Rapport de projet Java

See&Share

Projet réalisé en 2011-2012 par

Guillaume GÉRARD, Julien GUÉPIN, Adrien SIEBERT et Julien VAUBOURG

ESIAL - Promotion 2013

E S I A L

UNIVERSITÉ DE LORRAINE

Sommaire

I. Introduction

Le projet Java consistait à développer un logiciel permettant la gestion de photos et d'albums d'un ordinateur, ainsi que la connexion au service Picasa de Google. Nous avons utilisé l'API Java de Google pour nous connecter au serveur et gérer facilement la synchronisation des photos et des albums. Nous détaillons dans ce dossier les fonctionnalités développées, l'architecture du logiciel, ainsi qu'une documentation utilisateur.

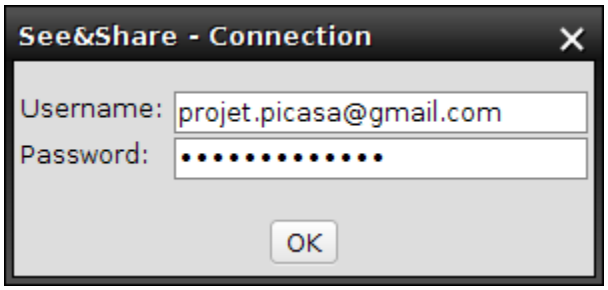
Le code du logiciel, les diagrammes de conception, les tests unitaires, ainsi que la documentation sont disponibles sur le dépôt Subversion à l'adresse :

http://redmine.esial.uhp-nancy.fr/projects/projet-java2011-groupe20

II. Fonctionnalités

1. Affichage du compte Picasa

Au lancement du logiciel, l'utilisateur peut saisir les coordonnées de son compte Google pour se connecter à Picasa. Si le compte est correct, la liste des albums en ligne est alors affichée. L'utilisateur peut cliquer sur un album et parcourir les photos qu'il contient.



Il pourra à tout instant changer de compte Picasa, ce qui aura pour effet de laisser les dossiers locaux pour ne remplacer que les albums en ligne et synchronisés.

2. Ajout de dossiers locaux et d'albums

L'utilisateur peut sélectionner un dossier sur le disque qu'il veut importer dans See&Share. Toutes les photos qu'il contient seront alors affichées dans le logiciel, et il pourra y apporter des traitements.

Tous les ajouts et suppressions sont **récursifs** (plusieurs albums seront créés si le dossier en contient d'autres contenant des médias).

Il est également possible de créer un **album *logique***, c'est-à-dire qui ne correspond à aucun dossier physique sur le disque, mais qui peut contenir des photos provenant de plusieurs albums (locaux ou en ligne).

3. Synchronisation entre le disque et Picasa

L'utilisateur peut synchroniser un album entre le disque et le service Picasa. Si l'album est **stocké en ligne**, il sera alors téléchargé et stocké sur le disque, dans le répertoire du logiciel (~/See-N-Share). Si l'album est **stocké sur le disque**, il sera alors envoyé vers Picasa dans un nouvel album.

Toutes les modifications effectuées depuis See&Share seront alors répercutées sur les fichiers locaux ainsi que sur les albums distants.

4. Gestion des albums

La liste des albums est affichée à gauche du logiciel, à côté d'**une icône indiquant le type d'album** mise à jour en temps réel. Un clic sur le nom de l'album permet d'afficher son contenu : une représentation sous forme de vignettes des médias qu'il contient. **Des informations sur l'album** sont affichées dans la partie haute de l'écran principal : nom de l'album, date de la dernière modification, type de l'album (local, en ligne ou synchronisé), visibilité de l'album en ligne (public, privé ou protégé).

Il est possible de cliquer sur le bouton *Sync* pour **synchroniser l'album avec Picasa** (si celui-ci ne l'est pas déjà). Il est également possible de changer le type de visibilité en ligne, pour **restreindre l'accès ou permettre le partage**.

Pour ajouter un album, l'utilisateur peut cliquer sur le bouton *Add album* pour ajouter un **album logique**, ou *Add folder* pour ajouter un **dossier du disque**. En faisant un clic droit sur le nom de l'album dans la liste de gauche, il est possible de **le renommer ou de le supprimer**.

5. Gestion des photos

a. Manipulations de base

voir documentation Utilisateur

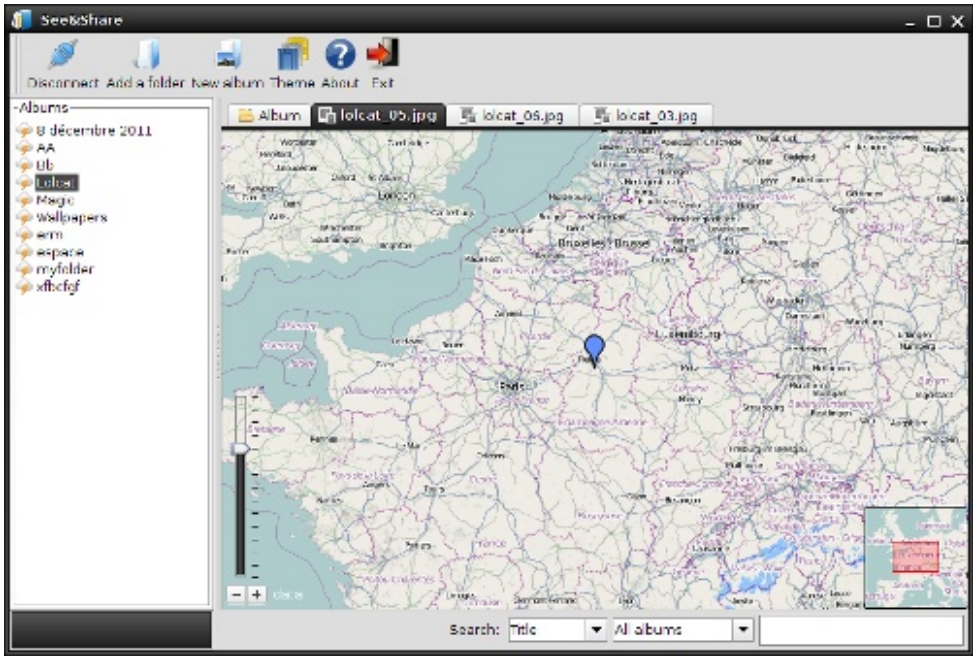
En cliquant sur le bouton "(*i*)", le panneau des détails s'affiche et présente les informations sur l'image :

