

File System Analyzer and Explorator (FSAE)



Auteur :	Müller Pierrick et Spinelli Isaia
Prof :	Bruegger Pascal
Assit :	Zurbuchen Nicolas
Date :	03.01.2021
Salle :	A4 – Lausanne
Classe :	T-MobOp

Table des matières

Introduction.....	- 2 -
But	- 2 -
Motivations	- 2 -
Analyse	- 2 -
Conception	- 3 -
Graphique.....	- 3 -
Mode explorateur.....	- 3 -
Mode analyseur	- 3 -
Classes	- 4 -
Implémentation.....	- 5 -
Technologies utilisées	- 6 -
Fonctionnalité supplémentaire	- 7 -
Analyse	- 7 -
Conception	- 8 -
Implémentation.....	- 8 -
Profiling	- 8 -
Problèmes rencontrées	- 9 -
Utilisation	- 10 -
Conclusion	- 10 -
Qui a fait quoi	- 10 -
Améliorations	- 10 -
Compétences acquises	- 10 -
Résultats obtenus	- 10 -
Sources	- 11 -
Annexes	- 11 -

Introduction

But

Dans le cadre du cours T-MobOp (Systèmes d'exploitation mobiles et applications) nous avons dû proposer et réaliser un mini-projet personnel afin de se familiariser avec l'environnement de développement et les concepts d'Android.

Motivations

Pour ce mini-projet, nous avons décidé de réaliser un explorateur de fichier mais par-dessus tout, un analyseur du système de fichier. Brièvement, l'explorateur de fichier permet de se déplacer dans toute l'arborescence du système et l'analyseur permet de fournir une bonne visualisations de la place mémoire prise par les différents répertoires.

Ce choix a été pris pour des raisons d'envie personnel. Certes, il existe déjà plusieurs explorateurs de fichier, cependant, c'est la partie analyse qui nous intéresse le plus. En effet, l'objectif principal est de permettre à tout le monde de prendre conscience, de manière agréable, évidente et compréhensible, la répartition et l'utilisation de la place mémoire.

Analyse

Pour la partie explorateurs de fichier, voici les fonctionnalités qui devront être implémentées :

1. Déplacement dans la hiérarchie inférieure du système de fichier de répertoire en répertoire.
2. Enregistrement et affichage du chemin actuel depuis la racine.
3. Déplacement instantané à un répertoire dans la hiérarchie supérieure.
4. Distinction claire entre un répertoire et un fichier.
5. Affichage de la place mémoire de chaque item (répertoire ou fichier).
6. Affichage du nombre de fichier pour chaque répertoire.
7. Créer un nouveau dossier
8. Créer un nouveau fichier
9. Supprimer un dossier
10. Supprimer un fichier
11. Trier la liste des items (répertoire/fichier)

Il sera en tout temps possible de passer en mode « analyseur » afin de visuellement de manière ergonomique la place mémoire de chaque item ainsi que la répartition. De ce fait, plusieurs points devront être mis en place :

1. Possibilité de passer en mode analyseur facilement.
2. Visualisation aisée de la répartition de la place mémoire.
3. Affichage en pourcent ou MB de la place mémoire des items.
4. Possibilité de passer en mode explorateur facilement.
5. Possibilité de se déplacer dans un répertoire.

Voici toutes les fonctionnalités désirées dans notre application.

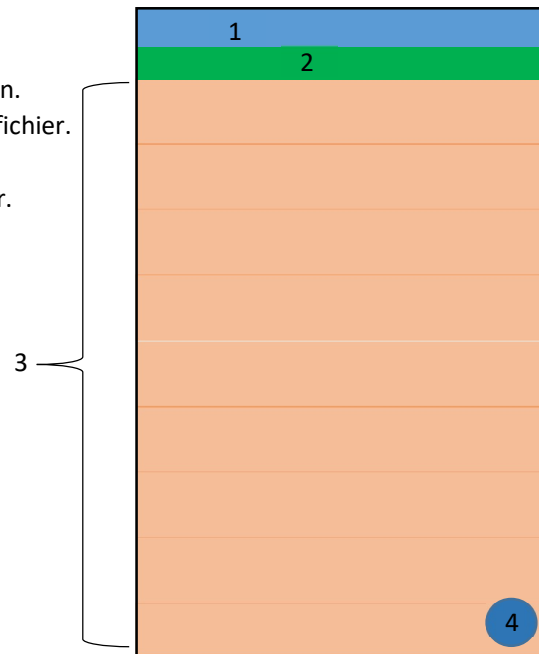
Conception

Graphique

Dans ce chapitre, nous allons voir comment nous avons pensé approximativement l'agencement des différents interfaces.

