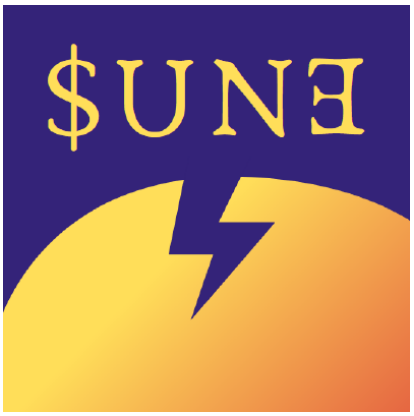
APPMOB

Compte rendu de projet

IUT de Paris
Informatique S4 mars 2021

Anicet Nougaret 202 | Thibault Henrion 202 | Daniel Aguiar 202
Bilal Mahdjoubi 202 | Jules Vanin 204



SUNE

Un utilitaire mobile et autonome pour les réparateurs de téléphones android et les curieux

Sommaire

Sommaire **2**

Présentation **3**

Cas d’utilisation 3

Technique **4**

Lexique 4

Fonctionnalités Android utilisées 4

Diagramme d’architecture 5

Visuels et expérience utilisateur **6**

Bilan **7**

Difficultés surmontées 7

Pistes d’amélioration 8

Conclusion 8

Présentation

L’application Android SUNE est un utilitaire pour les réparateurs de téléphone.

Il permet aux réparateurs d’effectuer la commande shell android “getprop”, indispensable pour récupérer des informations de débuggage. et de visualiser son résultat dans une interface bien plus facile à naviguer que le terminal.

De plus, les réparateurs peuvent s’enregistrer pour sauvegarder les résultats obtenus sur n’importe quel téléphone dans le cloud et les consulter depuis leurs autres appareils.

Cas d’utilisation

On peut imaginer le cas d’utilisation suivant : Un utilisateur ayant un problème logiciel sous Android se rend chez un réparateur pour faire diagnostiquer son appareil. Le réparateur lui installe l’application SUNE dans la minute, se connecte à son compte et lance la commande de débogage sans même avoir à sortir des utilitaires de débogage avancés, ni rooter le téléphone du client, ni utiliser une connexion câblée avec un ordinateur.

Le résultat de la commande de débogage est immédiatement enregistré dans le cloud ce qui permet au réparateur de très rapidement retirer l’utilitaire et rendre le téléphone au client. Après le départ du client, le réparateur se connecte à son compte sur son téléphone de travail et peut consulter les résultats obtenus sur l’appareil du client pour diagnostiquer.

L’application pourrait facilement être étendue avec plus de commandes de débogages populaires par la suite pour offrir plus de capacités de diagnostic.

