INSTITUT UNIVERSITAIRE DE TECHNOLOGIE

de BLAGNAC –TOULOUSE II
Département Informatique
1, place Georges Brassens –BP 73-
31703 BLAGNAC Cedex

UNIVERSITÉ ROBERT GORDON

Garthdee House, Garthdee Rd,
Garthdee, Aberdeen AB10 7AQ

Rapport de stage
Rubén Longèque

Rapport de stage effectué du 17/04/2023 au 16/06/2023

À destination de :

- M. Bruel, tuteur pédagogique
- Dr Yang Jiang, tutrice professionnel

Rubén Longèque

Année 2022-2023

INSTITUT UNIVERSITAIRE DE TECHNOLOGIE

de BLAGNAC –TOULOUSE II
Département Informatique
1, place Georges Brassens –BP 73-
31703 BLAGNAC Cedex

UNIVERSITÉ ROBERT GORDON

Garthdee House, Garthdee Rd,
Garthdee, Aberdeen AB10 7AQ

Rapport de stage
Rubén Longèque

Rapport de stage effectué du 17/04/2023 au 08/06/2023

À destination de :

- M. Bruel, tuteur pédagogique
- Dr Yang Jiang, tutrice professionnel

Rubén Longèque

Année 2022-2023

Sommaire

Remerciements

Avant toute chose, je tiens à remercier toutes les personnes qui ont pu m'aider, de près ou de loin, lors de ces deux mois de stage à l'étranger :

Je tiens à remercier l'université Robert Gordon University, pour m'avoir accueilli durant ces deux mois de stage en Écosse.

Tout d'abord, je tiens à remercier Dr Yang Jiang pour m'avoir donné l'opportunité d'avoir pu travailler sur mon sujet de stage. Je tiens aussi à remercier M. Kyle McPherson, assistant du Dr Yang Jiang, pour m'avoir accompagné tout au long de ce projet, d'avoir toujours su apporter son soutien et d'avoir répondu à tous mes questionnements lié au projet.

Ensuite, je souhaiterais remercier mon professeur référent, M. BRUEL, pour ses retours sur mon travail, ainsi que d'avoir pris le temps de communiquer avec moi, via visioconférence, pour suivre mon avancement et me donner de précieux conseils. De même, je voudrais aussi remercier Mme LEGRAND pour ses suggestions en communication orale et écrite, sans lesquelles ce rapport ne serait pas présenté de la manière actuelle.

Je souhaiterais aussi remercier, particulièrement, Mme CLAVEL, pour m'avoir donnée l'opportunité d'effectuer ce stage en Ecosse.

Enfin, j'aimerais remercier, en dernier lieu, tous les autres stagiaires français que j'ai pu côtoyer lors de mon stage, et que je te tiens à remercier pour tous ces instants passés ensemble et que je garderai en mémoire pendant encore longtemps.

Introduction

Cela fait un peu plus d'une dizaine d'années que la technologie de réalité augmentée a été introduite au grand public, et elle continue, encore aujourd'hui, de fasciner et d'être étudié et exploité.

Que ce soit dans les domaines scientifiques, médicaux, industriels ou encore du divertissement, la réalité augmentée permet de rendre existant, palpable ce qui ne l'est pas encore. La majorité du temps, on utilise ce genre de technologie pour prévoir par rapport à ce qu'on ne peut avoir dans l'immédiat.

L'objectif de mon stage était de continuer le développement d'une application mobile Android, afin de pouvoir appliquer différents filtres sur un visage en temps réel.

La finalité de cette application, était de pouvoir s'assurer de la possibilité d'appliquer différents filtres à plusieurs visages, en simultané, et indépendamment de chaque visage.

Je commencerai par présenter le client, puis j'aborderai l'analyse de sa demande et les moyens mis à sa disposition. Par la suite, j'aborderai les différentes réalisations techniques de la méthodologie jusqu'à la présentation des résultats, en passant par le déroulement du projet, avant de finir par dresser les différents bilans techniques, humains et personnels de ce stage.

1. Présentation de l'organisation

L'Université Robert Gordon, communément appelé RGU, est une université publique Écossaise, basée à Aberdeen. Elle obtient son statut d'université en 1992, suite à la *Further and Higer Education Act 1992*, bien qu'elle tire son origine d'un établissement d'enseignement fondé au XVIII^{e} par un reconnu et prospère marchand d'Aberdeen, Robert Gordon.

Aujourd'hui l'université propose à tous ses élèves, des parcours d'études pouvant aller du Bachelor au Doctorat, dans les domaines suivants : Commerce, Sciences sociales, Informatique, Communication, Ingénierie, Art, Sciences médicales, Droit, Infirmerie et pharmacie et enfin Architecture.

Comme toute université, celle-ci doit s'assurer de proposer les meilleures conditions pour la réussite de ses élèves et ainsi d'accroître sa réputation afin d'attirer les meilleurs dans son enceinte. Le journal *The Guardian* classe l'université première, en Écosse, dans 4 domaines d'études en 2017. RGU fait partie des universités avec le plus haut taux d'étudiants employé ou en poursuite d'études, dans les 6 mois suivant l'obtention d'un diplôme, au Royaume-Uni.

En ce qui concerne ma place dans toute cette organisation, il faut savoir que au sein du département Informatique, il existe une partie dédiée aux stages fait par des étudiants étrangers au sein de l'université. Je vais donc résumer ma place actuelle par l'organigramme ci-dessous :

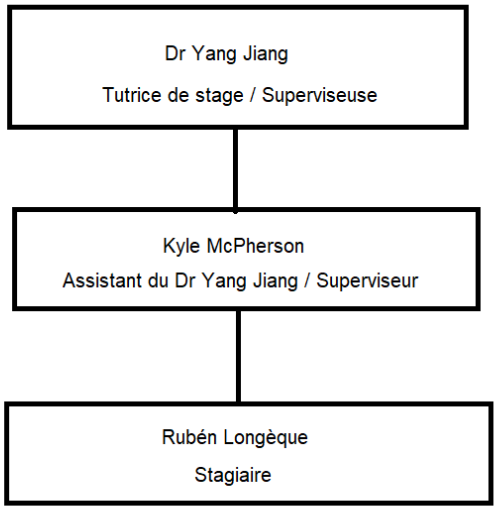

Figure 1 : Organigramme du projet

2. Analyse du besoin

Le sujet de mon stage était l'amélioration d'une application mobile pouvant apposer des filtres, en réalité augmentée, sur des visages via la caméra frontale d'un téléphone. Une fois le filtre posé, l'utilisateur peut alors effectuer une capture d'écran de son téléphone en appuyant sur un bouton, similaire à celui des appareils photo des téléphones modernes. Il était possible pour l'utilisateur de faire apparaître et disparaître, l'interface utilisateur composé d'une barre déroulante avec tous les filtres disponibles et du bouton pour effectuer la capture d'écran.

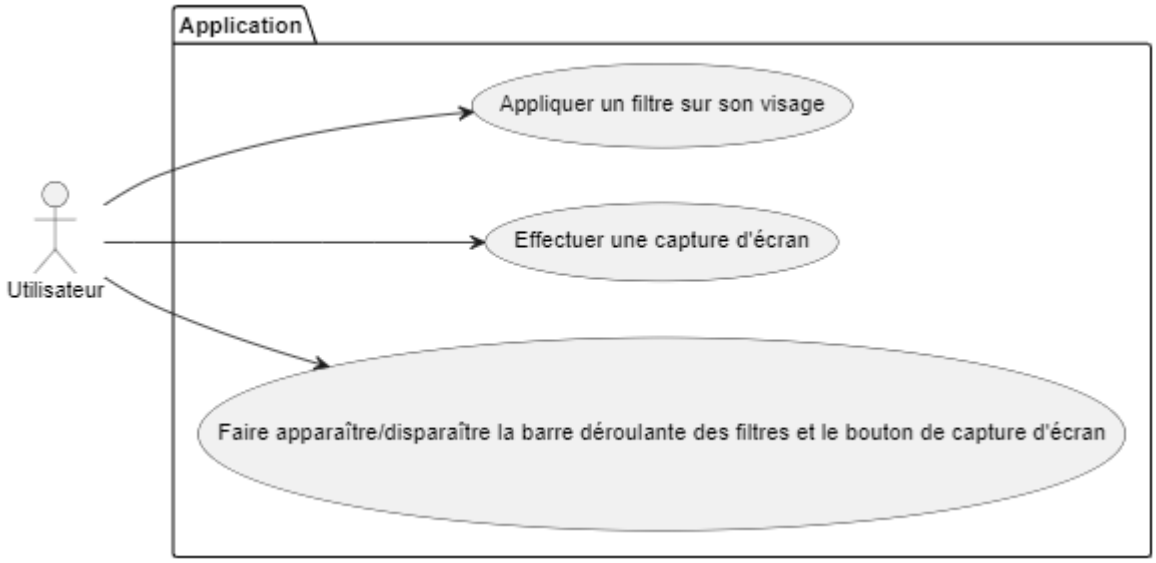

Figure 2 : Diagramme cas d'utilisation originel de l'application

Lorsque je fus introduit à mon sujet, on m'a présenté une première vague de mission à effectuer :