- **titre**
- **description**
- **tags** (séparés par des espaces)

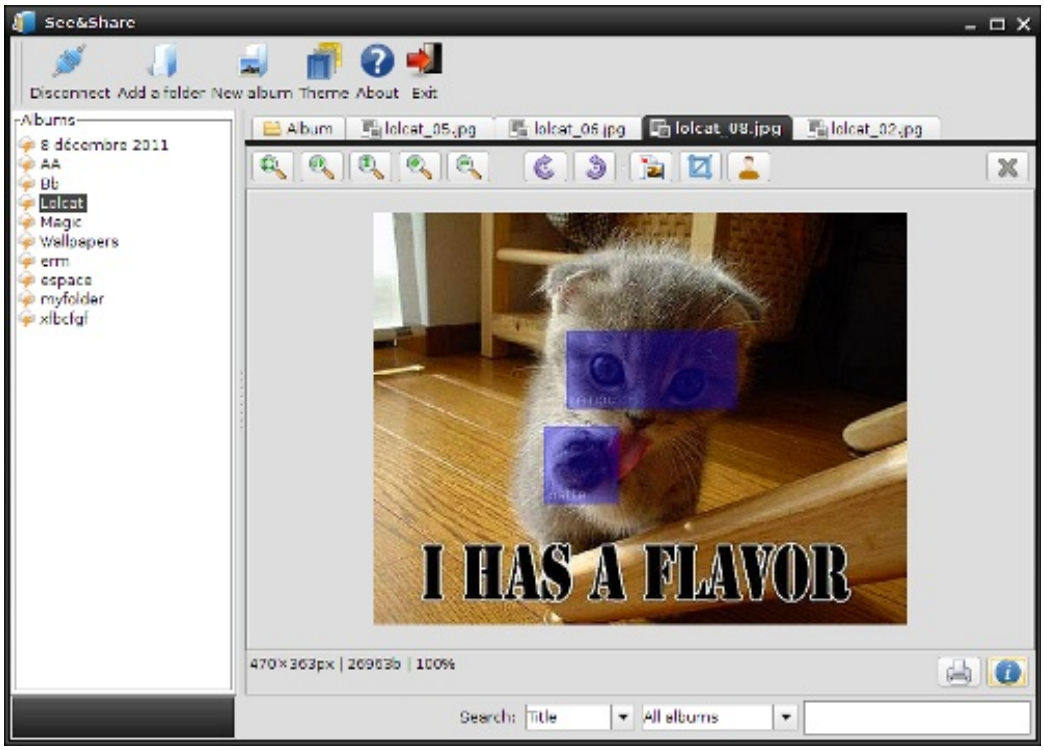
Pour chacun de ces champs, le contenu est **enregistré automatiquement** et en toute transparence pour l'utilisateur dès que celui-ci quitte le champ.
L'utilisateur peut **imprimer la photo** en cliquant sur le bouton avec une icône d'imprimante (cf. la liste des problèmes en fin de document).

b. Manipulations avancées

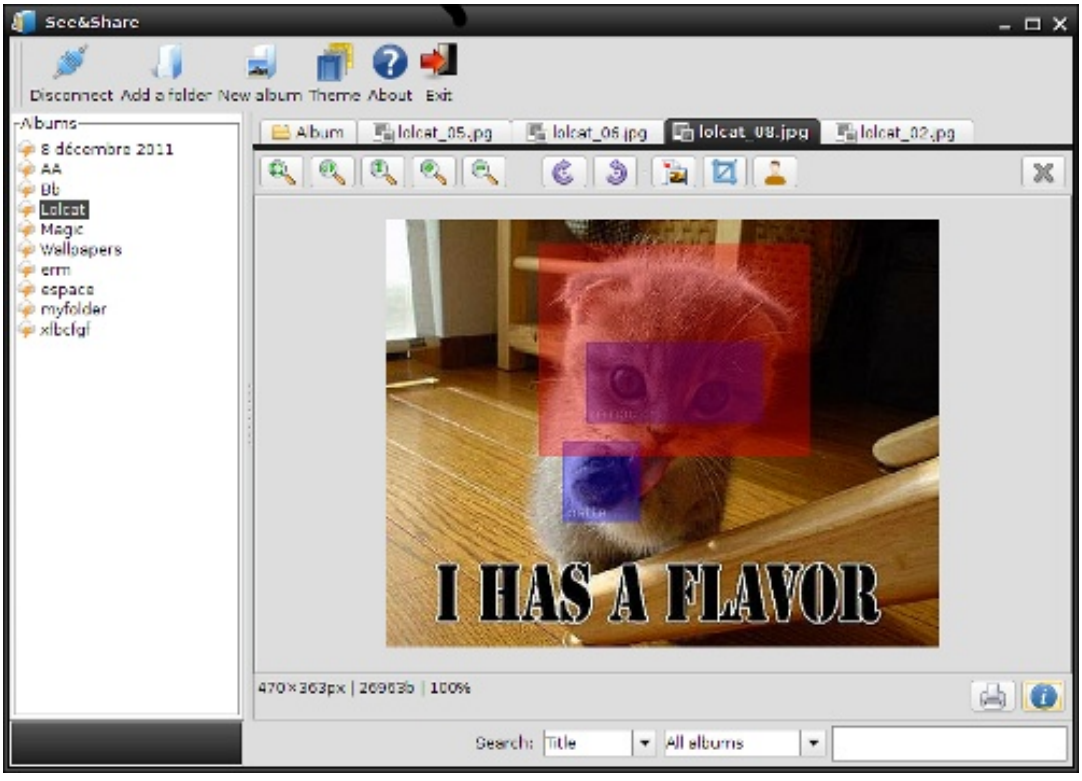
- la **géolocalisation** de n'importe quel média est possible, simplement en cliquant sur le champ correspondant. Une carte mondiale s'affiche (cartes *OpenStreetMap*) et permet de sélectionner précisément un lieu. Une fois le lieu sélectionné en cliquant dessus, la localisation en anglais correspondante aux coordonnées polaires est affichée (base Geocoders de Google) **du numéro de rue jusqu'au pays**.



