 - Identifiant : TABLEAU_TRI
 - Description : Test visant à vérifier que cette fonction crée bien un tableau HTML, avec les bonnes données à l'intérieur.
 - Contraintes :
 - Dépendances :
 - Procédure du test :
 1. Créer un tableau d'indices comportant pour chaque case les données nécessaires et un autre comportant pour chaque cases les noms de chaque utilisateurs.
 2. Placer ces deux variables dans les paramètres de l'appel de la fonction.
 3. Exécuter le code, si un tableau à double entrée cohérent est affiché le test est concluant.
- *creationTableauBaseHTML*
 - Identifiant : TABLEAU_BASE
 - Description : Test visant à vérifier que cette fonction crée bien un tableau HTML, avec les bonnes données à l'intérieur.
 - Contraintes :
 - Dépendances :
 - Procédure du test :

1. Créer deux variables une correspondant à un fichier, et l'autre à son nom.
2. Placer ces deux variables dans les paramètres de l'appel de la fonction.
3. Exécuter le code, si un tableau cohérent est affiché le test est concluant.

5.4.3 **graphes.php**

- Identifiant : GRAPHES_CREATION
- Description : Ce test vise à vérifier que les données et l'affichage des graphes sont exacts.
- Contraintes : Avoir saisi un code valide sur la page de connexion.
- Dépendances : Il n'y a pas de dépendances associées à cette méthode.
- Procédure du test :
 1. Créer une fonction PHP qui récupère le code saisi par l'utilisateur via des cookies, lis les données de chaque ligne du fichier texte en séparant les nom des participants et leurs valeurs de mesure de charge mentale, on utilisera pour cela les fonctions PHP *init* et *noms*. Ces données seront stockées dans les variables \$noms et \$moyennes.
 2. Convertir les données du php vers JS en les convertissant en objets JSON via la fonction php *json_encode()* et en les plaçant dans deux variables \$noms_json et \$moyennes_json .
 3. Exécuter les fonctions JavaScript, si un tableau cohérent est affiché le test est concluant.

6 Tests d'intégration

Du côté de l'application, les tests d'intégration vont être réalisés dans l'ordre suivant :

1. Tester que les différentes classes *Manager* sont intégrables dans les *Activités* nécessaires.
2. Tester que les objets des classes *Manager* peuvent bien être stockées dans les classes *Singleton*
3. Tester que la transition entre les différentes *Activités* se réalisent bien.

6.1 Tests d'intégration de l'application

- Test des différentes classes *Manager* dans les Activités
 - Identification: INTEG_MANAGER
 - Description: Le but de ce test est de vérifier que pour chaque *Activité* qui nécessite l'utilisation d'une classe *Manager*, alors cette activité pourra l'utiliser sans souci de visibilité (encapsulation).
 - Contraintes: Il n'y a pas de contrainte liée à ce test.
 - Dépendances: Pour effectuer ce test, les tests unitaires liés aux différentes classes *Manager* devront être effectués afin de vérifier leur fonctionnement individuel afin de les utiliser dans les *Activités*.
 - Procédure du test: Afin de réaliser le test, le testeur devra suivre les étapes suivantes :
 1. Créer une activité ***activity***.
 2. Créer autant d'objets de classe dites *Manager* nécessaires afin de tous les tester dans ***activity***.
 3. Tester les différentes méthodes *set* et *get* des différentes classes afin de vérifier qu'elles sont bien utilisables dans ***activity***. Si c'est le cas, alors l'intégration a été réalisée avec succès. (*Ce test est à réaliser avec toutes les activités qui nécessitent l'utilisation de classe dites Manager.*)

- **Test des classes *singleton* avec les classes *Manager***
 - Identification: INTEG_SINGLETON
 - Description: Le but est de vérifier que la classe Singleton permet bien de stocker et de manipuler les différents objets de classe *Manager*.
 - Contraintes: Il n'y a pas de contrainte liée à ce test.
 - Dépendances: Pour effectuer ce test, les tests unitaires liés aux différentes classes *Manager* et des classes *Singleton* devront être effectués afin de vérifier leur fonctionnement individuel afin de les utiliser dans les *Activités*.
 - Procédure du test: Afin de réaliser le test, le testeur devra suivre les étapes suivantes :
 1. Créer une activité ***activity***.
 2. Créer autant d'objets de classe dites *Manager* nécessaires afin de tous les tester dans ***activity***.
 3. Créer des objets de classes *DataManager* (resp. *UserManager*) dans ***activity***
 4. Utiliser les méthodes *set* des classes *Singleton* (*DataManager* et *UserManager*) avec les objets de classe dites *Manager*.
 5. Vérifier que ces objets sont bien récupérables depuis les classes *singleton* avec la méthode *getInstance* et les méthodes *get*. Si c'est le cas, alors l'intégration a été réalisée avec succès.

- **Test d'intégration des activités**

- Identification: INTEG_ACTIVITY;
- Description: Le but est de vérifier que toutes les *Activités* puisse interagir entre elles sans perturber le fonctionnement individuel de chaque *Activité*.
- Contraintes: Il n'y a pas de contrainte liée à ce test.
- Dépendances: Pour effectuer ce test, chaque méthode d'une *Activité* doit fonctionner parfaitement. Elle doit être testée par le biais d'un test unitaire. La méthode *startActivity* de la classe *Intent* doit également être testée afin de pouvoir changer d'*Activité*.
- Procédure du test: Afin de réaliser le test, le testeur devra suivre les étapes suivantes :
 1. Créer un objet ***transition*** de classe *Intent* dans chaque activité.
 2. Dans chaque *Activité*, connecter chaque *Activité* selon les besoins à l'aide de ***transition*** et de la méthode *startActivity*. Par exemple *MainActivity* avec *ChoixModeActivity* comme ceci:

```
Intent transition= new Intent(getApplicationContext(), ChoixModeActivity.class);
startActivity(transition);
```
 3. Vérifier que le comportement de chaque *Activité* n'est pas remise en cause par une autre *Activité*. Si c'est le cas, alors l'intégration a été réalisée avec succès.

7 Références et sources

[1] Le lien permettant de télécharger et consulter le cahier des charges:
https://forge.ens.math-info.univ-paris5.fr/attachments/download/10942/Cahier_des_charges_PROJET_L2P1%20VERSION%202.0.0.pdf

[2] Le lien permettant de télécharger et consulter le cahier des recettes:
https://forge.ens.math-info.univ-paris5.fr/attachments/download/11217/Cahier_des_recettes_PROJET_L2P1_VERSION_1.01.pdf

[3] Le lien permettant de télécharger et consulter la conception détaillée:
<https://forge.ens.math-info.univ-paris5.fr/attachments/download/11943/Conception%C3%A9taill%C3%A9e.pdf>

[4][5][6][7] Définitions liées aux tests:
<https://www.atlassian.com/fr/continuous-delivery/software-testing/types-of-software-testing>

Index

- application, 4, 5, 14, 26, 31, 32, 35, 56,
64–67, 74, 78
- expérimentation, 5, 7–9, 11, 14, 16, 31,
72–74
- module d'exploitation, 14, 16, 18, 20,
22, 24, 31, 72–74
- test, 5, 26, 74, 78
- tests fonctionnels, 3, 5, 64
- tests unitaires, 3, 26, 74, 79, 80