- Réparer un bug provoquant un écran noir comme capture d'écran et non ce que la caméra pouvait voir.
- Changer l'interaction avec l'interface utilisateur. L'objectif est de pouvoir gérer, indépendamment, la barre déroulante et le bouton. Pour le premier, l'utilisateur doit appuyer longuement, et pour le second, l'utilisateur doit tapoter. Cette mission est liée à la suivante.
- Dans une volonté de pouvoir appliquer des filtres sur plusieurs visages, il m'a été demandé de rendre l'interaction avec l'interface utilisateur possible que lorsque que l'on appuie sur le visage, et non partout sur l'écran. Donc l'objectif était de pouvoir délimiter le champ d'action sur le visage détecté seulement.
- Ajouter des trous pour les yeux et la bouche sur les filtres donnés

Nous pouvons alors résumer ce qui était attendu, par la suite, sur ce que pouvait faire l'utilisateur, par le digramme de cas d'utilisation ci-dessous :

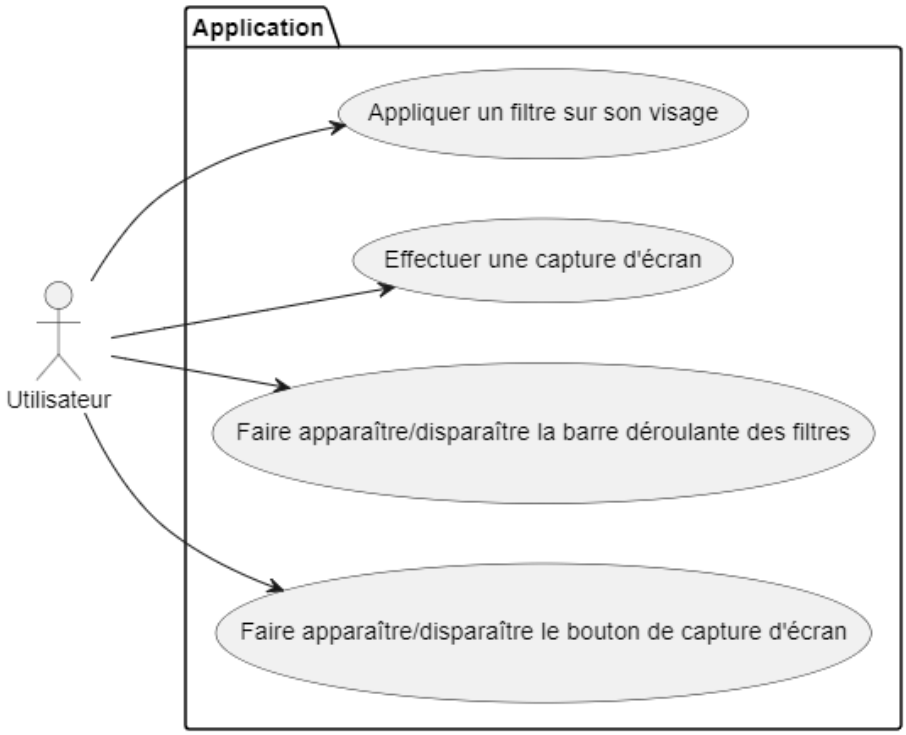

Figure 3 : Diagramme cas d'utilisation attendu de l'application

L’application étant une application Android, j’avais alors une contrainte de langage pour le développement de mon application, qui était le Java. J’ajoute à cela la contrainte de devoir utiliser le framework Mediapipe pour le bon fonctionnement de l’application. Mediapipe est un framework open source développé par Google, conçu pour la création de pipelines de traitement de données multimédia en temps réel.

Une autre contrainte technique à laquelle j'ai été confronté était la nécessité de développer sur une machine ayant, la distribution Linux, Ubuntu 22.04 comme système d'exploitation.

J’ai aussi eu comme contrainte, l’utilisation de Bazel comme outil de construction logicielle. Bazel est un système de compilation et de construction open source qui offre une approche modulaire et évolutive pour la construction d'applications. Il permet de gérer efficacement les dépendances, d'automatiser le processus de compilation et de faciliter la maintenance du code source.

À cela, il s’est ajouté une contrainte matérielle, puisque j'avais besoin d'un appareil mobile fonctionnant sous Android 13.

3. Méthodologie du projet

3.1. Organisation de la gestion du projet

Ce projet a été différent, à bien des égards, de la plupart que j'ai pu effectuer, depuis le début de ma scolarité au sein de l'IUT de Blagnac. Je me suis retrouvé, ici, seul à travailler sur le projet. Ce qu'il faut comprendre, c'est que ce projet a déjà été commencé par un de mes deux superviseurs, et donc, mon rôle était de permettre une avancée sur ce projet, car cette personne, qui avait déjà commencé le projet, était aussi prise sur d'autres projets et donc n'avait pas forcément le temps pour continuer celui-ci.

On peut dire, dans un certain sens, qu'on était une équipe de deux personnes, mais lors de mon stage, j'étais le seul à vraiment travailler dessus, bien que je pouvais compter sur mon superviseur pour m'aider, si besoin. Cela a été le cas, à de nombreuses reprises me permettant de pouvoir trouver solution à mes différents problèmes et donc avancer dans mon travail.

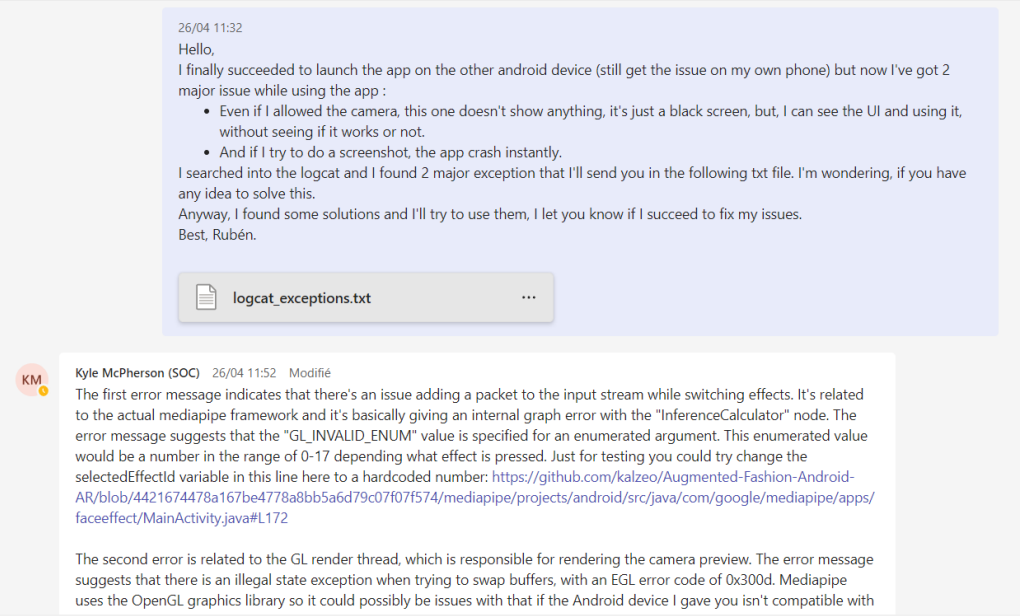

Figure 4 : Exemple de conversation avec mon superviseur

Une autre différence marquante, a été la très grande liberté que j'avais pour organiser mon travail. En effet, je n'avais aucune obligation concernant le lieu de travail, ni la limite des rendus attendus, en dehors de la fin de mon stage concrète, je n'avais aucune réelle pression par rapport à cela. Il y avait toutefois, une seule obligation, c'était un rendez-vous hebdomadaire, effectué le lundi, afin de mettre au courant, mes superviseurs sur mon avancée, et pouvoir discuter du projet, si jamais il

y avait de nouvelles demandes ou pour m'aider sur une tâche que j'étais en train d'effectuer.

Enfin, malgré tout cela, il a été mis en place un dépôt GitHub, afin de pouvoir partager les avancés que j'effectuais dans le projet. Ce partage de code permet, à la fois, de ne pas perdre une trace de ces avancées, et à la fois, à mon superviseur de pouvoir reprendre là où je me suis arrêté pour la suite du projet.

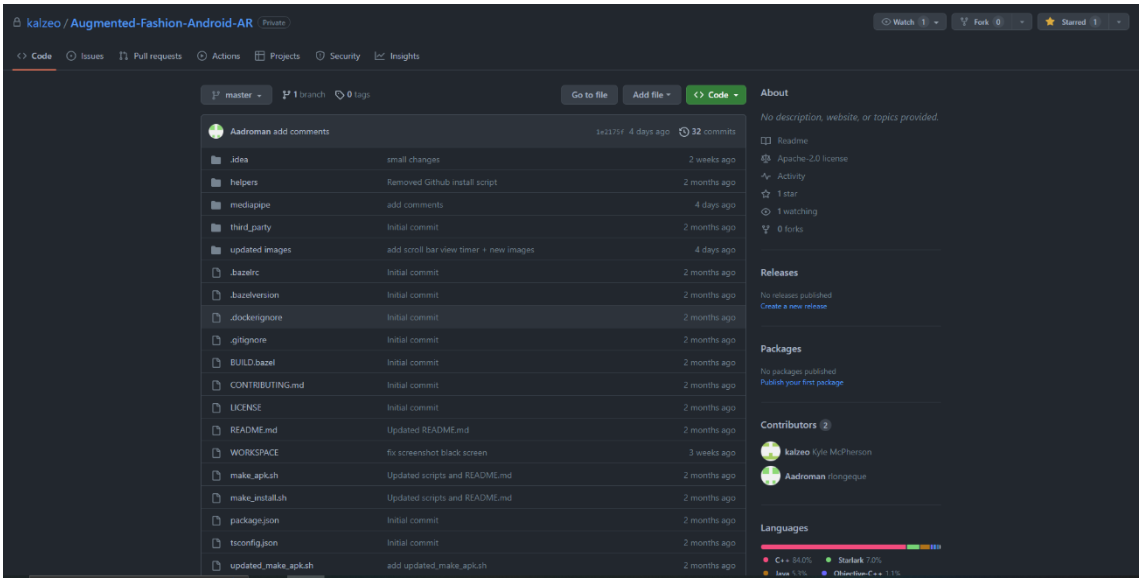

Figure 5 : Extrait du répertoire GitHub du projet

3.2. Justification des choix et des technologies

Dans le cadre de mon stage, j'ai donc été confronté à différentes contraintes techniques auxquelles il était nécessaire d'associer les bons outils pour assurer le bon développement du projet.