- le **marquage des personnes** est possible sur n'importe quelle photo. Il suffit de cliquer sur le bouton correspondant (ou de maintenant enfoncée la touche *Ctrl*) et de tracer un rectangle sur la zone concernée. L'utilisateur est alors invité à saisir le nom correspondant, qui sera affiché sur le rectangle directement sur la photo. Les zones de marquage sont entièrement compatibles avec le reste des fonctionnalités (rotations, zoom, etc.). Les zones sont enregistrées automatiquement sur le disque dur, pour tous les types d'album (Google ne propose pas de les exporter via l'API Picasa pour les albums en ligne, elles ne sont donc que locales). Pour supprimer un marquage, il suffit de cliquer sur la zone correspondante en enfonçant la touche *Ctrl*.



- le **recadrage des photos** est possible, simplement en sélectionnant une zone de la photo et en confirmant à l'invite qui s'affiche en lâchant la souris. Pour l'utilisateur, un bouton est disponible ainsi qu'un raccourci (*Ctrl* + *Shift*). Cette fonctionnalité est entièrement compatible avec les zones de marquage, qu'il recalcule ou supprime selon les cas.



c. Ajout d'un média dans un album

Pour ajouter une photo dans un album, l'utilisateur peut cliquer sur le bouton *Add Media,* ou faire un clic droit puis *Add media*, depuis la partie centrale du logiciel.

Il est également possible de déplacer ou copier une photo dans un album en faisant un simple **glisser-déposer** (en maintenant éventuellement la touche *Ctrl* enfoncée pour une copie plutôt qu'un déplacement) de la vignette vers le nom de l'album dans la liste. Cette fonctionnalité est toutefois indisponible depuis les albums en ligne, l'API ne permettant pas ce genre de manipulation.

d. Glisser-déposer externe

En plus du glisser-déposer interne (vignette vers album dans la liste), l'application supporte l'import de fichiers de cette façon.

Ainsi, il suffit de glisser-déposer un fichier depuis un navigateur de fichiers vers la liste des albums :

- Un ou plusieurs fichier(s) (photos ou/et vidéos) en une seule fois
- Un ou plusieurs répertoire(s) en une seule fois (qui seront ajoutés en tant que nouveaux albums)
- Une combinaison de photos, vidéo, dossiers, en une seule fois

6. Lecture de vidéos

La méthode naturelle pour afficher des vidéos dans Java est d'utiliser le *Java Media Framework*. Nous avons implémenté celui-ci entièrement, avant de constater que nous ne pouvions pas lire les flux de Google, et que le nombre de pilotes supportés était extrêmement limité.

Afin d'améliorer l'expérience utilisateur, nous avons préféré adopter une autre stratégie en utilisant le lecteur vidéo par défaut du système de l'utilisateur. Pour les flux de Google des vidéos qui sont en ligne, la plupart des lecteurs ne supportant pas le format Flash, nous lançons le navigateur pour l'afficher.

7. Recherche

Il est possible d'effectuer une recherche dans les albums selon plusieurs critères : recherche dans les titres, les descriptions, les tags, les personnes marquées. La portée de la recherche peut aller d'un album en particulier à l'ensemble des albums.

8. Diaporama

Il est possible de lancer un diaporama qui affiche en boucle et en plein écran les images d'un album ou du résultat d'une recherche.

Le diaporama dispose de plusieurs raccourcis clavier pour contrôler aisément son comportement : *J* et *K* pour naviguer, *P* pour lancer ou arrêter le défilement automatique (toutes les 4s) et la touche *échap* pour quitter. Ces raccourcis sont rappelés durant le diaporama, en transparence au-dessus des photos.

9. Réactivité de l’application

Afin que l’utilisateur ne perde pas patience devant son écran, nous avons utilisé des *threads* pour ne pas bloquer le logiciel pendant les opérations lourdes. Nous avons utilisé des barres de progression qui affichent l’état de chargement d’une opération.

Ainsi, la liste des albums s’affiche rapidement à l’ouverture de l’application et les photos des albums s’affichent progressivement en tâche de fond.

10. Thème Switcher

L’utilisateur peut changer le thème graphique de l’application à tout instant, en choisissant dans une riche collection qu’il a à sa disposition selon ses goûts et son environnement graphique.

III. Architecture

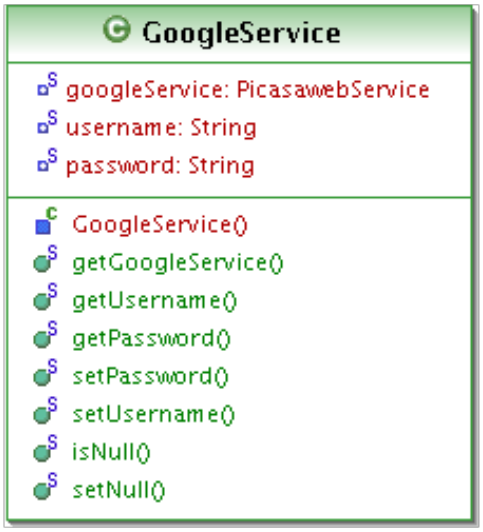
1. Diagramme de classes UML

Le diagramme UML du projet est joint en annexe. De par la difficulté d'organiser (et de lire) un diagramme complet du projet et de ses 60 classes avec tous leurs liens, le diagramme fourni est une version simplifiée. Des extraits complets illustrent cependant les patrons de conception employés, se voulant pertinents.

2. Patrons de conception utilisés

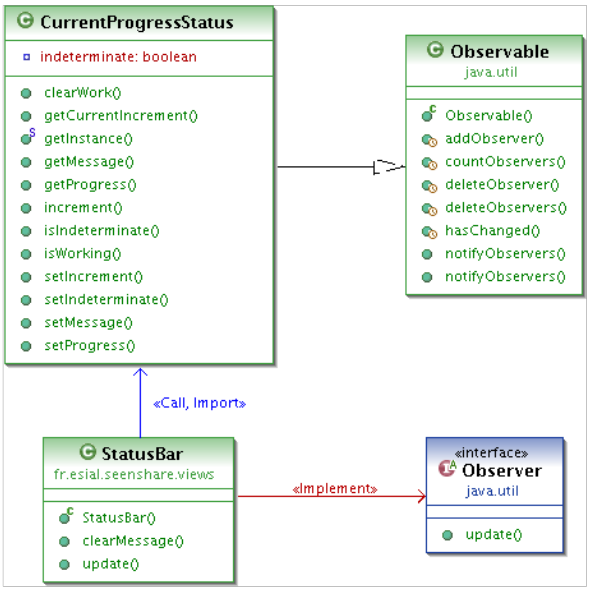
a. Singleton

Nous avons utilisé le patron de conception Singleton pour gérer la connexion au service Google. Ainsi, une seule instance de connexion peut exister à la fois, puisque nous n'avons pas accès au constructeur en dehors de la classe. Nous appelons une méthode qui se charge d'instancier la connexion, ou de retourner la connexion existante.