Technique

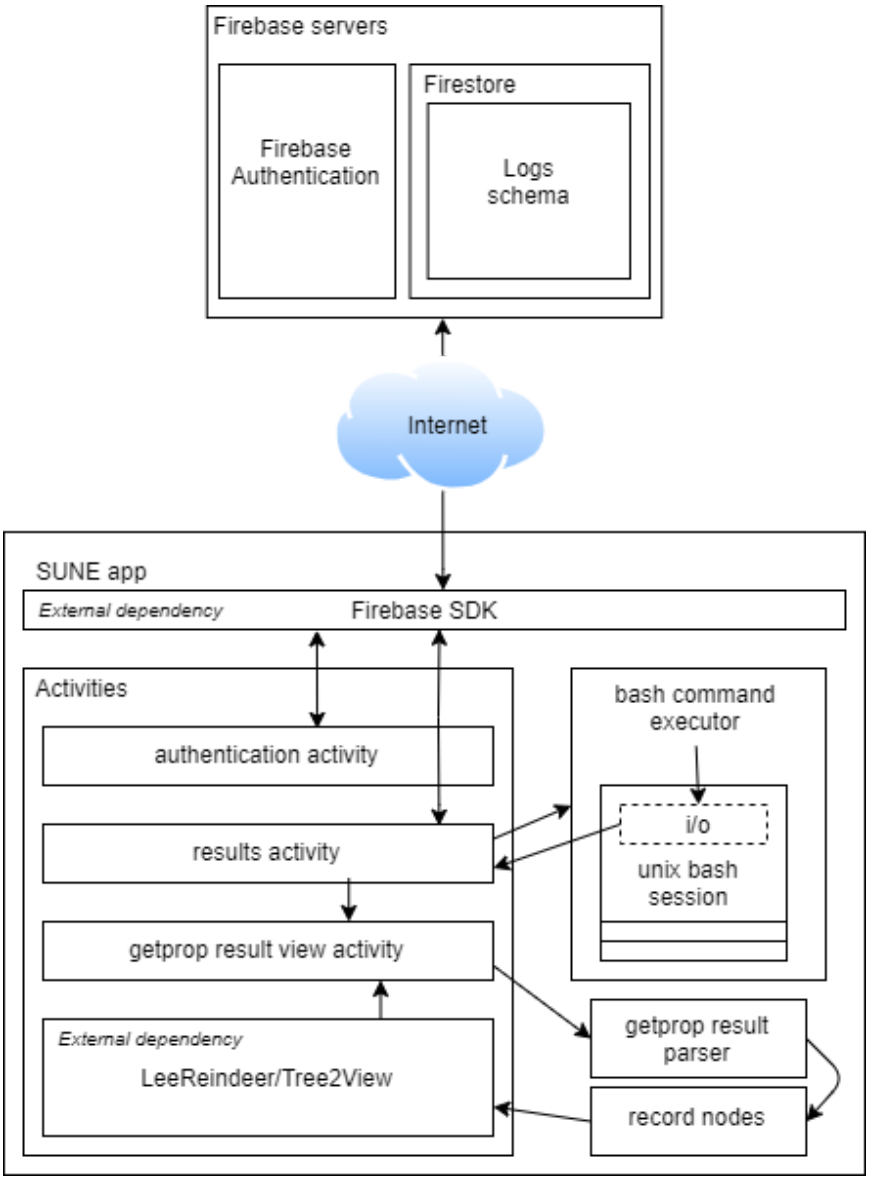
Lexique

- “Getprop” : Commande shell Android affichant une centaine de propriétés systèmes classées par domaines de noms. Exemple : “[os.version.update.date]: [18/02/2021 4pm GMT]”
- “Result” : Un résultat de la commande “getprop” consistant en une liste de propriétés sous forme de texte ou sous forme d’arbre.
- “Parser” : Algorithme servant à passer des résultats de “getprop” sous forme de texte à une structure en arbre (classe ResultNode).
- “Tree2View” : Librairie Android open-source permettant de naviguer des données organisées sous forme d’arbre.
- “Firebase” : Service cloud de Google permettant l’enregistrement et l’identification des utilisateurs, ainsi que la persistance de données de ces utilisateurs dans le cloud.

Fonctionnalités Android utilisées

- Changements d’activité
- Utilisation de librairies externes (Firebase, Tree2View)
- Persistance des données dans le cloud avec Firebase
- Exécution de commandes shell et lecture de l’IO depuis le code Java.
- Tests unitaires (classe ResultNodeTest)
- Stylisation de l’interface (changement des couleurs et de la police)