Tout d'abord, Android Studio s'est avéré être un outil essentiel pour le développement de mon application. En tant qu'environnement de développement intégré (IDE), il offre une large gamme de fonctionnalités et d'outils spécifiques à Android, ce qui facilite grandement le processus de développement. J'ai utilisé Android Studio pour la conception de l'interface utilisateur, la gestion des ressources, l'écriture du code source, le débogage et le déploiement de l'application sur des appareils virtuels.

Android Studio m'a, surtout, permis de pouvoir modifier le code de l'application, alors en Java. Java est un langage de programmation couramment utilisé pour le développement d'applications Android, offrant une syntaxe claire et une vaste bibliothèque standard.

Étant donné que mon poste de travail utilisait un autre système d'exploitation, j'ai dû configurer une machine virtuelle pour exécuter Ubuntu 22.04, via Oracle VM Virtualbox. Cela m'a permis de travailler dans un environnement compatible avec les outils de développement Android et d'accéder aux fonctionnalités spécifiques nécessaires pour mon projet.

De plus, pour tester l'application, j'avais besoin d'un appareil mobile fonctionnant sous Android 13, j'ai donc dû utiliser mon téléphone personnel. Cela était important pour évaluer les performances de l'application dans un contexte réel, ainsi que pour vérifier sa compatibilité avec la version la plus récente d'Android. En utilisant un appareil physique, j'ai pu identifier et résoudre rapidement les problèmes liés à la compatibilité matérielle et aux fonctionnalités spécifiques à Android 13.

Grâce à Bazel, j'ai pu structurer mon projet de manière efficace, gérer les dépendances externes et automatiser le processus de construction, ce qui a contribué à améliorer la productivité et la qualité du code. De plus, celui-ci est beaucoup plus optimisé sur des lourds projets, comme celui sur lequel j'ai pu travailler.

Bien que ces outils possèdent tous leurs avantages, ils ont également leurs inconvénients. Le projet étant sur un code source de framework, je n'ai pas pu exploiter toutes les fonctionnalités disponibles sur Android Studio. L'utilisation d'une machine virtuelle est très utile, cependant cela ne permet pas d'exploiter correctement les performances de la machine, dite, hôte. Et enfin, bien que Bazel soit très efficace, il est très compliqué à prendre en main pour un néophyte, beaucoup plus que son alternative Gradle, bien que celui-ci ne peut pas de s'appliquer sur d'aussi lourds projets.

4. Déroulement du projet

Maintenant, que vous connaissez tous les outils pour la réalisation du projet, je vais vous expliquer comment s'est déroulé ce projet. Celui-ci va s'articuler autour de deux points majeurs, tout d'abord la mise en place du projet, puis ensuite le développement de l'application.

4.1. Mise en place du projet

Il est important pour moi de raconter la mise en place de ce projet, car il y a eu beaucoup plus d'imprévus qu'espéré. Une fois le répertoire GitHub du projet mis en place, j'ai pu donc commencer à mettre en place tout ce dont j'avais besoin pour pouvoir travailler dessus.

Les premiers imprévus sont apparus lors de la mise en place de la machine virtuelle. En effet, avant même le début de ce projet, dans le cadre de mes études, je possédais une machine virtuelle, déjà existante. Je pensais que si je ne changeais que les caractéristiques telles que la RAM allouée et le stockage alloué, cela serait suffisant. Il s'avérera que l'utilisation d'Android Studio, ainsi que de toutes les extensions nécessaires pour pouvoir travailler sur le projet, prenait beaucoup plus de place qu'espéré. Il était donc nécessaire de recréer une nouvelle machine virtuelle adaptée à mes besoins.

Une fois ce problème réglé, et une fois que tout ce dont j'avais besoin pour travailler fut installé, il était alors temps d'essayer de lancer l'application sur mon téléphone.

Pour pouvoir procéder à cela, nous utilisons des scripts Shell, qui eux-mêmes lancent des commandes Bazel. Ces commandes permettaient d'effectuer deux actions précises. D'une part, créer un APK de l'application, et d'une deuxième part, installer ce même APK sur le téléphone, alors connecté directement à l'ordinateur via un câble USB.

Cependant, cela fit apparaître une autre erreur. Pour la comprendre, il faut que j'explique, brièvement, une partie de ce qui fait un smartphone. Il faut savoir que comme un ordinateur, un smartphone est équipé de ce qu'on appelle un système d'exploitation, ou OS. Il existe un grand nombre de systèmes d'exploitation dans le monde des smartphones, mais celui qui est le plus équipé, et dont il est question dans mon projet, c'est Android, un OS basé sur un noyau Linux.

Cependant, Android ne produit, aucunement, des smartphones. Il existe de nombreuses entreprises, qui ont une part de ce marché. Parmi celles-ci, Xiaomi Corporation, une entreprise chinoise d'électronique et d'informatique. Xiaomi, comme bien d'autres entreprises, proposent leur propre distribution propriétaire du système d'exploitation Android, dans ce cas-ci MIUI. Ces distributions vont donc modifier Android pour être, la plupart du temps, propre à l'identité de l'entreprise.

Donc, lorsque une fois que j'avais installé l'application, via la commande Bazel, et que j'exécutais celle-ci sur mon téléphone, j'avais alors une erreur qui était à cette distribution MIUI. Il était question d'un élément manquant dans un fichier, qui m'était hors d'atteinte, pour pouvoir le modifier. Après quelques recherches, et quelques tentatives pour réparer ce problème. J'ai décidé, alors, de demander un prêt d'un autre appareil sur lequel je pourrai lancer correctement l'application.

J'ai donc obtenu un autre appareil Android pour continuer mon travail. Cependant, celui-ci possédait une version bien antérieure d'Android, précisément Android 6.0, avec impossibilité d'avoir une version plus récente. J'ai donc dû adapter les commandes Bazel pour pouvoir installer l'application sur cet appareil.

J’ai par la suite réussi à lancer l’application, toutefois, la caméra ne fonctionnait pas et lorsque j’essayais d’effectuer une capture d’écran l’application cessait de fonctionner. Cela été dû au fait que l’appareil que j’utilisais, ayant une version trop ancienne d’Android, ne supportait pas OpenGL ES, une API multi-plateforme utilisé pour la conception d'applications générant des images 3D, dérivée de la spécification OpenGL, sous une forme adaptée aux plateformes mobiles ou embarquées telles que les smartphones.

Je fus donc décontenancé face à ce problème matériel, pensant n’avoir aucune possibilité de travailler sur mon projet. J’ai donc décidé de trouver coûte que coûte, une solution pour faire fonctionner l’application sur mon téléphone.

Je finis par trouver une solution, qui ne venait pas réparer mon erreur, mais plutôt la contourner. En effet, au lieu d’utiliser la commande Bazel pour installer l’application via l’APK, je ne générais uniquement que l’APK, puis j’utilisais le service de stockage dans le cloud, Google Drive, pour transférer cet APK et pouvoir le télécharger, par la suite, sur mon téléphone. Et lorsque je faisais cette manipulation, une fois installée, je pouvais alors exécuter, correctement, l’application.

Une fois, cela effectué, je pus, alors, enfin commencer à travailler, près d’une semaine après l’obtention des codes source du projet.

4.2. Développement de l’application

4.2.1. Interaction et réparation de l’interface utilisateur

La première mission sur laquelle je me suis attelé, était de changer la manière dont l’utilisateur pouvait interagir avec l’interface utilisateur et de réparer les possibles problèmes que cela pouvait engendrer.

Dans l’état initial de l’application, l’utilisateur pouvait faire apparaître et disparaître le bouton de capture d’écran en tapotant, quasiment, n’importe où sur l’écran, et faire apparaître et disparaître la barre déroulante des filtres et le bouton de capture d’écran, en restant appuyé, quasiment, n’importe où sur l’écran.

L’objectif premier, était alors de devoir séparer la gestion de la barre déroulante, de celle du bouton.

Une application Android possède ce qu’on appelle des composants, parmi ces composants, il existe celui que l’on appelle une « Activité ». Une « activité », ou activity, représente un écran dans une application Android. Lorsque l’on lance une application, il existe une méthode, par défaut, nommée *onCreate()* qui permet d’instancier cette même activité. On peut dans cette méthode donc associer des objets avec les éléments de mise en page présents sur un fichier XML, ou par exemple, écrire des méthodes, ou des bouts de code pour qu’ils s’exécutent dès le lancement de l’application.

Dans l'existant, on utilise un objet de type *GestureDetector*, à travers duquel on va pouvoir définir, ce qui s'exécute lorsque qu'on tapote l'écran ou lorsque l'on reste appuyé dessus (Annexe n°1).