Nous avons également utilisé un Singleton pour gérer l'instanciation de la liste d'albums : pendant toute une instance d'utilisation du logiciel, il n'existe qu'une seule liste d'albums (qui regroupe tous les albums). Nous pouvions donc utiliser ce patron pour accéder facilement à cette liste, et pour ne l'instancier qu'une seule fois.

La classe *CurrentProgressStatus* permet d'accéder à la barre de statut du logiciel et de modifier son état. Il s'agit également d'un élément qui ne peut être instancié qu'une seule fois, nous avons donc aussi utilisé un Singleton pour cette classe.

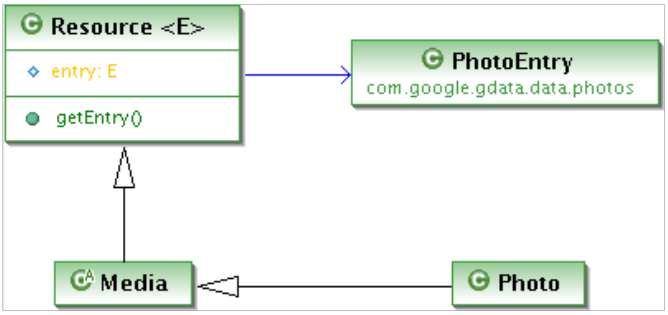


b. Observer

Nous avons utilisé le patron Observer dans la cadre du MVC (Modèle Vue Contrôleur). Tous les éléments de l'interface graphique respectent ce patron, et sont donc des observateurs du modèle associé : lorsque les données du modèle sont modifiées, les vues concernées sont notifiées, puis se mettent à jour.

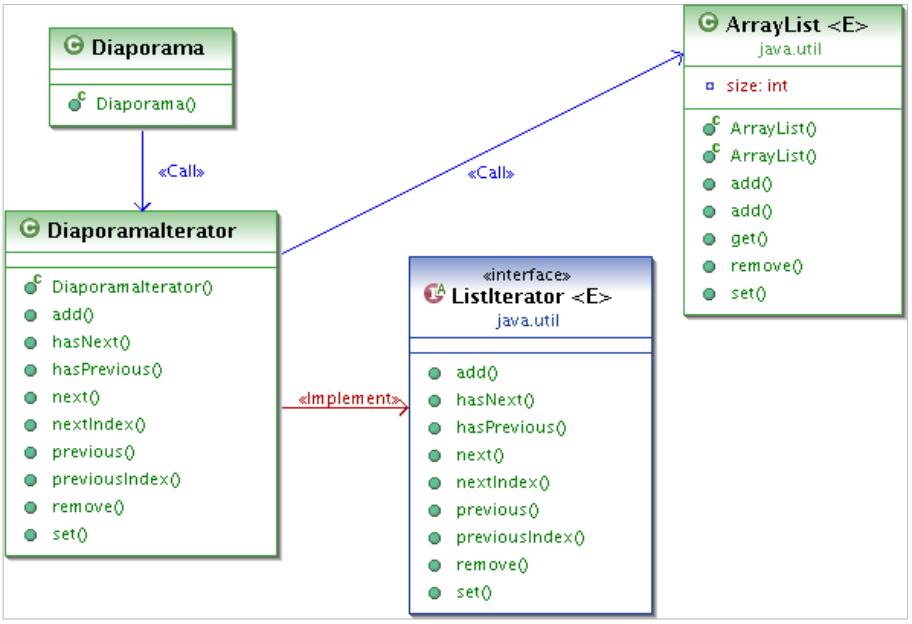
c. Adapter

Nous avons utilisé le patron Adapter dans la classe *Resource* afin de pouvoir utiliser les classes *AlbumEntry* et *PhotoEntry* de l'API Picasa, tout en pouvant exploiter les classes *Album* et *Media* comme nous le souhaitions. La classe *Resource* contient donc un "adaptee", un objet qui contient toutes les informations liées à Picasa dans un entry. Nous pouvons donc ajouter des fonctionnalités à ces classes, comme la Serialization, qui n'auraient pas été possibles sans passer par un adapter.



d. Iterator

Nous avons utilisé le patron Iterator dans la classe *DiaporamaIterator* afin de parcourir la liste des photos d'un album et les afficher.



e. Memento

Utiliser un Memento (avec un Commande) pour gérer l'historique des modifications de photos (redimensionnement, recadrage, gestion des marquages de personnes, etc) aurait permis de proposer à l'utilisateur d'annuler ses actions. Malheureusement, les cours dédiés aux Design Patterns étant intervenus après notre conception et le début du code, nous n'avons pas pu l'intégrer à temps.

3. API utilisées

- Google Picasa Client pour la synchronisation avec Picasa
- JXMapKit OpenStreetMap pour l'affichage d'une carte pour la géolocalisation
- Google Geocoding pour afficher l'adresse de coordonnées géographiques
- Look&Feel Swing pour permettre à l'utilisateur de changer de thème

4. Débogage et tests unitaires

En plus du débogage, au fur et à mesure des projets, nous avons utilisés l'outil *FindBugs*. FindBugs est un logiciel libre d'analyse statique de bytecode Java. Son but est de trouver des bugs dans les programmes Java en identifiant des patterns reconnus comme étant des bugs.

Pour les tests unitaires, nous avons utilisé l'outil JUnit, qui permet de lancer une série de tests unitaires et de mettre en évidence les problèmes du logiciel. Nous avons programmé une série de tests sur les opérations du modèle afin de nous assurer de ne pas introduire de régressions durant le développement du projet. Cependant, il est impossible d'effectuer des tests sur l'interface graphique, notre logiciel étant basé sur une interface, nous n'avons pu tester de cette manière toutes les fonctionnalités du projet.

5. Difficultés rencontrées

- Lors du projet, l'API Google a subi certains changements non documentés ni annoncés. Un exemple notable est celui de la taille des photos récupérées : du jour au lendemain, les photos téléchargées étaient limitées à 512px de large. Nous avons donc envoyé un message sur le forum de support de l'API. Nous avons finalement obtenu la modification à appliquer en discutant avec d'autres groupes travaillant sur le même projet.

- La documentation de l'API Java est peu claire sur l'utilisation des transformations avec l'objet *Graphics* de l'API Java, ne précisant pas que les transformations s'appliquent aux coordonnées de l'espace de dessin.