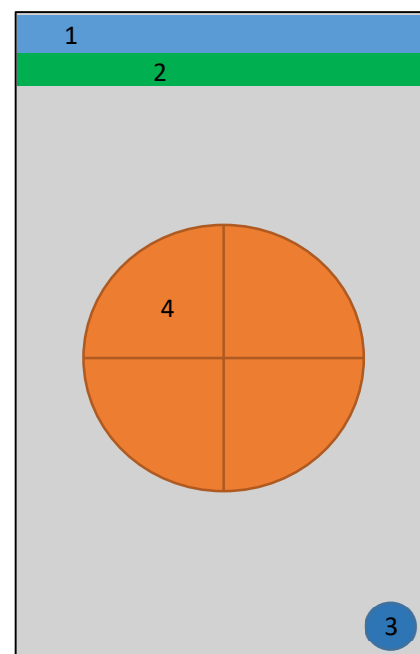
Mode explorateur

1. Bar de menu avec les fonctionnalités à disposition.
2. Chemin actuel dans la hiérarchie du système de fichier.
3. La liste des items.
4. Bouton permettant de passer en mode analyseur.



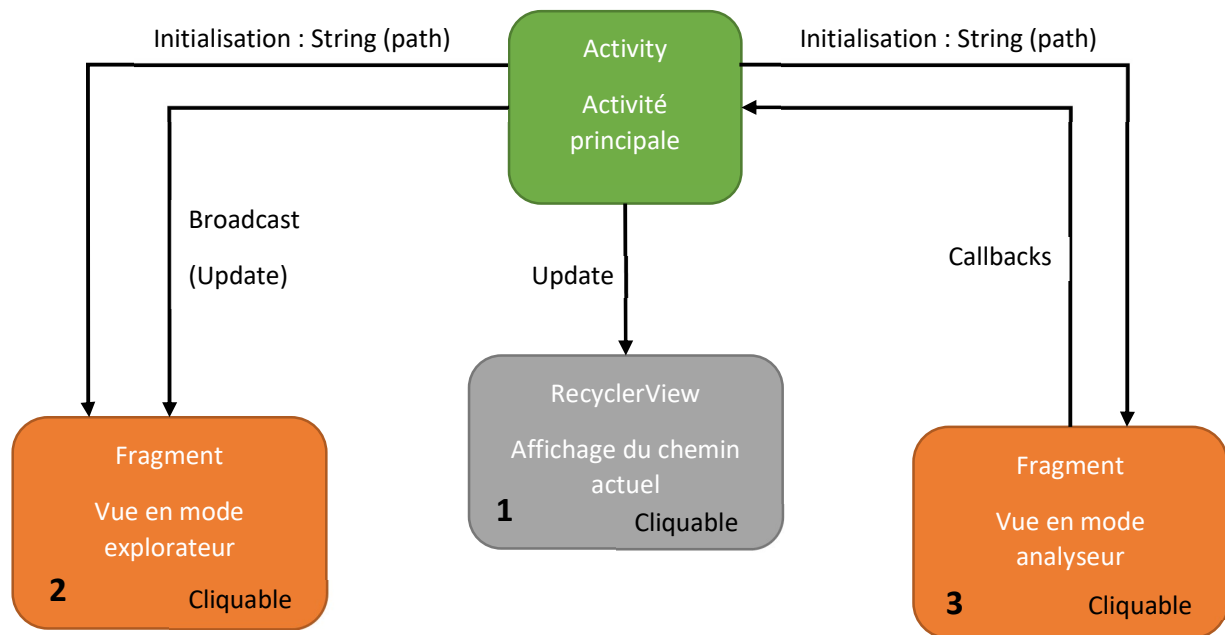
Mode analyseur

1. Bar de menu avec les fonctionnalités à disposition.
2. Chemin actuel dans la hiérarchie du système de fichier.
3. Bouton permettant de passer en mode explorateur.
4. Diagramme circulaire permettant la visualisation de la répartition de la place mémoire.