À l'origine, il était question de faire disparaître la barre déroulante et le bouton en même temps lorsqu'on reste appuyé, et uniquement le bouton lorsqu'on tapote l'écran. J'ai donc séparé les deux, on doit désormais rester appuyé sur l'écran pour faire apparaître ou disparaître la barre déroulante des filtres, et tapoter l'écran pour faire apparaître ou disparaître le bouton de capture d'écran. Ces deux actions appellent *HideMaterialComponent()* et *HideScreenshotComponent()*.

Ces deux méthodes font sensiblement la même chose. Elles font, toutes les deux, appel à la méthode *ToggleVisibility(View v)* (Annexe n°2), en donnant en paramètre la vue (l'élément de mise en page) concernée. Ce que fait cette méthode est très simple : pour tout élément donné en paramètre, elle va définir la visibilité de celle-ci, selon une condition, qui est que si elle est apparente (ou VISIBLE), alors on la rend invisible (ou GONE), sinon elle reste apparente.

Cependant, la disparition du bouton de capture d'écran provoquait un problème. Une fois disparu, la barre déroulante se retrouvait au sommet de l'écran sans raison apparente. Pour régler cette erreur, il a fallu modifier le fichier XML de mise en page de l'application.

Le problème était dû au fait que dans ce fichier, on a précisé que l'élément de la barre déroulante se retrouver collé au bouton, tout en restant au-dessus. Donc, dans un certain sens, son emplacement dépendait d'où se trouvait ce bouton. Ce qui explique que, si celui-ci disparaissait, alors la barre n'avait plus aucune attache et donc se retrouvait au sommet de l'écran (Annexe n°3).

La solution a donc été de lui ajouter un « margin » constant, par rapport au bas du « layout » parent, pour que lorsque le bouton disparaisse la barre ne se retrouve pas en haut mais reste attaché en bas.

4.2.2. Réparation de la capture d'écran

La deuxième mission sur laquelle je me suis attelé, était de rendre fonctionnelle la capture d'écran de l'application.

Tout d'abord, j'ai dû gérer un premier problème lié aux permissions de l'application. Il faut savoir que toute application qui souhaite interagir avec un certain aspect du téléphone, peut ou doit avoir besoin d'une permission.

Pour pouvoir enregistrer une photo dans la galerie d'un téléphone, il faut une permission d'écriture dans du stockage externe, appelé WRITE_EXTERNAL_STORAGE. Dans l'existant, il y avait une méthode, appelé *onRequestPermissionResult()* (Annexe n°4), qui permettait de lancer la méthode de capture d'écran, nommé *TakeScreenshot()*, si la requête renvoyait bien le code pour la permission d'écriture dans un stockage externe.

Or, dans la méthode *TakeScreenshot()*, il y avait une partie du code qui vérifiait à nouveau s'il y avait bien cette permission. J'ajoute, à cela, que cette permission est dépréciée depuis Android 10, donc pour les appareils sous Android 13, il ne s'agit que de vérifications un peu inutiles.

Une fois cette erreur réglée, j'ai pu m'attaquer au véritable problème. Dans l'existant, lorsqu'on appuyait sur le bouton de capture d'écran, celui-ci ne générait qu'un écran noir.

L'idée est qu'il faut arriver à capturer ce qui est affiché sur l'écran pour ensuite le transformer comme une image et l'enregistrer dans la galerie. Déjà, je devais identifier quel était la source que je devais exploiter. Il s'est avéré qu'il existait un objet de type *SurfaceView*, appelé *previewDisplayView*, qui est l'objet qui va montrer ce que les images de la caméra, une fois qu'elles ont été traitées par le framework Mediapipe. Donc c'est ce qui permet d'afficher sur l'application, ce à quoi ressemble notre visage avec un filtre appliqué.

Une fois la source détectée, il faut trouver un moyen de transformer cette source en image. Pour cela, j'ai eu besoin de plusieurs composants. Tout d'abord, il m'a fallu transformer ce qu'affichait *previewDisplayView* en *Bitmap*. Un *Bitmap* est une représentation en mémoire d'une image bitmap, c'est-à-dire une image composée de pixels individuels. Il s'agit d'une grille bidimensionnelle de pixels, où chaque pixel est décrit par ses composantes de couleur. Un *Bitmap* peut représenter une image en niveaux de gris, une image en couleur ou une image avec une transparence. Il contient des informations sur les dimensions de l'image (largeur et hauteur), ainsi que sur les valeurs de couleur de chaque pixel. Ces valeurs de couleur peuvent être stockées sous différentes formes, telles que RVB (rouge, vert, bleu), ARGB (alpha, rouge, vert, bleu) ou niveaux de gris. C'est largement utilisé pour manipuler, afficher et traiter des images dans le développement d'applications.

Pour arriver à cela, j'ai utilisé une classe nommée *PixelCopy*. La classe *PixelCopy* est une classe qui permet de copier les pixels d'une surface ou d'une vue dans un objet Bitmap. Elle offre une fonctionnalité de copie rapide et efficace des pixels, ce qui est utile dans divers scénarios, tels que la capture d'écran, le traitement d'image ou la création de miniatures. La classe *PixelCopy* fournit une méthode statique appelée *request*(), qui permet de demander une copie des pixels à partir d'une source spécifiée vers un objet Bitmap.

J'ai créé une méthode, nommée *checkScreenshot()* (Annexe n°5), qui prend en paramètre un objet *Bitmap* qui correspond au bitmap, sur lequel, on va copier *previewDisplayView*, un objet *Activity* qui va correspondre à la classe *Activité* dans laquelle on se trouve, un objet *Rect* qui va permettre de définir les dimensions de la copie, et enfin un objet *Consumer<Bitmap>* (Annexe n°6).

Je me dois de faire un petit aparté sur cette classe *Consumer<Bitmap>*. La classe Consumer<Bitmap> est une interface fonctionnelle de Java. Elle représente un consommateur qui accepte un argument de type Bitmap et effectue une opération dessus, sans renvoyer de résultat. L'interface Consumer<Bitmap> définit une seule méthode, appelée *accept*(), qui prend en paramètre un objet Bitmap et ne renvoie rien. Cette méthode peut être implémentée par une classe ou une expression lambda pour définir le comportement à exécuter sur le *Bitmap* donné.

En utilisant cette méthode, je fais appel à la méthode statique *request()* de *PixelCopy*, en lui donnant en paramètre : la source à copier de type *SurfaceView*, L'aire de la zone à copier via l'objet *Rect*, le bitmap de destination où sera copié la source, un callback qui exécutera la méthode *accept()* du Consumer, si la copie est réussie, et enfin un handler, pour appeler le callback, une fois la copie terminée.

J'ai pu donc réussir la copie de *previewDisplayView* dans le *Bitmap* donné. Maintenant, il me fallait pouvoir convertir ce bitmap en un fichier JPEG, et l'enregistrer dans la galerie.

Concrètement, j'ai réutilisé le code de l'existant, que je l'ai séparé dans plusieurs méthodes, notamment pour être utilisable au sein du *Consumer<Bitmap>*. Mais ce que font ces méthodes, l'existant était déjà capable de le faire.

L'idée est que j'ai fait appel, d'abord, à un objet de type *ContentValues* pour spécifier les valeurs de colonnes à insérer dans la galerie. Les valeurs comprennent le type MIME de l'image (dans cet exemple, "image/jpeg"), la date d'ajout de l'image, le nom d'affichage du fichier, le chemin relatif où l'image doit être enregistrée (dans le répertoire Pictures), et le marquage de l'image comme en cours de traitement. Il insère les valeurs dans la galerie via le *ContentResolver* et récupère l'*URI* du nouvel enregistrement. Ensuite, il met à jour les valeurs pour marquer l'image comme terminée et effectue une mise à jour via le *ContentResolver*. Un message toast est affiché pour indiquer que la capture d'écran a été enregistrée avec succès. Enfin, la méthode *OpenScreenshot()* est appelée pour ouvrir l'image enregistrée. La méthode *saveImageToStream()* est une méthode utilisée par *saveImage()* pour effectuer la compression et l'enregistrement réel de l'image dans le flux de sortie fourni (Annexe n°7).

4.2.3. Modification des filtres

Cette mission était facultative, mais étant relativement simple, cela aurait été dommage de ne pas pouvoir la faire.

Tout d'abord, j'ai obtenu un nouvel ensemble de filtres, ce qui m'a obligé à modifier mes fichiers de Build, ainsi que le code de la méthode *onCreate()* de l'activité afin de, notamment, s'adapter au nombre de filtres du nouveau set.

Avant toute chose, il y a quelques points à éclaircir. D'abord, les fichiers de build de Bazel. Ce sont des fichiers utilisés pour décrire la configuration et la construction d'un projet à l'aide de Bazel. Parmi ces fichiers, il y a toujours, à minima,

un fichier BUILD. Il s'agit du fichier principal de configuration de build dans Bazel. Il contient des règles de construction pour les différentes cibles du projet, telles que les bibliothèques, les exécutables, les tests, etc. Dans ce fichier, vous spécifiez les dépendances, les sources, les options de compilation et d'autres paramètres nécessaires pour générer les cibles de build. C'est, notamment, dans le fichier BUILD lié à la classe « Activité » que nous assignerons l'emplacement des fichiers *.pngblob* correspondant aux filtres.