- Impression des photos : nous avons implémenté l'impression des photos à l'aide de l'API Java, mais celle-ci n'est pas compatible avec Linux : la boîte de dialogue pour sélectionner l'imprimante ne s'ouvre jamais[0]. Voir le bug

- L'API Google Maps est marquée dépréciée par Google[1] qui déconseille donc de l'utiliser dans le cadre d'une nouvelle application. Google étant de moins en moins conciliant[2] avec les applications qui vont chercher directement leurs images de cartes en statique, nous avons décidé d'utiliser l'API JXMapKit de SwingX qui utilise les cartes communautaires de OpenStreetMap. Le second souci a été de trouver les archives de l'API, le site officiel[3] ne proposant que des liens morts (probablement dus à la migration de Sun vers Oracle).

[0] http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=6506286
[1] maps-data-api-deprecation-announcement.html
[2] http://code.google.com/p/gmaps-api-issues/issues/detail?id=1396
[3] http://swinglabs.org/downloads.jsp

IV. Documentation utilisateur

Démarrage : Au lancement du programme, celui-ci vous propose d'entrer vos identifiants de compte Picasa.
Une fois votre compte chargé, la liste des albums sera affichée à gauche. En cliquant sur un album, son contenu sera affiché dans le cadre à droite.

Ajout d'album

-  Ajouter un dossier local
-  Créer un nouvel album vide

Manipulation d'un album
(une fois sélectionné)

-  Ajouter un média dans l'album
-  Afficher un diaporama des médias de l'album

Synchronisation des albums

Le bouton  permet de déclencher la synchronisation de l'album courant.
Dans la liste des albums, l'icône devant chaque album indique le statut de synchronisation :

-  Album en ligne
-  Album/Dossier synchronisé
-  Album/Dossier local

 **Renommer** ou
 **Supprimer** un Album/Média :
clic-droit sur l'élément concerné, et choisir l'action désirée.

Note : il est possible de glisser-déposer un média sur un album pour l'y déplacer (maintenir *Ctrl* enfoncé pour l'y copier)

Fenêtre photo
Après avoir cliqué sur un média d'un album, celui-ci s'ouvre dans un nouvel onglet.

- **Manipuler l'image**

  Zoomer sur l'image à l'aide de la molette de la souris, ou des icônes suivantes :
   Zoomer /  Dézoomer
   Afficher en taille originale
   Adapter à la fenêtre
   Adapter en largeur

  Déplacer l'image en la "glissant - déplaçant" à l'aide de votre souris.

- **Modifier l'image**

   Pivoter l'image vers la droite ou la gauche
   Redimensionner l'image
   Rogner l'image
   *Taguer* une personne dans l'image
    Appliquer un filtre sépia ou N&B
   Flouter l'image
   Afficher le panneau des informations, contenant le titre, les tags, la géolocalisation et la description du média
   Fermer le média et enregistrer ses modifications

V. Crédits

Merci à Hernàn Vanzetto pour l'aide et le suivi qu'il a apporté à ce projet.

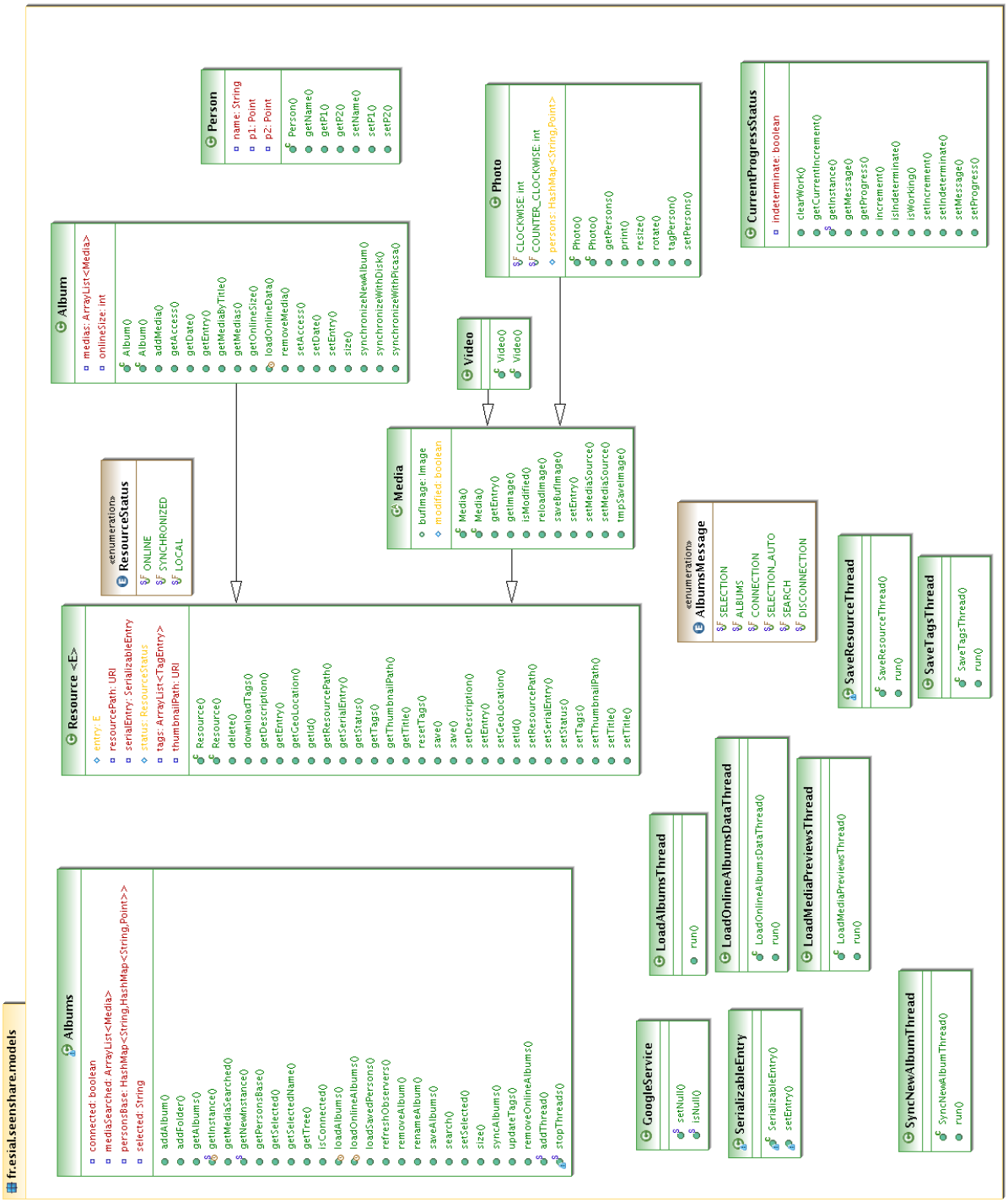
Nous avons utilisé lors du développement de notre projet, les sites suivants :