Classes

Dans ce chapitre, nous allons voir comment nous avons pensé approximativement l'organisation des différents objets/classes.



Comme on peut le voir ci-dessus, l'activité principale gèrera 3 principaux objets différents :

1. RecyclerView : Permet de mémoriser et d'afficher en tout temps le chemin actuel dans la hiérarchie du système de fichier de l'utilisateur.
2. Fragment : Permet d'afficher et mémoriser tous les items du répertoire courant (mode explorateur).
3. Fragment : Permet d'afficher la répartition de tous les items du répertoire courant (mode analyseur).

Ces trois entités seront bien évidemment cliquables pour permettre ou faciliter la navigation dans l'arborescence du système de fichier.

De plus, il sera possible d'interagir avec ces entités :

1. RecyclerView : Pour chaque déplacement dans l'arborescence, le chemin courant devra être mis à jour.
2. Fragment : Il sera nécessaire lors de l'instanciation d'un fragment du mode explorateur, de lui fournir le chemin courant. De plus, il sera possible de mettre à jour ce fragment dans le cas où un dossier/fichier serait supprimé ou ajouté.
3. Fragment : Il sera nécessaire lors de l'instanciation d'un fragment du mode analyseur, de lui fournir le chemin courant. De plus, dans le cas où un déplacement serait effectué, il sera nécessaire d'en informer l'activité principale.

Implémentation

Dans ce chapitre nous n'allons pas décrire le code implémenté car cela risquerait d'être trop fastidieux. Néanmoins, nous allons lister les différentes étapes d'implémentation réalisées après la conception ainsi que quelques points ne concernant pas directement l'implémentation de code. Incontestablement, tous le code du projet est disponible en annexe est commenté.

Pour commencer, voici la liste des grandes étapes réalisées :

- Recherche pour le mode explorateur de fichier.
- Recherche pour le mode analyseur de fichier.
- Création et implémentation d'un projet pour le mode explorateur de fichier.
- Création et implémentation d'un projet pour le mode graphique (analyse).
- Fusion des deux projets. (Partie la plus complexe)
- Amélioration visuelles de l'application.
- Ajout de différentes fonctionnalités.
- Refactor du projet.
- Recherche et correction de bugs.
- Ajout d'une fonctionnalité non prévue
- Refactor du projet.
- Recherche et correction de bugs.

Bien entendu, la partie recherche et débogage est une tâche constante lors de la réalisation d'un projet.

Voici quelques modifications apportées au fichier « AndroidManifest.xml » :

1. La point le plus important, les permissions de l'application :

```
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />  
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

2. La modification du nom de l'application ainsi que son icône personnalisée :

A screenshot of an AndroidManifest.xml file. On the left, there are two small icons: a green square with a white dot and a green circle with a white dot. To the right, the XML code is displayed: `android:icon="@mipmap/ic_launcher"`, `android:label="FSAE"`, and `android:roundIcon="@mipmap/ic_launcher_round"`.

```
android:icon="@mipmap/ic_launcher"  
android:label="FSAE"  
android:roundIcon="@mipmap/ic_launcher_round"
```

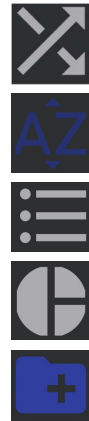
Remarque : l'icône de l'application est visible sur la page de garde de ce rapport.

Dans le fichier « build.gradle » il a été nécessaire d'ajouter la dépendance pour la partie graphique du projet :