Il m'a fallu aussi modifier un autre type de fichier, mais qui pour le coup à un aspect plus technique. Il s'agit d'un fichier *.pbtxt*, et plus précisément celui lié au modules « face effect » de Mediapipe. Les fichiers *.pbtxt*, abréviation de "Protocol Buffers Text Format", sont des fichiers de texte utilisés pour décrire la structure et les paramètres des modèles et des graphiques dans le contexte de Bazel et de certains outils associés. Dans le contexte de Bazel, les fichiers BUILD peuvent faire référence à des modèles et à des graphiques utilisés pour la génération et l'exécution de cibles de build. Les fichiers .pbtxt sont utilisés pour décrire ces modèles et graphiques. Ils suivent une syntaxe spécifique basée sur le format de sérialisation de données de Protobuf. Les fichiers .pbtxt contiennent des déclarations d'objets avec des champs et des valeurs correspondantes. Ces objets peuvent représenter des règles de construction, des dépendances, des configurations de compilation, des paramètres d'exécution, etc. Les champs et les valeurs sont généralement spécifiés sous forme de paires clés-valeurs, où la clé est le nom du champ et la valeur est la valeur associée. Les fichiers .pbtxt permettent de décrire de manière structurée et lisible par l'homme la configuration des modèles et des graphiques utilisés dans le processus de construction avec Bazel. Ils sont souvent utilisés pour spécifier les paramètres et les options spécifiques des règles de construction, ce qui permet de personnaliser et d'adapter le comportement des cibles de build en fonction des besoins du projet. Et dans le cas de notre projet, il y avait un « nœud » (ou node en anglais) qui permettait d'utiliser un outil pour permettre l'application des filtres sur le visage reconnu, et on lui donnait un ensemble d'options qui sont les fichiers *.pngblob* des filtres.

Une fois cette partie traitée, je devais, ensuite, ajouter des trous pour les yeux et la bouche du visage. Pour cela, j'ai utilisé l'outil de manipulation et de retouche d'image GIMP, et une image du modèle canonique du visage utilisé par Mediapipe.

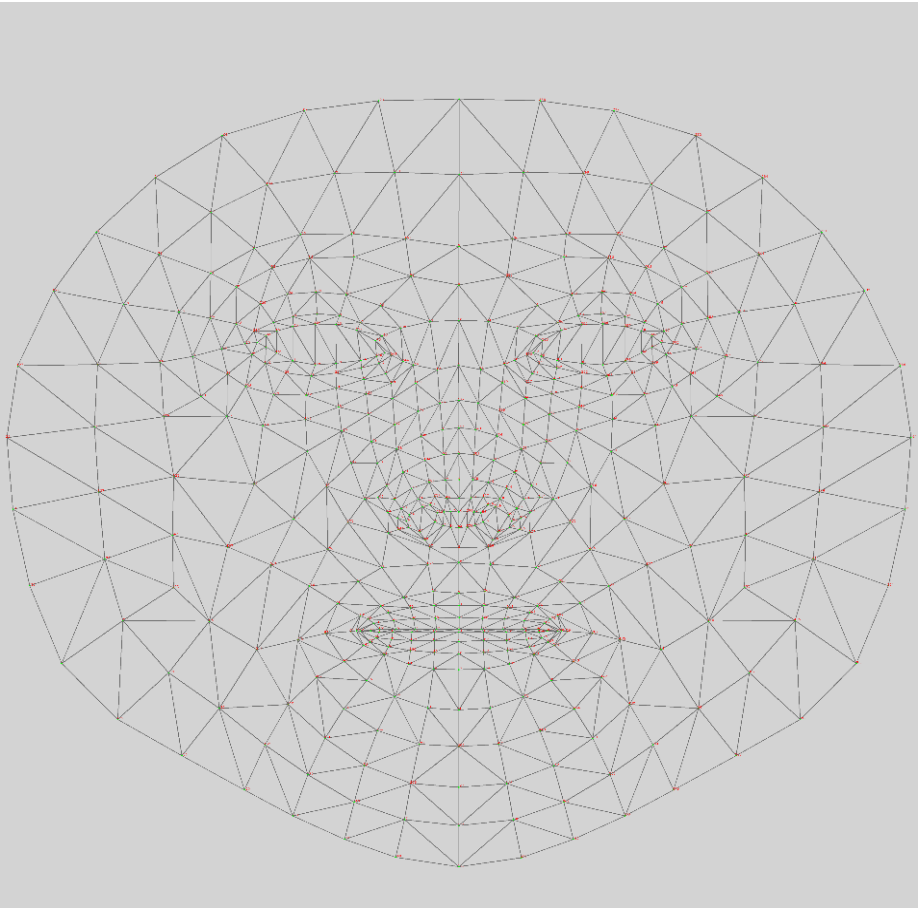

Figure 6 : Modèle canonique du visage utilisé par Mediapipe

Le fait que les fichiers soient, des dérivés des fichiers *.png*, me laissait la possibilité d'utiliser GIMP pour créer un fond transparent sur certaines parties des filtres. Le modèle canonique représente comme est structuré la modélisation du visage par le framework Mediapipe. Les points d'intersection entre deux lignes forment alors un point de repère accompagné d'un numéro pour l'identifier. Ici, cela m'a surtout servi pour délimiter les zones des yeux et de la bouche pour appliquer le même traitement à tous les filtres.

Figure 7 : Exemple de nouveau filtre avant modification

Figure 8 : Exemple de nouveau filtre après modification

4.2.4. Limiter l'espace d'interaction au visage identifié

Cette quatrième mission, sur laquelle je me suis attelé, avait pour but de limiter la zone, pour gérer l'interface utilisateur, uniquement au visage de l'utilisateur.

Pour arriver à cela, il a fallu diviser le tout en deux objectifs distincts : dans un premier temps, obtenir les coordonnées de la zone d'action de l'utilisateur, et dans un deuxième temps, délimiter la zone du visage à l'aide des coordonnées des points de repères du modèle canonique du visage.

L'obtention des coordonnées consistait à utiliser une méthode native de la classe *View* représentant les éléments d'une interface utilisateur (ou UI) avec lequelle l'utilisateur peut interagir. Lors de l'instanciation de l'application, nous avons associé des éléments de mise en page, appelés des « vues », à différents objets *View* ou qui hérite de cette classe, notamment l'élément qui représente l'ensemble de l'écran.

Donc, à cet objet, nous allons associer un « listener ». Un « listener » est un composant logiciel utilisé pour détecter et réagir à des événements ou à des changements d'état dans un système, et il est souvent utilisé pour gérer des interactions utilisateur, dans le cadre du développement d'application. Les « listeners » fonctionnent selon le principe de la programmation événementielle. Ils sont généralement attachés à des composants d'interface utilisateur tels que des boutons, des vues ou des éléments interactifs, et attendent que ces composants génèrent des événements.

La classe *View* possède une sorte de « sous-classe » interface appelée *OnTouchListener()*, et qui possède une méthode appelée *onTouch()*, qui permet d'identifier le type d'action survenu sur un objet *View* donné (Annexe n°8).

Donc, l'idée est que lorsque l'utilisateur effectue une action où il viendrait à toucher l'écran, la méthode va stocker dans un tableau de *float* de taille 2, les coordonnées X et Y de là où a interagi l'utilisateur. Une fois ces coordonnées stockées, elles pourront être utilisées, pour comparer avec les coordonnées qui vont délimiter le visage de l'utilisateur.

Pour effectuer la délimitation du visage, j'ai dû modifier le fichier .pbtxt du module « face effect » de Mediapipe, car celui-ci ne me permettait pas, par défaut, d'obtenir les points de repère (ou « landmarks » en anglais) du modèle canonique du visage, utilisé par Mediapipe (Annexe n°9).

Ces modifications permettent de définir ce qui peut être appelé au sein des différents « callback » possible. Un « callback » est une fonction ou une méthode qui est passée en tant que paramètre à une autre fonction ou méthode. Le « callback » est ensuite invoqué ou exécuté à un moment donné par la fonction ou la méthode à laquelle il a été transmis. Le concept de « callback » est couramment utilisé en programmation asynchrone ou événementielle, où les opérations peuvent prendre du temps et doivent être exécutées de manière non-bloquante. Au lieu d'attendre le résultat d'une opération, on peut fournir une fonction de « callback » qui sera appelée lorsque l'opération est terminée ou lorsqu'un événement se produit. Dans notre cas, nous allons associer ce « callback » à l'objet *FrameProcessor* du projet, qui est l'objet qui va traiter les images en temps réel de la caméra (Annexe n°10).

Tout cela nous permettra d'obtenir un paquet avec l'ensemble des points de repères, et donc de les exploiter, notamment, en les enregistrant au sein des variables globales pour être utilisable lors de la comparaison avec les coordonnées de la zone d'action de l'utilisateur. Ce même paquet est constamment renouvelé par le callback ce qui permet d'avoir les coordonnées en temps réel des points de repères du visage.

Il est important d'apporter quelques précisions sur cette méthode. Je n'ai cherché qu'à utiliser que quatre points de repère parmi tous ceux disponible. J'ai fait ce choix pour deux raisons principales, tout d'abord, en termes de performances, c'est toujours mieux d'optimiser au plus ce genre de méthode pour éviter trop de calcul, simultanément, pour le processeur du téléphone, et enfin, je tenais à obtenir une sorte de boîte qui entourerait le visage de l'utilisateur pour délimiter au mieux la zone d'interaction possible. Et donc pour cela, j'ai utilisé l'axe Y des points les plus en haut et en bas du visage, ainsi que l'axe X des points les plus à droite et à gauche du visage. Ensuite, j'ai multiplié le sommet par la hauteur de la résolution de l'écran, soit 1920. Le point le plus à gauche, par la largeur de la résolution, soit 1080. Le point le plus bas, par la valeur de la limite basse de l'écran via l'objet *View*, et le point le plus

à droite, par la valeur de la limite à droite de l'écran, aussi, via l'objet *View.*

Une fois tout cela obtenu, j'ai dû modifier les méthodes d'interaction avec l'interface utilisateur, en ajoutant une grande condition qui vérifie que l'axe X de la zone d'action de l'utilisateur est inférieur à l'axe X de la limite à droite, et supérieur à la limite à gauche. Et, où l'axe Y de la zone d'action de l'utilisateur est inférieur à l'axe Y de la limite basse, et supérieur à la limite haute (Annexe n°11). Cela s'explique, car le point 0 de l'axe X se place tout à gauche de l'écran, et l'axe Y tout en haut.