- Documentation de l'API Picasa
- Documentation Java
- Stack Overflow
- Example Depot
- Icon Finder, où nous avons trouvé nos icônes
- Le projet OpenStreetMap pour la géolocalisation

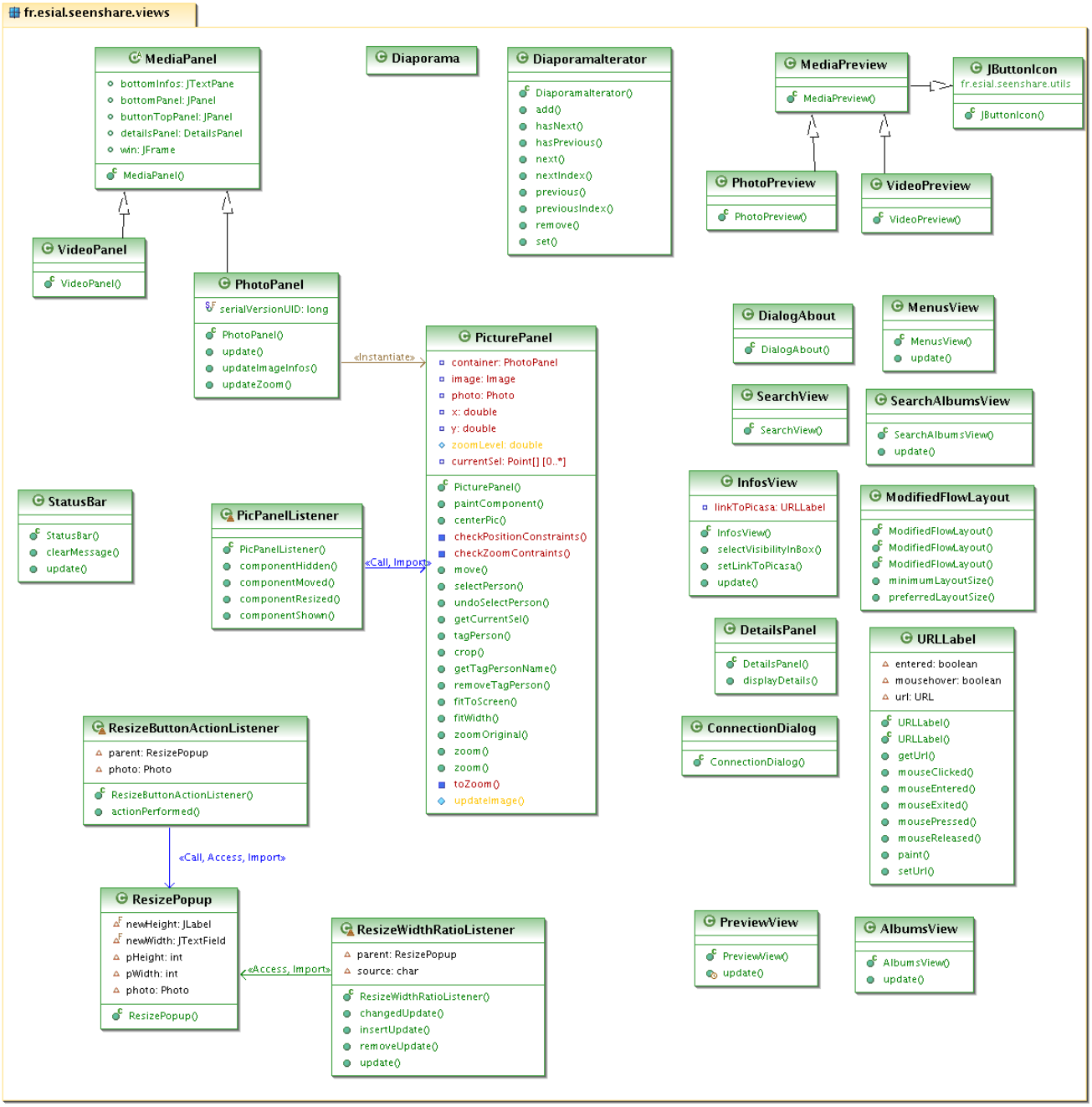
Annexes

A. Diagrammes UML

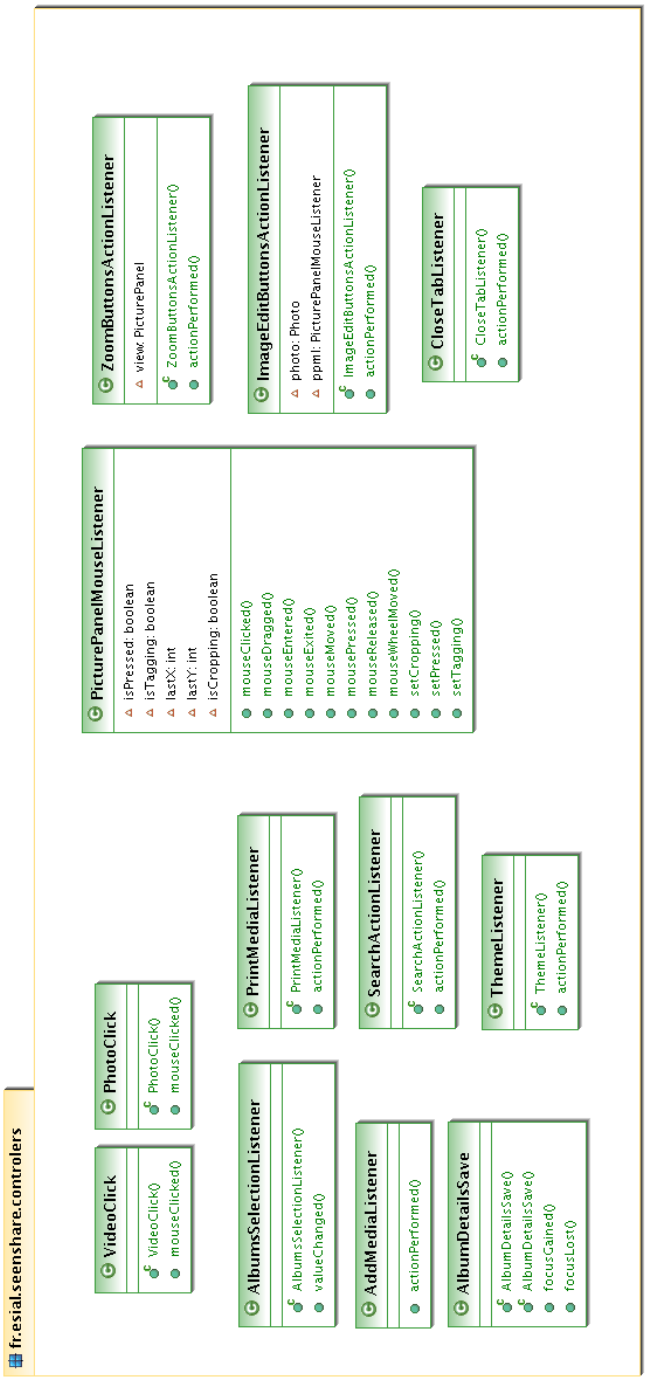
1. Modèles



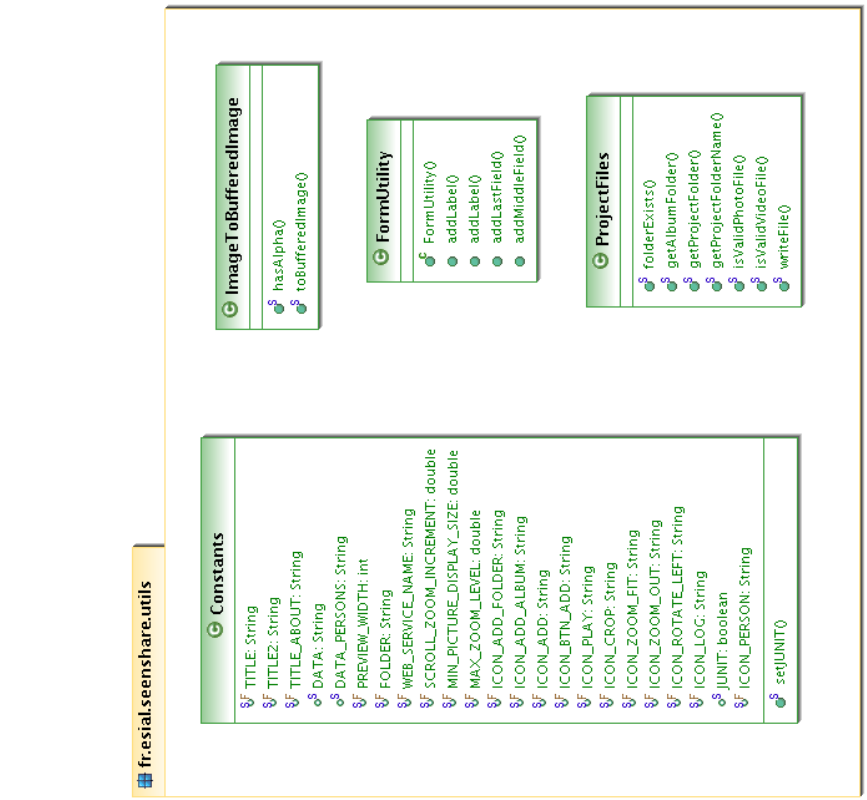
2. Vues

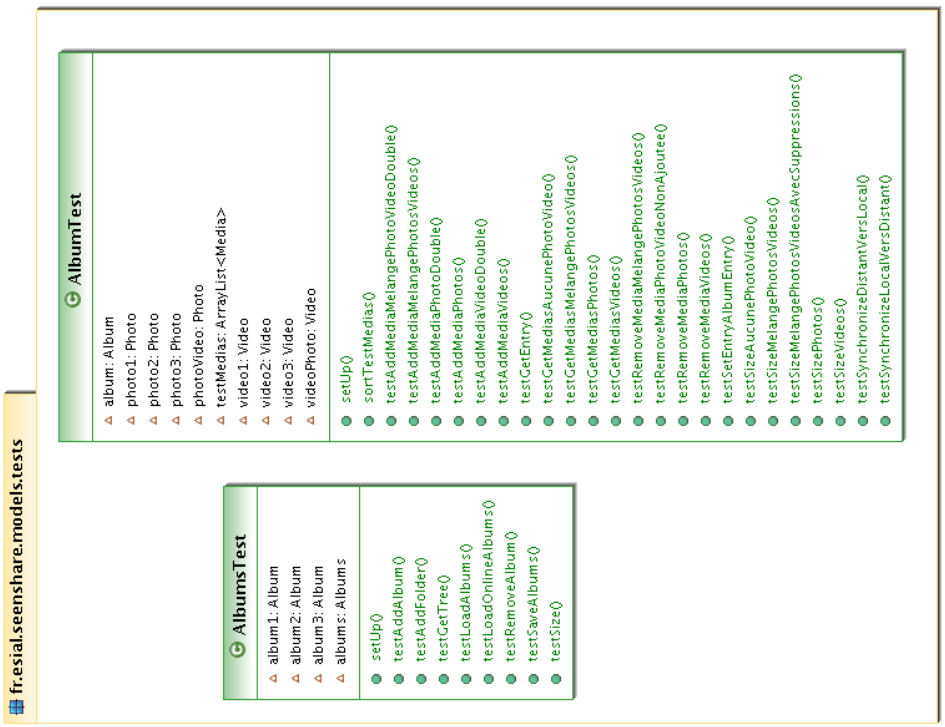

fr.esial.seenshare.views
MediaPanel
bottomInfos: JTextPane
bottomPanel: JPanel
buttonTopPanel: JPanel
detailsPanel: DetailsPanel
win: JFrame
MediaPanel()
Diaporama
Diaporamalterator
Diaporamalterator()
add()
hasNext()
hasPrevious()
next()
nextIndex()
previous()
previousIndex()
remove()
set()
MediaPreview
MediaPreview()
JButtonIcon
fr.esial.seenshare.utils
JButtonIcon()
PhotoPreview
PhotoPreview()
VideoPreview
VideoPreview()
VideoPanel
VideoPanel()
PhotoPanel
serialVersionUID: long
PhotoPanel()
update()
updateImageInfos()
updateZoom()
«Instantiate»
PicturePanel
container: PhotoPanel
image: Image
photo: Photo
x: double
y: double
zoomLevel: double
currentSel: Point[] [0..*]
PicturePanel()
paintComponent()
centerPic()
checkPositionConstraints()
checkZoomContraints()
move()
selectPerson()
undoSelectPerson()
getCurrentSel()
tagPerson()
crop()
getTagPersonName()
removeTagPerson()
fitToScreen()
fitWidth()
zoomOriginal()
zoom()
zoom()
toZoom()
updateImage()
DialogAbout
DialogAbout()
MenusView
MenusView()
update()
SearchView
SearchView()
SearchAlbumsView
SearchAlbumsView()
update()
StatusBar
StatusBar()
clearMessage()
update()
PicPanelListener
PicPanelListener()
componentHidden()
componentMoved()
componentResized()
componentShown()
«Call, Import»
InfosView
linkToPicasa: URLLabel
InfosView()
selectVisibilityInBox()
setLinkToPicasa()
update()
ModifiedFlowLayout
ModifiedFlowLayout()
ModifiedFlowLayout()
ModifiedFlowLayout()
minimumLayoutSize()
preferredLayoutSize()
DetailsPanel
DetailsPanel()
displayDetails()
URLLabel
entered: boolean
mousehover: boolean
