Guide

Sopra Steria

HandsOn MDK Android

Hands-on MDK Android - Mes notes de frais

Version 1.01 du vendredi 5 février 2016

Historique

Version	Date	Origine de la mise à jour	Rédigée par	Validée par
1.00	8/12/15	Refonte du document	Antoine Belliard	Sébastien Maitre
1.01	5/2/16	Passe de corrections	Quentin Lagarde	Sébastien Maitre

Sommaire

1. Introduction

Dans le cadre de la session de formation « Hands-on MDK », vous allez apprendre à utiliser le « Mobile Development Kit » afin de générer, surcharger et personnaliser une application pour la plateforme mobile *Android*.

L'application qui servira de fil conducteur à cette formation s'intitule « Notes de frais ». Comme son nom l'indique, elle vous permettra de gérer des notes de frais, ainsi que leurs dépenses associées.

1.1. Présentation de l'application

L'application « Notes de frais » sera principalement axée autour de quatre écrans :

- **L'écran principal**, permettant de naviguer vers les trois écrans secondaires ci-dessous, à l'aide de trois boutons.
- **L'écran secondaire** « **Notes de frais »**, qui regroupe :
  - Un affichage en liste des notes de frais par client, illustré à gauche en Figure 1.
  - Un affichage détaillé et partiellement éditable d'une note de frais et de ses dépenses, illustré au milieu en Figure 1.
  - Un affichage totalement éditable d'une dépense, illustré à droite en Figure 1.
- **L'écran secondaire « A propos »**, qui contiendra une page Web présentant l'application.
- **L'écran secondaire « Carnet de voyage »**. Pour l'instant, cet écran secondaire est vide.

1.2. Plan de la réalisation

Comme indiqué précédemment, la réalisation de l'application se découpera en trois grandes étapes :

1. La création du projet et la génération de l'application.
2. La surcharge métier et la personnalisation visuelle de l'application.
3. La mise en place de la synchronisation de l'application avec un serveur.

Seule la première étape est obligatoire. Les deux suivantes peuvent être réalisées dans l'ordre souhaité.