Une fois, ces quatre premières missions effectuées, j'ai eu deux nouvelles missions principales, cependant, je n'ai pu réussir à n'en faire qu'une seule des deux.

4.2.5. Ajout d'un minuteur pour gérer la barre déroulante des filtres

L'objectif de cette mission était d'ajouter un « timer », ou minuteur, pour faire disparaître la barre déroulante des filtres, une fois que l'utilisateur a cliqué sur un des boutons de la barre, au bout d'un temps défini, qui ici sera 5 secondes.

Pour arriver à cela, j'ai dû utiliser deux classes Java qui se complémentent : *Timer* et *TimerTask*. *Timer* est une classe fournie par Java qui permet de planifier et d'exécuter des tâches à des intervalles spécifiés. Elle possède une méthode appelée *schedule()* qui prend en paramètre, un objet de la classe interface *TimerTask*, et qui permet d'exécuter la tâche contenue dans cet objet au bout d'un temps donnée, en milliseconde. *TimerTask*, étant une classe interface, j'ai donc dû « réécrire » sa méthode *run()* qui va définir ce qui sera exécuté une fois la limite de temps atteint.

Cependant, j'ai eu affaire à un premier problème. Il faut savoir que *TimerTask* exécute sa tâche dans un nouveau thread, et donc en dehors du thread principal de l'application. Ce qui fait que je ne pouvais pas interagir sur les éléments de l'application hors du thread principal. La solution a été d'utiliser une méthode native de la classe *Activity*, qui se nomme *runOnUiThread()*. Cette méthode permet d'exécuter du code sur le thread de l'interface utilisateur (UI thread) à partir d'un thread différent. Ce qui a été très utile pour pouvoir finaliser cette mission.

Cependant, un nouveau problème est apparu à la suite de cela. Lorsque l'utilisateur clique sur un autre bouton, une fois après avoir choisi un filtre, dans le temps donné avant la disparition de la barre, cela va faire cumuler les objets *TimerTask*. Donc, cela provoque une disparition incontrôlée de cette vue, lorsqu'on ne souhaite pas qu'elle s'effectue.

Pour régler cela, j'ai mis en place une variable booléenne globale qui prend en compte le fait que la tâche a été planifié ou non. Si la variable est fausse alors nous

la mettons vrai, pour la marquer comme en cours, puis nous lançons le *TimerTask* normalement. Si la variable est vrai alors nous arrêtons le timer puis nous faisons une nouvelle instance de la classe *Timer* à laquelle nous assignons à nouveau le *TimerTask* attendu, cela permet d'éviter que plusieurs instances de *TimerTask* se superpose l'une à l'autre, et donc, cela provoque une sorte de réinitialisation du temps du *TimerTask* pour qu'elle se relance et fonctionne normalement tant qu'un autre bouton n'a pas été cliqué, et donc, provoquer une nouvelle réinitialisation du temps (Annexe n°12).

5. Résultats du projet

Une fois mon travail terminé, au sein de ce projet, il ne manque plus que de vérifier que tout ce que j'ai développé était bien fonctionnel, et les résultats obtenus sont des plus satisfaisants.

L'objectif initial était de rendre l'application, sur laquelle j'ai travaillé, fonctionnelle. Donc, tout d'abord, il est important de préciser que toute l'application se tient sur une seule et même « page », il n'y a pas d'autres accès menant à d'autres fonctionnalités.

Quand on arrive sur l'application, il y a la caméra activée avec un premier filtre, par défaut, la barre déroulante des filtres et le bouton pour effectuer la capture d'écran. La première interaction qui viendra à l'esprit de l'utilisateur sera de modifier le filtre appliqué à son visage.

Figure 9 : Exemple de filtre applicable

Figure 10 : Exemple de filtre applicable

Il est possible, pour l'utilisateur, de gérer les différents éléments avec lesquels il peut interagir. Par exemple, la barre déroulante peut soit disparaître, au bout d'un certain temps, une fois un filtre sélectionné, ou alors c'est l'utilisateur lui-même qui le fait disparaître ou apparaître. Quant au bouton de capture d'écran, c'est l'utilisateur lui-même qui décide de sa visibilité ou non sur l'application.

Figure 38 : Barre déroulante rendue invisible par l'utilisateur ou le minuteur

Figure 29 : Message d'erreur pour la gestion de la barre déroulante

Figure 20 : Bouton de capture d'écran rendu invisible par l'utilisateur

Figure 11 : Message d'erreur pour la gestion du bouton de capture d'écran

La réussite de la gestion des éléments interactifs, via la zone d'interaction délimitée au visage, est des plus importantes pour la possible évolution du projet. En effet, avec une volonté de pouvoir appliquer des filtres sur plusieurs visages en même temps, et indépendamment des uns et des autres, il est nécessaire de pouvoir gérer les filtres à appliquer selon, par exemple, le visage sélectionné.

Enfin, il est possible, désormais, pour l'utilisateur d'effectuer une vraie photographie avec le filtre appliqué et de pouvoir la visualiser, sans que celle-ci ne soit qu'un écran noir.

Figure 45 : Exemple de capture d'écran obtenu via l'interaction avec le bouton

Mon travail étant complètement présenté, quels bilans en ai-je tiré ?

6. Bilans

Maintenant, que mon projet a été entièrement présenté, il est temps de dresser les différents bilans de ces 2 mois de stage. On abordera d'abord le bilan technique qui permettra de faire le point sur mes objectifs, puis on abordera le bilan humain qui traitera la gestion de ce projet et quels enseignements en tirer, et enfin le bilan personnel, où j'exprimerai tout ce que j'ai pu apprendre, personnellement, au cours de ce projet.

6.1. Bilan technique

Au vu des différents objectifs qui ont été mis en place pour ce projet, le bilan technique est plus que satisfaisant. En effet, toutes les fonctionnalités demandées, dans un premier temps, ont été développées. Ce qui a mené à l'arrivée de nouvelles demandes que j'ai pu, quasiment, complété.

En effet, sur les deux dernières fonctionnalités demandés, je n'ai pu développer celle pour appliquer les filtres sur plusieurs visages en simultanés. Car, après de nombreuses recherches, il s'est avéré que le module utilisé, du framework Mediapipe, ne pouvait supporter une telle demande, à l'heure actuelle.

Bien que l'objectif était d'essayer de rendre cela possible, je me suis retrouvé dans l'incapacité de pouvoir obtenir la moindre documentation utile, pour pouvoir trouver une solution.

Au-delà de cette petite déception, j'ai toujours su répondre aux attentes de mes superviseurs, tout en étant assez efficace, car malgré le retard que j'ai pu prendre en début de projet, j'ai tout de même réussi à finaliser 90 % de ce que l'on m'a demandé au cours de ce projet.

Pour ce qui est de l'apprentissage technique, j'ai pu, principalement, découvrir le framework Mediapipe, dont je ne connaissais absolument pas auparavant, et qui m'a introduit au développement d'application utilisant de la réalité augmentée. De même pour l'outil de construction logicielle Bazel. J'ai aussi commencé à apprendre comment manipuler et gérer des fichiers de configurations complexes comme les fichier *.pbtxt*, ou les fichiers BUILD de Bazel.

6.2. Bilan humain

La gestion de ce projet était bien différente de ce que j'ai pu effectuer, depuis mon arrivée dans les études supérieures. Le fait que je sois véritablement seul à développer le projet, sur l'instant, mais que mon travail sera, par la suite, réutilisé par la suite pour être finalisé, est très changeant de mes projets, où majoritairement, je développais des fonctionnalités en même temps que mes collaborateurs en développaient d'autres.

Cependant, je ne suis jamais senti isolé, ou ignoré par mes superviseurs. J'étais en constante communication, avec au moins l'un d'entre eux. De plus, les réunions hebdomadaires me permettaient de garder un cap sur le projet, ainsi que de montrer mon avancement dans mon développement.

Donc, bien que je n'effectuasse pas mon travail en équipe, à proprement parlé, je n'ai jamais eu cette impression d'être seul, tant l'implication de mes superviseurs pour m'aider dans ce projet a été importante.

6.3. Bilan personnel

Dans ce projet, j'avais principalement un véritable rôle de développeur. J'entends par là, que j'ai reçu des directives, des objectifs, et que c'était à moi de trouver, comment avec les outils à ma disposition, je pouvais répondre à ces attentes.

Ce que je retiendrai de ce stage, c'est que je me suis rendu compte que j'étais quelqu'un qui faisait preuve d'abnégation pour pouvoir régler un problème par soi-même, sans pour autant s'isoler dans son coin, lorsque j'étais coincé, je n'hésitais pas à demander de l'aide pour pouvoir trouver comme un « nouveau souffle » dans mes recherches. Cela m'a permis de me sortir plus rapidement, que prévu, dans de nombreuses situations.

Au-delà des découvertes techniques que j'ai citées auparavant, j'ai surtout continué d'approfondir mes connaissances en développement d'applications mobiles, que j'avais acquis durant cette année scolaire.