url: URL
URLLabel()
URLLabel()
getUrl()
mouseClicked()
mouseEntered()
mouseExited()
mousePressed()
mouseReleased()
paint()
setUrl()
ConnectionDialog
ConnectionDialog()
ResizeButtonActionListener
parent: ResizePopup
photo: Photo
ResizeButtonActionListener()
actionPerformed()
«Call, Access, Import»
ResizePopup
newHeight: JLabel
newWidth: JTextField
pHeight: int
pWidth: int
photo: Photo
ResizePopup()
«Access, Import»
ResizeWidthRatioListener
parent: ResizePopup
source: char
ResizeWidthRatioListener()
changedUpdate()
insertUpdate()
removeUpdate()
update()
PreviewView
PreviewView()
update()
AlbumsView
AlbumsView()
update()

3. Contrôleurs



4. Autres


fr.esial.seenshare.utils
Constants
TITLE: String
TITLE2: String
TITLE_ABOUT: String
DATA: String
DATA_PERSONS: String
PREVIEW_WIDTH: int
FOLDER: String
WEB_SERVICE_NAME: String
SCROLL_ZOOM_INCREMENT: double
MIN_PICTURE_DISPLAY_SIZE: double
MAX_ZOOM_LEVEL: double
ICON_ADD_FOLDER: String
ICON_ADD_ALBUM: String
ICON_ADD: String
ICON_BTN_ADD: String
ICON_PLAY: String
ICON_CROP: String
ICON_ZOOM_FIT: String
ICON_ZOOM_OUT: String
ICON_ROTATE_LEFT: String
ICON_LOG: String
JUNIT: boolean
ICON_PERSON: String
setJUNIT()
ImageToBufferedImage
hasAlpha()
toBufferedImage()
FormUtility
FormUtility()
addLabel()
addLabel()
addLastField()
addMiddleField()
ProjectFiles
folderExists()
getAlbumFolder()
getProjectFolder()
getProjectFolderName()
isValidPhotoFile()
isValidVideoFile()
writeFile()


fr.esial.seenshare.models.tests
AlbumTest
album: Album
photo1: Photo
photo2: Photo
photo3: Photo
photoVideo: Photo
testMedias: ArrayList<Media>
video1: Video
video2: Video
video3: Video
videoPhoto: Video
setUp()
sortTestMedias()
testAddMediaMelangePhotoVideoDouble()
testAddMediaMelangePhotosVideos()
testAddMediaPhotoDouble()
testAddMediaPhotos()
testAddMediaVideoDouble()
testAddMediaVideos()
testGetEntry()
testGetMediasAucunePhotoVideo()
testGetMediasMelangePhotosVideos()
testGetMediasPhotos()
testGetMediasVideos()