```
// https://github.com/PhilJay/MPAndroidChart  
implementation 'com.github.PhilJay:MPAndroidChart:v3.1.0'
```

Finalement, plusieurs icônes ont été créés dans le dossier « res/drawable » afin d'améliorer l'ergonomie de l'application. En voici une liste non-exhaustive :

- Icône du bouton pour choisir le sens du tri des items.
- Icône du bouton pour choisir un tri par ordre alphabétique.
- Icône du bouton pour passer en mode exploreur.
- Icône du bouton pour passer en mode analyseur.
- Icône du bouton pour créer un nouveau dossier.



Technologies utilisées

Voici ci-dessous, la liste des différentes technologies utilisées lors de ce mini-projet :

1. Activités
2. Fournisseur de contenu
3. Intentions
4. Récepteurs de diffusion
5. Fragment
6. Vue recyclée
7. Action bar
8. Bouton flottant
9. PieChart
10. Menu & groupe de Menu
11. Dialogue
12. Toast
13. Notification
14. Thread
15. Filtre de fichier
16. Préférences partagées (not yet)

Nous pouvons constater qu'une grande variété de technologies ont été utilisées pour la réalisation de notre application.

Fonctionnalité supplémentaire

Étant donné que nous arrivions gentiment à un état stable de l'application souhaitée, nous avons décidé d'ajouter une fonctionnalité. Encore une fois, cette fonctionnalité est une motivation personnel car nous la désirions depuis un moment déjà.

Cette fonctionnalité se nommera « Seek & Class all image files ». Comme l'indique son nom, l'objectif est de rechercher toutes les images de l'appareil et de les classer par date.

Analyse

Malheureusement la date de la prise d'une photo n'est pas toujours disponible. Cependant, il est fréquent que la nom de l'image contienne la date effective de la photo. Dans le cas où la date n'est pas disponible dans le nom de l'image, la date de la dernière modification du fichier sera pris en compte.

L'idée finale est d'avoir un dossier contenant toutes les images de l'appareil. Toutes ces images seront placées dans une hiérarchie de trois niveaux :

1. Répertoire de l'année
2. Répertoire du mois
3. Répertoire du jour

Pas exemple, une image prise le 23 janvier 2018 sera dans le dossier « 2018/01_Jan/23/ ».

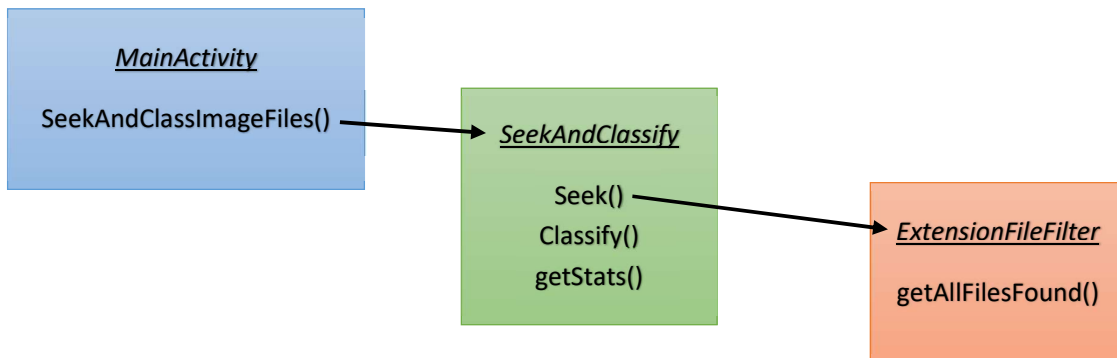
Pour commencer, nous avons cherché toutes les possibilités de nom d'image en notre possession contenant la date. Voici les différents exemples de noms trouvés :

1. 20171016_183246.jpg
2. 20190203011737_picture.jpg
3. IMG_20190110_210549.jpg
4. IMG-20200815-WA0010.jpg
5. Screenshot_2015-05-25-05-08-26.png
6. Screenshot_20190324-111621.jpg
7. Screenshot_20190808_232048_com.android.chrome.jpg
8. photo_2019-04-04_10-17-16.jpg

Nous devons donc mettre en place un système permettant de détecter et parser tous ces différents type de nom. Comme mentionné précédemment, si aucun de ces noms n'est détectés, la date de la dernière modification du fichier sera prise comme date.

Conception

Ci-dessous, on peut voir un petit schéma résumant la conception de cette fonctionnalité :



Lorsque l'utilisateur fera appel à cette fonctionnalité, l'activité principale appellera la fonction « SeekAndClassImageFiles() ». Celle-ciinstanciera et utilisera la classe « SeekAndClassify » afin de :

- Seek() : Rechercher toutes les images depuis un répertoire.
- Classify() : Classer toutes ces images.
- getStats() : Avoir des statistiques, ceci peut être très utile pour le développement de cette fonctionnalité.

La fonction « Seek() » utilisera la classe « ExtensionFileFilter » afin de rechercher tous les fichiers de type image.

La fonction « Classify() » devra essayer de détecter et parser les noms des images afin de les classer dans les bons répertoires.

Implémentation