Je pense que ce stage, à l'étranger, m'a aussi beaucoup aidé pour être plus avenant envers les autres et à mieux comprendre comment je dois m'organiser pour être des plus efficaces. En effet, je suis fier d'avoir réussi à ne pas avoir cédé à la tentation de ne travailler que très peu, en profitant de la certaine liberté de gestion du projet que m'ont laissé mes superviseurs. J'ai réussi à travailler efficacement, tout en gérant, de la manière la plus agréable possible pour moi-même, mon emploi du temps.

Conclusion

Il est temps de conclure ce rapport autour de ce projet très enrichissant d'un point de vue tant humain, que technique ou que professionnellement.

L'objectif de mon stage était de rendre fonctionnelle une application Android, utilisant une technologie de réalité augmentée, pour appliquer des filtres, en temps réel, sur le visage d'un utilisateur. Mon projet s'est divisé en quatre grandes missions principales : interaction et réparation de l'interface utilisateur, réparation de la capture d'écran, limiter l'espace d'interaction au visage identifié, et l'ajout d'un minuteur pour gérer la barre déroulante des filtres. À cela s'est ajouter une mission facultative : la modification des filtres. J'ai pu répondre à l'ensemble de ces fonctionnalités attendues. Cependant, j'ai aussi eu comme demande de pouvoir appliquer les filtres sur plusieurs visages simultanément, mais les outils que j'avais sa disposition ne me permettait de répondre à cette demande.

Pour ce qui est des perspectives autour de ce projet, l'application a encore de nombreux points d'amélioration, comme l'interface utilisateur qui pourrait être améliorée, ou encore la possibilité d'appliquer des filtres sur plusieurs visages. Il s'agit même du réel objectif principal de ce projet : trouver comment améliorer l'outil utilisé pour qu'il puisse répondre à cette demande. Mon travail sera repris, par la suite, par mes superviseurs pour faciliter leurs recherches, leur permettant de ne pas s'attarder sur les fonctionnalités que j'ai pu développer.

De plus, ce projet m'a permis d'acquérir un bagage technique réutilisable dans le monde professionnel, principalement dans le domaine du développement d'application mobile. Ce qui est très intéressant, au vu de la place qu'on les smartphones dans nos quotidiens.

Si c'était à refaire, je penseais faire, à nouveau, les mêmes choix et organiserai mon travail à l'identique, car je suis très satisfait et fier du projet que j'ai pu livrer à mes superviseurs. Je pense être, désormais, capable de produire à nouveau un travail de cette ampleur, et de cette qualité.

Lexique

Framework : Un framework est un ensemble de ressources préétablies et organisées qui facilite le développement d'applications en fournissant des fonctionnalités communes, une structure cohérente et des abstractions réutilisables. Il permet aux développeurs de se concentrer sur la logique métier spécifique de leur application plutôt que de passer du temps à résoudre des problèmes techniques courants.

Pipelines : Un pipeline est un flux de traitement de données qui comprend des étapes ou des modules interconnectés. Il est utilisé pour traiter des flux de données en temps réel et effectuer des tâches de vision par ordinateur. Les pipelines de Mediapipe permettent aux développeurs de créer des applications de traitement d'images et de vidéos avancées en utilisant des composants configurables et modulaires.

Outil de construction logicielle : Un outil de construction logicielle est un logiciel qui automatise le processus de création d'un logiciel à partir de son code source. Il permet de gérer les dépendances, de compiler les fichiers source, d'exécuter des tests et de générer des fichiers binaires prêts à être déployés. C'est un élément essentiel du développement logiciel moderne qui facilite la construction, la gestion et la livraison de logiciels de manière efficace et reproductible.

APK : Un fichier APK (Android Package) est le format de fichier utilisé par le système d'exploitation Android pour distribuer et installer des applications sur les appareils Android. Un APK contient tous les éléments nécessaires à l'exécution d'une application Android, y compris le code compilé (fichiers de classes), les ressources (images, fichiers de configuration, etc.), les fichiers de manifeste, les bibliothèques partagées et autres fichiers nécessaires à l'exécution de l'application.

Distribution propriétaire : Une distribution propriétaire fait référence à un logiciel ou à un système d'exploitation qui est développé et détenu par une entité spécifique, telle qu'une entreprise ou un individu, et dont le code source n'est pas librement accessible ou modifiable par le public. Dans une distribution propriétaire, l'entité propriétaire conserve généralement les droits exclusifs sur le logiciel et peut imposer des restrictions sur son utilisation, sa modification et sa redistribution.

API : Une API (Application Programming Interface) est un ensemble de règles et de protocoles qui permettent à différentes applications logicielles de communiquer et d'interagir les unes avec les autres. Elle définit les méthodes, les structures de données et les conventions utilisées pour faciliter l'échange d'informations entre les différents composants logiciels.

XML : XML (eXtensible Markup Language) est un langage de balisage utilisé pour structurer et organiser les données de manière lisible par les humains et les machines. Il s'agit d'un format de données textuelles qui permet de représenter des informations hiérarchiques sous la forme de balises imbriquées.

Margin : Un « margin », en français marge, se réfère à l'espace vide situé à l'extérieur des limites d'un élément dans une mise en page ou une conception graphique. Il s'agit de l'espace ajouté autour d'un élément pour créer une séparation visuelle entre cet élément et les autres éléments environnants.

Layout : Un layout, dans le contexte du développement logiciel, se réfère à la structure et à l'arrangement visuel des éléments d'interface utilisateur d'une application ou d'une page web. Il définit comment les différents composants graphiques tels que les boutons, les champs de saisie, les images et autres éléments sont positionnés et alignés les uns par rapport aux autres.

MIME : Le type MIME (Multipurpose Internet Mail Extensions) est un standard de codage utilisé pour identifier la nature et le format des fichiers transmis sur Internet. Il a été initialement développé pour les e-mails afin de permettre l'envoi de pièces jointes de différents types de fichiers. Il permet de spécifier la catégorie générale du contenu et son sous-type spécifique, ce qui facilite l'interprétation et le traitement approprié des fichiers par les applications et les serveurs.

URI : Un URI (Uniform Resource Identifier) est une chaîne de caractères utilisée pour identifier de manière unique une ressource sur Internet. Il est couramment utilisé pour spécifier l'emplacement d'une ressource telle qu'une page Web, un fichier, un service Web, une image, etc.

Thread : Un thread est une unité d'exécution d'un programme informatique. C'est une séquence d'instructions qui peut être exécutée indépendamment et simultanément avec d'autres threads au sein d'un même processus. Un processus peut contenir plusieurs threads qui travaillent de manière concurrente, permettant ainsi l'exécution parallèle de différentes tâches.

Bibliographie

Liste des Annexes

Table des illustrations

Annexes

Annexe n°1 - Extrait du code concernant la gestion de l'interaction sur l'écran :

```
// We use the tap gesture detector to switch between face effects. This allows users to try
// multiple pre-bundled face effects without a need to recompile the app.
// We use the tap gesture detector to hide the UI components.
tapGestureDetector =
        new GestureDetector(
                this,
                new GestureDetector.SimpleOnGestureListener() {

                    // TODO: Long press on face so that they can change the effect
                    @Override
                    public void onLongPress(MotionEvent event) {HideMaterialComponents();}

                    // DONE: Hide UI when screen is tapped, ideally it should only hide the scree
                    @Override
                    public boolean onSingleTapUp(MotionEvent event) {
                        HideScreenShotComponent();
                        return true;
                    }
                });
```

Annexe n°2 - Méthode ToggleVisibility() :

```
/**
 * Toggle the visibility of a View to show or hide it
 * @param v - The View to be hidden or shown.
 */
private void ToggleVisibility(View v) {
    v.setVisibility(v.getVisibility() == View.VISIBLE ? View.GONE : View.VISIBLE);
}
```

Annexe n°3 - Extrait du code de mise en page concernant la barre déroulant avec mise en avant des éléments nécessaires à la compréhension du problème et de sa solution :

```
<!-- Horizontal scroll view for preview materials -->
<HorizontalScrollView
    android:id="@+id/horizontal_scrollview"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_above="@id/screenshot_button"
    android:layout_alignParentBottom="true"
    android:layout_marginBottom="125dp"
    android:scrollbars="none">

    <LinearLayout
        android:id="@+id/preview_materials_layout"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:orientation="horizontal" />
</HorizontalScrollView>

<!-- Screenshot button -->
<Button
    android:id="@+id/screenshot_button"
    android:layout_width="80dp"
    android:layout_height="80dp"
    android:layout_alignParentBottom="true"
    android:layout_centerHorizontal="true"
    android:layout_marginBottom="20dp"
    android:background="@drawable/white_circle"
    android:text="" />
```

Annexe n°4 - Méthode onRequestPermissionResult() :

```
@Override
public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[] grantResults) {
    super.onRequestPermissionsResult(requestCode, permissions, grantResults);
    if (requestCode == REQUEST_CODE_PERMISSION_WRITE_EXTERNAL_STORAGE) {
        if (grantResults.length > 0 && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
            TakeScreenshot();
        } else {
            Toast.makeText(this, "Permission to save screenshot denied", Toast.LENGTH_LONG).show();
        }
    }
}
```