testRemoveMediaMelangePhotosVideos()
testRemoveMediaPhotoVideoNonAjoutee()
testRemoveMediaPhotos()
testRemoveMediaVideos()
testSetEntryAlbumEntry()
testSizeAucunePhotoVideo()
testSizeMelangePhotosVideos()
testSizeMelangePhotosVideosAvecSuppressions()
testSizePhotos()
testSizeVideos()
testSynchronizeDistantVersLocal()
testSynchronizeLocalVersDistant()
AlbumsTest
album1: Album
album2: Album
album3: Album
albums: Albums
setUp()
testAddAlbum()
testAddFolder()
testGetTree()
testLoadAlbums()
testLoadOnlineAlbums()
testRemoveAlbum()
testSaveAlbums()
testSize()

B. Feuilles de temps

Guillaume GÉRARD (107 heures)

- 5h - Conception
- 20h - Projet de base, mise en place d'un premier modèle avec ses design patterns
- 50h - Interface Graphique (fenêtres, mise en page, icônes, écouteurs)
- 10h - Diaporama (automatisé)
- 4h - Lecteur vidéo
- 4h - Édition avancée d'une image (Sépia, Noir et blanc et flou)
- 4h - Changeur de thème : 4 heures
- 10h - Débogage
- 10h - Débogage du projet avec FindBugs

Julien VAUBOURG (101 heures)

- 5h - Conception
- 10h - Classe de tests
- 5h - Glisser-déposer d'un album à un autre
- 4h - Lecteur vidéo
- 30h - Géolocalisation et Géocoding
- 2h - Icônes de status dans la liste des albums
- 15h - Interface Graphique (onglet, page de détail)
- 20h - Tags des personnes
- 2h - Édition avancée d'une image (recadrage)
- 5h - Glisser-déposer du système à l'application

Julien GUÉPIN (110 heures)

- 5h - Conception
- 40h - Adaptation du modèle à l'API Picasa
- 10h - Interface Graphique (ajout nouvel album, page de détails)
- 15h - Threads des différents chargements
- 15h - Synchronisation des albums et sauvegarde locale
- 5h - Javadoc
- 20h - Débogage

Adrien SIEBERT (75 heures)

- 5h - Conception
- 15h - Interface graphique
- 20h - Affichage d'une image (zoom/déplacement)
- 10h - Édition basique d'une image (rotation, redimensionnement)
- 2h - Barre de statut (feedback utilisateur)
- 10h - Débogage
- 5h - Suivi UML