À la base nous souhaitions créer uniquement des liens symboliques afin d'éviter de consommer de la place mémoire, malheureusement, nous n'avons pas réussi car nous n'avons pas les droits nécessaires dans l'arborescence de système de fichier.

Tous le code nécessaire est en annexe. Plus précisément, les fichiers « SeekAndClassify.kt », « ImgFileFilter.kt » et « MainActivity.kt » sont concernés par cette fonctionnalité.

Profiling

Au départ nous testions cette fonctionnalité avec une dizaine images, donc le temps de traitement était relativement rapide. En testant avec un plus grand nombre d'image, nous avons remarqué que le temps nécessaire était non négligeable. De ce fait, nous avons commencé par essayer de comprendre plus exactement ce qui prenait du temps grâce à l'outil de profiling intégré à Andoid Studio.

Name	Total (μs)	%	Self (μs)	%	Children (μs)	%
onOptionsItemSelected() (com.example.fileexplorer.MainActivity)	8'096'699	100.00	0	0.00	8'096'699	100.00
▶ classify() (com.example.fileexplorer.seekAndClassify)	7'315'242	90.35	0	0.00	7'315'242	90.35
▶ seek() (com.example.fileexplorer.seekAndClassify)	652'662	8.06	0	0.00	652'662	8.06
▶ updateContentOfCurrentFragment() (com.example.fileexplorer.MainActivity)	128'795	1.59	0	0.00	128'795	1.59

On peut voir dans l'image ci-dessus, que la fonction de classification (classify) consomme plus de 90% du temps. La fonction de recherche (seek) et la fonction de mise à jour du fragment (updateContentOfCurrentFragment) prend relativement peu de temps.

Name	Total (µs)	%	Self (µs)	%	Children (µs)	%
onOptionsItemSelected() (com.example.fileexplorer.MainActivity)	8'096'699	100.00	0	0.00	8'096'699	100.00
classify() (com.example.fileexplorer.seekAndClassify)	7'315'242	90.35	0	0.00	7'315'242	90.35
classifyFile() (com.example.fileexplorer.seekAndClassify)	7'279'469	89.91	2'149	0.03	7'277'320	89.88
copyTo\$default() (kotlin.io.FilesKt__UtilsKt)	7'061'085	87.21	0	0.00	7'061'085	87.21
DateToPath() (com.example.fileexplorer.seekAndClassify)	202'724	2.50	7'451	0.09	195'273	2.41
<init>() (java.io.File)	7'720	0.10	0	0.00	7'720	0.10
<init>() (java.lang.StringBuilder)	5'791	0.07	3'659	0.05	2'132	0.03
getLastModified() (com.example.fileexplorer.seekAndClassify)	6'894	0.09	0	0.00	6'894	0.09
getAbsolutePath() (java.io.File)	6'830	0.08	0	0.00	6'830	0.08
sortWith() (kotlin.collections.CollectionsKt__MutableCollectionsJVMKt)	4'685	0.06	0	0.00	4'685	0.06
getDateFromEWA() (com.example.fileexplorer.seekAndClassify)	3'543	0.04	0	0.00	3'543	0.04
getDateFromEWDash() (com.example.fileexplorer.seekAndClassify)	3'052	0.04	0	0.00	3'052	0.04
distinct() (kotlin.collections.CollectionsKt__CollectionsKt)	2'753	0.03	0	0.00	2'753	0.03
getName() (java.io.File)	2'185	0.03	0	0.00	2'185	0.03
getDateFromEWDash() (com.example.fileexplorer.seekAndClassify)	1'983	0.02	0	0.00	1'983	0.02
getStartDate() (com.example.fileexplorer.seekAndClassify)	1'973	0.02	0	0.00	1'973	0.02
<init>() (java.lang.StringBuilder)	1'875	0.02	0	0.00	1'875	0.02
seek() (com.example.fileexplorer.seekAndClassify)	652'662	8.06	0	0.00	652'662	8.06

Plus précisément, on peut voir que c'est la fonction de copie (copyTo) qui consomme la grande majorité du temps. Il est difficilement possible d'optimiser cette fonction car elle provient d'une librairie.

Afin de ne pas bloquer l'interface utilisateur au risque de gêner l'utilisateur, nous avons décidé d'exécuter la fonction de classification (classify) dans un Thread. De plus, nous avons ajouté une notification avec une barre de progression afin d'informer l'utilisateur de la progression de la tâche en cours.

Finalement, nous avons aussi ajouté un dialogue d'alerte comme confirmation lors du lancement de la fonctionnalité étant donné que celle-ci peut prendre passablement de temps et de place mémoire.

Problèmes rencontrés