Annexe n°5 - Code source de la méthode checkScreenshot() :

```
/**
 * Copy into a given bitmap, the return value from PixelCopy.request and execute an operation from the given consumer
 * @param bitmap
 * @param activity
 * @param viewRect
 * @param onSuccess
 */
private void checkScreenshot(Bitmap bitmap, Activity activity, Rect viewRect, Consumer<Bitmap> onSuccess) {
    //Requests for the display content of a SurfaceView to be copied into a provided Bitmap.
    //The contents of the source will be scaled to fit exactly inside the bitmap.
    // The pixel format of the source buffer will be converted, as part of the copy, to fit the the bitmap's Bitmap.Config.
    // The most recently queued buffer in the SurfaceView's Surface will be used as the source of the copy.
    PixelCopy.request(previewDisplayView, viewRect, bitmap,
            (copyResult) -> {
                if (copyResult == PixelCopy.SUCCESS) {
                    onSuccess.accept(bitmap);
                } else {
                    throw new RuntimeException("problem with taking a screenshot");
                }
                bitmap.recycle();
            },
            new Handler(Looper.getMainLooper())
    );
}
```

Annexe n°6 - Code source de l'objet Consumer<Bitmap> utilisé :

```
Consumer<Bitmap> check= new Consumer<Bitmap>() {//Represents an operation that accepts a single input argument and returns no result.
    @Override
    public void accept(Bitmap bitmap) {//Performs this operation on the given argument.
        saveImage(bitmap, filename);
    }
};
```

Annexe n°7 - Code source de la méthode de sauvegarde du Bitmap en image dans la galerie :

```java
/**
 * Save an image to a phone's gallery
 * @param bitmap - The bitmap that it has to be convert as an Image file to be saved into the gallery .
 * @param Filename - The string that contains the name of the file.
 */
protected void saveImage(Bitmap bitmap, String Filename) {
    if (android.os.Build.VERSION.SDK_INT >= 29) {
        ContentValues values = new ContentValues();
        values.put(MediaStore.MediaColumns.MIME_TYPE, "image/jpeg");
        values.put(MediaStore.Images.Media.DATE_ADDED, System.currentTimeMillis() / 1000);
        values.put(MediaStore.MediaColumns.DISPLAY_NAME, Filename);
        values.put(MediaStore.MediaColumns.RELATIVE_PATH, Environment.DIRECTORY_PICTURES);
        values.put(MediaStore.Images.Media.IS_PENDING, true);

        Uri uri = getContentResolver().insert(MediaStore.Images.Media.EXTERNAL_CONTENT_URI, values);
        if (uri != null) {
            try {
                saveImageToStream(bitmap, getContentResolver().openOutputStream(uri));
                values.put(MediaStore.Images.Media.IS_PENDING, false);
                this.getContentResolver().update(uri, values, null, null);
                Toast.makeText(this, "Screenshot saved", Toast.LENGTH_SHORT).show();
                OpenScreenshot(uri);
            } catch (FileNotFoundException e) {
                e.printStackTrace();
            }

        }
    }
}

/**
 * Save an image using a given outputstream and a given bitmap
 */
private void saveImageToStream(Bitmap bitmap, OutputStream outputStream) {
    if (outputStream != null) {
        try {
            bitmap.compress(Bitmap.CompressFormat.JPEG, 100, outputStream);
            outputStream.flush();
            outputStream.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Annexe n°8 - Méthode « listener » de la vue :

```java
// the purpose of the touch listener is just to store the touch X,Y coordinates
View.OnTouchListener touchListener = new View.OnTouchListener() {
    @Override
    public boolean onTouch(View v, MotionEvent event) {

        // save the X,Y coordinates
        if (event.getActionMasked() == MotionEvent.ACTION_DOWN) {
            lastTouchDownXY[0] = event.getX();
            lastTouchDownXY[1] = event.getY();
        }

        // let the touch event pass on to whoever needs it
        return false;
    }
};
```

Annexe n°9 - Extrait des modifications du fichier .pbtxt du module "face effect" :

```
#UPDATE
# Collection of detected/processed faces, each represented as a list of
# landmarks. (std::vector<NormalizedLandmarkList>)
output_stream: "multi_face_landmarks"
```

```
90      #UPDATE
91      # Subgraph that detects faces and corresponding landmarks.
92      node {
93        calculator: "FaceLandmarkFrontGpu"
94        input_stream: "IMAGE:throttled_input_video"
95        input_side_packet: "NUM_FACES:num_faces"
96        output_stream: "LANDMARKS:multi_face_landmarks"
97      }
98
99      #UPDATE
100     # Extracts a single set of face landmarks associated with the most prominent
101     # face detected from a collection.
102     node {
103       calculator: "SplitNormalizedLandmarkListVectorCalculator"
104       input_stream: "multi_face_landmarks"
105       output_stream: "face_landmarks"
106       node_options: {
107         [type.googleapis.com/mediapipe.SplitVectorCalculatorOptions] {
108           ranges: { begin: 0 end: 1 }
109           element_only: true
110         }
111       }
112     }
```

Annexe n°10 - Code source du callback pour obtenir les coordonnées des points de repères du visages nécessaires :

```
//This callback demonstrates how the output normalized landmark packet can be obtained and used in an Android app.
//initialize coordinates of used landmarks to create kind of bounding box
processor.addPacketCallback(
       OUTPUT_LANDMARKS_STREAM_NAME,
       (packet) -> {
           Log.v(TAG, "Received multi face landmarks packet.");
           List<NormalizedLandmarkList> multiFaceLandmarks =
                   PacketGetter.getProtoVector(packet, NormalizedLandmarkList.parser());
           //initialize coordinates of the limit of the "bounding box"
           boxlimitTop = multiFaceLandmarks.get(0).getLandmarkList().get(10).getY()*1920f; //multiply by 1920 allow to obtain an usable value as cooridnate to mark the top limit of the bounding box
           boxlimitLeft = multiFaceLandmarks.get(0).getLandmarkList().get(234).getX()*1080f;//multiply by 1080 allow to obtain an usable value as cooridnate to mark the left limit of the bounding box
           boxlimitRight = multiFaceLandmarks.get(0).getLandmarkList().get(454).getX()*viewGroup.getRight(); // *1080f make the right limit shorter
           boxlimitBottom = multiFaceLandmarks.get(0).getLandmarkList().get(152).getY()*viewGroup.getBottom();// *9120f make the bottom limit shorter
       });
```

Annexe n°11 - Code source de l'interaction avec l'interface utilisateur pour gérer la barre déroulante des filtres :

```
private void HideMaterialComponents() {
    if(lastTouchDownXY[1]>boxlimitTop && lastTouchDownXY[0]>boxlimitLeft && lastTouchDownXY[1]<boxlimitBottom && lastTouchDownXY[0]<boxlimitRight){
        ToggleVisibility(horizontalScrollView);
    }else{
        Toast.makeText(this, "Press on your face", Toast.LENGTH_SHORT).show();
    }
}
```

Annexe n°12 - Code source de la méthode du minuteur :

```
/**
 * Create a timer that hide the scrollbar view until a some amount of seconds
 */
public void hideScrollbar(){
    //timer to make disappear the scrollbar once we hit a effect button
    if (!taskScheduled) {
        taskScheduled = true; //mark the task scheduled
        timer.schedule(new TimerTask() { //Schedules the specified task for execution after the specified delay
            @Override
            public void run() {
                runOnUiThread(new Runnable() {//Allow to modify UI as running on the UI thread (called main thread)
                    public void run() {
                        horizontalScrollView.setVisibility(View.GONE);
                    }
                });
                taskScheduled = false;
            }
        }, 5000);
    } else if (taskScheduled) {//if we hit the button while a task is already scheduled
        timer.cancel(); //Terminates this timer, discarding any currently scheduled tasks to avoid that 2 timer.Schedule() works alongside
        timer = new Timer(); //create a new one and then work as expected
        timer.schedule(new TimerTask() { //Schedules the specified task for execution after the specified delay
            @Override
            public void run() {
                runOnUiThread(new Runnable() {//Allow to modify UI as running on the UI thread (called main thread)
                    public void run() {
                        horizontalScrollView.setVisibility(View.GONE);
                    }
                });
                taskScheduled = false;
            }
        }, 5000);
    }
}
```

Résumé

Lors de mon stage, j'ai travaillé sur une application Android, utilisant une technologie de réalité augmentée, pour appliquer des filtres sur le visage d'un utilisateur.

Mon objectif principal, était de rendre l'application fonctionnelle, pour à terme, faciliter les recherches pour permettre de réussir à atteindre le véritable objectif de cette application : trouver comment appliquer différents filtres sur plusieurs visages, en simultanée, et indépendamment des uns et des autres.

J'ai réussi à répondre à quasiment toutes les demandes que j'ai eu concernant l'application. Cependant, je n'ai pu trouver de solution pour atteindre le véritable objectif de l'application, car les outils utilisés ne le permettent pas actuellement, et que je n'avais aucune idée pour arriver à produire un tel résultat.

TRAITEMENT DOCUMENTAIRE

NOM ETUDIANT :

Signature

Date de Naissance :

NOM TUTEUR IUT :

DEPT. – ANNEE – PROMO :

NOM ENTREPRISE :

Ville- Pays :

NOM TUTEUR ENTREPRISE :

Signature

☐ **Confidentiel ***

SUJET DE STAGE :

MOTS CLES (5) :

RESUME :

NB :Votre signature valide et donne droit à la bibliothèque de l'IUT de Toulouse II Blagnac, de diffuser ce document dans le catalogue commun du réseau des bibliothèques des universités de Toulouse.

Portail du réseau : http://bibliotheques.univ-toulouse.fr/38310688/0/fiche___pagelibre/&RH=

*** Mettre une croix si confidentiel**