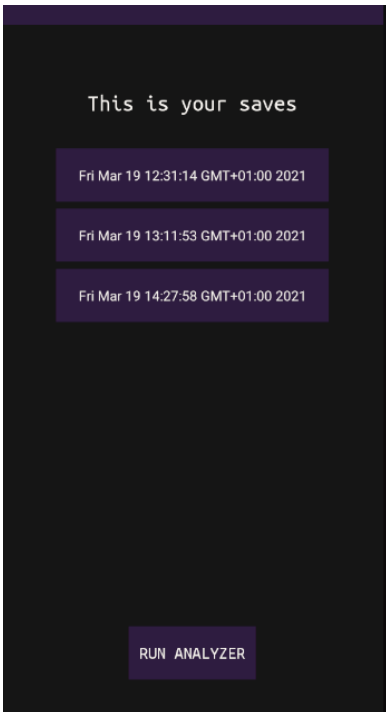
Diagramme d'architecture

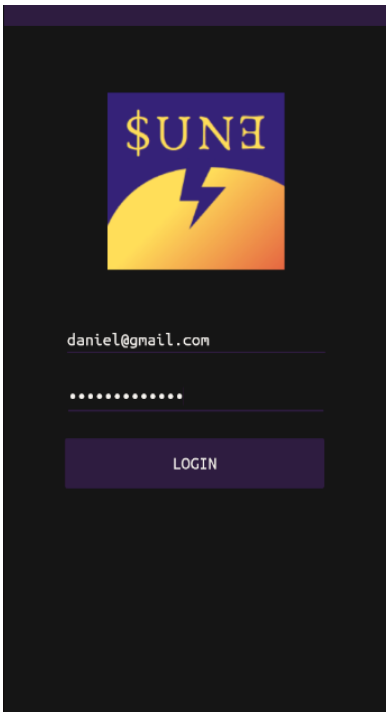


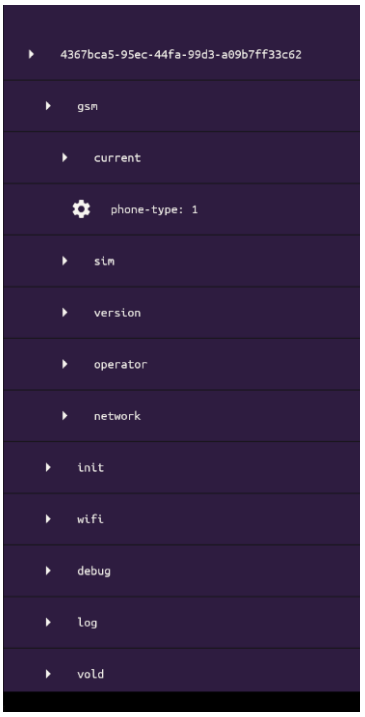
Visuels et expérience utilisateur

Avec cette application nous souhaitions proposer une interface simple et sans distractions.

Nous avons tiré parti de librairies externes pour nous épauler, telle que Tree2View, une librairie permettant de naviguer des structures arborescentes avec aisance. Au vue de la hiérarchisation des données apparente lors de l'utilisation de la commande getprop, nous avons trouvé judicieux l'utilisation de cette librairie.







Bilan

Difficultés surmontées

Le cahier des charges du projet a rencontré de nombreux changements à la suite de recherches et d'essais infructueux, mais enrichissants, pour notre concept initial.

En effet, il s'agissait de faire une application capable de mesurer la consommation électrique de chaque application.

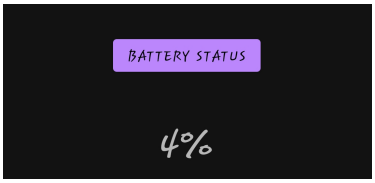
Seulement, en raison des limitations techniques et des maigres permissions des applications Android tournant sur des appareils non rootés, il s'est avéré après plusieurs semaines d'essais et de recherches impossible d'implémenter ces fonctionnalités sans nuire totalement à l'accessibilité de l'application.

Par exemple, nous avons été amenés à réaliser de nombreux prototypes utilisant des commandes unix telles que "time" ou "ps", et des commandes adb shell comme "dumpsys cpuinfo". Nous avons réussi à exécuter ces commandes depuis Java, à récupérer leur résultat. Mais malheureusement, aucune des commandes trouvées ne permettaient d'obtenir à la fois des informations permettant de déduire la consommation électrique et des informations sur les différentes applications concernées.

```
u -c dumpsys activity | grep -e "Proc #" -e "PERS #" returns 1 line per process:

 PERS #47: sys   F/ /PER  trm: 0 1897:system/1000 (fixed)
 PERS #46: pers  F/ /PER  trm: 0 2070:com.android.systemui/u0a29 (fixed)
 ...
 Proc # 2: fore  T/A/TOP  trm: 0 8160:com.whatsapp/u0a101 (top-activity)
 Proc #36: vis   F/ /BFGS trm: 0 2178:com.breel.wallpapers18/u0a51 (service)
 Proc #30: vis   F/ /BFGS trm: 0 2426:com.google.android.ext.services/u0a35 (service)
 Proc # 7: vis   F/ /BFGS trm: 0 14594:com.google.android.googlequicksearchbox:interactor/u0a39 (service)
 Proc # 6: vis   F/ /BFGS trm: 0 2897:com.google.android.apps.nexuslauncher/u0a31 (service)
 Proc #10: cch   B/ /CEM  trm: 0 11853:com.android.providers.calendar/u0a41 (cch-empty)
```

Nous avons tout de même mis au point un prototype de mesure de la consommation de batterie ne pouvant fonctionner qu'en debug à cause des permissions Android :



C’était un processus de recherche très intéressant, mais afin d’obtenir une application utilisable en production, nous avons décidé de rebondir et de changer de concept.

Nous avons donc finalement décidé de faire un utilitaire se servant de la commande “getprop” offrant une structure en arborescence intéressante à traiter et à visualiser, ainsi que des informations riches de débogage. Ce changement de cap nous a permis tout de même de nous servir des techniques apprises lors de la phase de recherche, comme l’exécution d’une commande avec Bash dans Java et la récupération du flux de sortie.

Pistes d’amélioration

Il serait intéressant pour une meilleure expérience utilisateur de proposer une fonctionnalité de recherche dans l’activité de visualisation des propriétés extraites par “getprop”.

Nous pourrions aussi facilement ajouter des fonctionnalités utilisant d’autres commandes de débogage, comme “ps” ou “time” à l’aide du code déjà présent dans l’application.

Conclusion

Nous sommes satisfaits de notre application. Nous pensons que malgré son minimalisme, les différentes phases de prototypage et de recherche pour l’utilisation de fonctionnalités bas-niveau d’Android nous ont beaucoup appris. De plus, nous avons fini par utiliser tous les aspects de la programmation Android souhaitée.