Comme tous développeurs, nous avons rencontrés des problèmes tout au long du projet. Voici entre autres quelques problèmes que nous avons rencontrés :

- Gestion de toutes les possibilités visuelle du diagramme circulaire
- Fusion des deux premiers projets de test (Explorateur - Analyseur)
- La ligne « android:requestLegacyExternalStorage="true" » dans le manifeste
- Premier lancement de l'application

Le premier lancement de l'application est un problème encore d'actualité. En effet, nous n'avons pas réussi à faire que l'affichage de l'application au premier lancement (après l'acceptation des permissions) se fasse correctement. Pour contourner ce problème, nous arrêtons l'application peu importe la réponse de l'utilisateur (permissions acceptées ou refusées). Dans le cas où l'utilisateur accepte les permissions, nous affichons un message confirmant l'acceptation de ces dernières et indiquant qu'il faut relancer l'application. Ceci sera démontré dans la vidéo démonstrative.

Utilisation

Une vidéo démonstrative est disponible en annexe dans le dossier fourni

Conclusion

Qui a fait quoi

Müller Pierrick	Spinelli Isaia
Recherche pour le mode explorateur de fichier	Recherche pour le mode analyseur de fichier
Création et implémentation d'un projet pour le mode explorateur de fichier	Création et implémentation d'un projet pour le mode graphique
Fusion des deux projets	Fusion des deux projets
Amélioration visuelles de l'application	Amélioration visuelles de l'application
Ajout de différentes fonctionnalités	Ajout de différentes fonctionnalités
Refactor du projet	Refactor du projet
Recherche et correction de bugs	Recherche et correction de bugs
...	Ajout de la fonctionnalité supplémentaire
...	...
...	...

Améliorations

Comme dans tous les projets, le refactor est une tâche qui peut presque toujours être appliquée. Pour le moment, notre application se place dans la mémoire interne du téléphone, mais serait intéressant de pouvoir se placer sur la carte SD s'il y en a une détectée. De plus, nous n'avons pas parlé de ça ce semestre mais nous pensons que mettre en place des tests unitaires et d'intégrations. Finalement, nous avons mis en place une fonctionnalité permettant de classer toutes les images de l'appareil, mais il serait pratique de pouvoir naviguer facilement au travers des images et des dates.

Compétences acquises

Le langage de programmation Kotlin.

Beaucoup d'Android Studio :

1. Toutes les technologies utilisées
2. L'outil de profiling d'Android Studio
3. La création est gestion de device virtualisé

Par-dessus tout ça, nous avons surtout appris la gestion complète d'un projet mobile, dont la conception, l'implémentation, la recherche et le débogages.

Résultats obtenus

Certes nos objectifs de base n'étaient pas à la hauteur d'un projet grandiose cependant nous sommes parvenus à réaliser complètement toutes les fonctionnalités souhaitées et plus encore. De plus, nous avons eu le plaisir de travailler sur des plus petits détails tels que la création d'icônes et la recherche de bugs afin d'avoir un projet final stable et ergonomique. Pour conclure, nous sommes fières de ce mini-projet et d'avoir pu accomplir quelques choses de réellement utile.

Sources

Recherche de fichiers par extensions : <https://stackoverflow.com/questions/11015833/getting-list-of-all-files-of-a-specific-type>

Librairie graphique pour Android : <https://github.com/PhilJay/MPAndroidChart>

Tutoriel explorateur de fichier : <http://thetechnocafe.com/build-a-file-explorer-in-kotlin-part-1-introduction-and-set-up/>

Annexes

1. Vidéo démonstrative
2. Projet complet Android Studio (.zip)
3. Application (.apk)

Date : 03.01.21

Nom de l'étudiant : Müller Pierrick et Spinelli Isaia