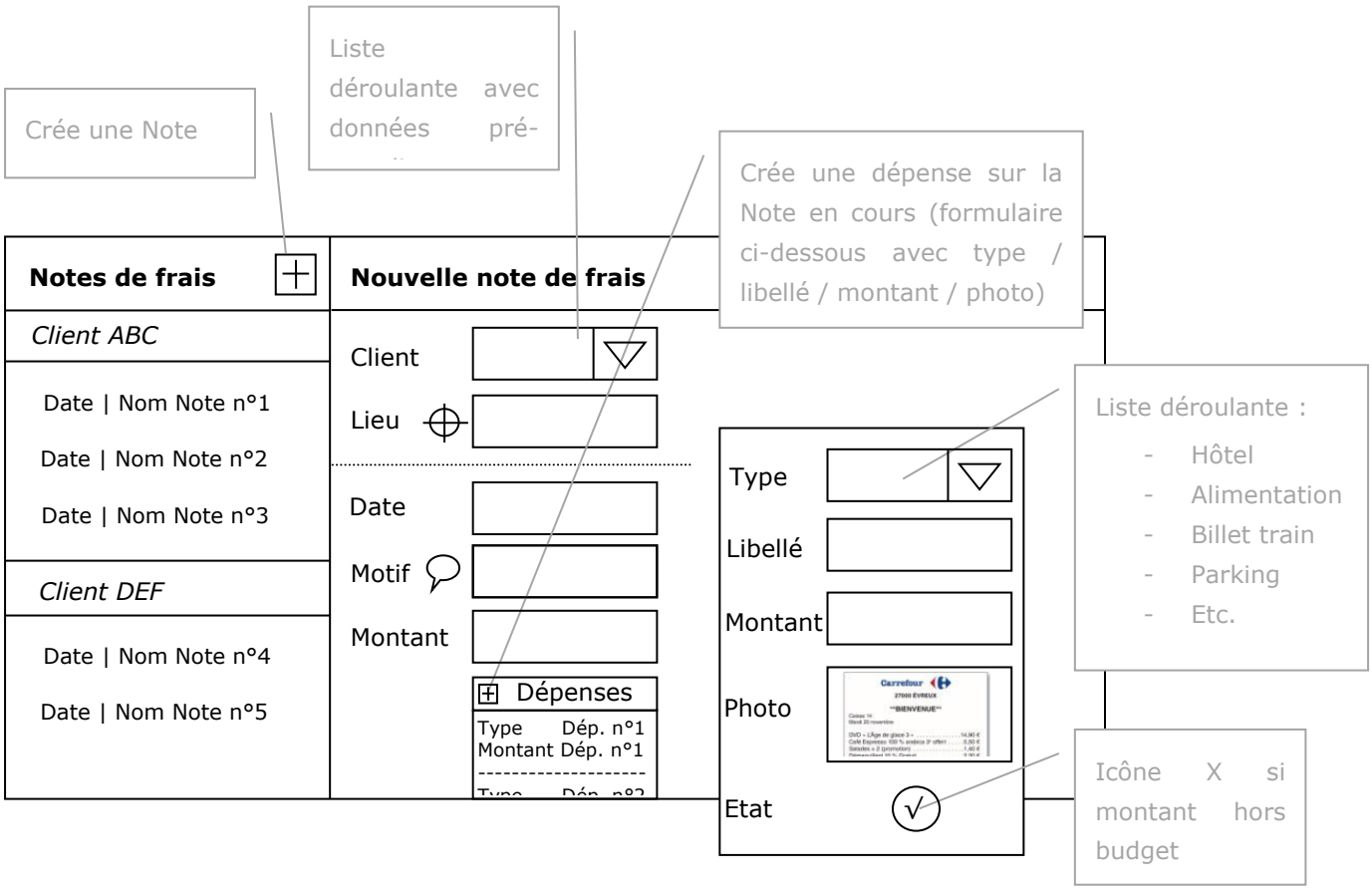

Figure 1 : Principe de l'écran secondaire « Notes de frais ».

1.3. Prérequis

Pour suivre cette formation, vous devez :

- Installer la *Command Line Interface* du mdk :
  - Installer Node.js
  - Installer mdk-cli :

```
$ npm install -g mdk-cli
```

- Installer les outils utilisés par le MDK pour réaliser des applications Android :

```
$ mdk tools-install android
```

- Installer Android Studio (**la version qui n'intègre pas le sdk android**)
  - Suite à l'installation, lancer Android Studio
  - Afficher les préférences
  - **Dans les étapes qui suivent *$HOME* est à remplacer par le chemin vers votre répertoire utilisateur**
  - Définir l'emplacement du *SDK Android* :
    - Rechercher « Android SDK »
    - Modifier la valeur du champ « Android SDK Location » par *$HOME/.mdk/tools/android-sdk*
  - Modifier la configuration de *gradle* :
    - Rechercher « gradle »
    - Modifier la valeur du champ « Service directory path » par *$HOME/.mdk/tools/gradle*
- Installer un outil de **modélisation UML** :
  - **MagicDraw** UML v17.0.5
    http://www.nomagic.com/products/magicdraw.html

Pour tester l'application, nous vous conseillons également d'utiliser l'émulateur *Android* officiel avec une image système *Intel x86* récente, ainsi que l'accélération matérielle (cf. http://developer.android.com/tools/devices/emulator.html#accel-vm).

Vous pouvez également utiliser votre propre terminal *Android* si vous en possédez un.

2. Création du projet

Tout projet *MDK* utilise une arborescence de fichiers et dossiers propre à cet outil. Sa création est la première étape de notre projet.

À l'aide d'une invite de commande :

1. Se placer dans un dossier qui contiendra le projet

> ⓘ *Sous Windows, il est vivement conseillé d'utiliser un chemin avec peu de caractères (par exemple **D:\dev**). En effet, les chemins générés par le SDK Android peuvent rapidement atteindre les limitations de ce système d'exploitation.*

2. Initialiser le projet à l'aide de la commande suivante :

```
$ mdk create com.soprasteria.mdk.handson.myexpenses -t "My Expenses Tuto"
```

   a. *com.soprasteria.mdk.handson.myexpenses* est l'identifiant unique de l'application
   b. L'option *–t* est facultative. Elle permet de définir un *template* qui sera appliqué au projet généré. Ici, cela permet d'appliquer un thème à l'application et d'ajouter un répertoire contenant les surcharges qui seront utilisées pendant ce *Hands-on*.
   c. Lorsque vous y êtes invité, saisir un login et un mot de passe
3. Ajouter la plateforme Android

```
$ cd myexpenses

$ mdk platform-add android
```

   d. Le dossier « ***myexpenses*** » du projet est créé. Son contenu doit être similaire à celui illustré dans la Figure 2.

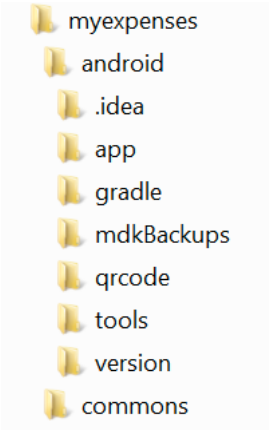

Figure 2 : Arborescence du dossier « myexpenses ».

3. Génération de l'application à l'aide du modèle UML

Une fois le projet créé, il est nécessaire de modéliser l'application dans des diagrammes UML de classes *UML* puis de générer.

3.1. Génération à partir du modèle fourni

1. Écraser le fichier *myexpenses/commons/modelisation.xml* en utilisant le fichier présent dans le répertoire *myexpenses/hands-on/android/31 - Generation.*
2. Générer et compiler l'application

```
$ mdk platform-build android
```

   a. L'application ainsi construite se trouve dans le répertoire *myexpenses/android/app/build/outputs/apk/*,
   b. elle peut être exécutée sur un terminal/émulateur

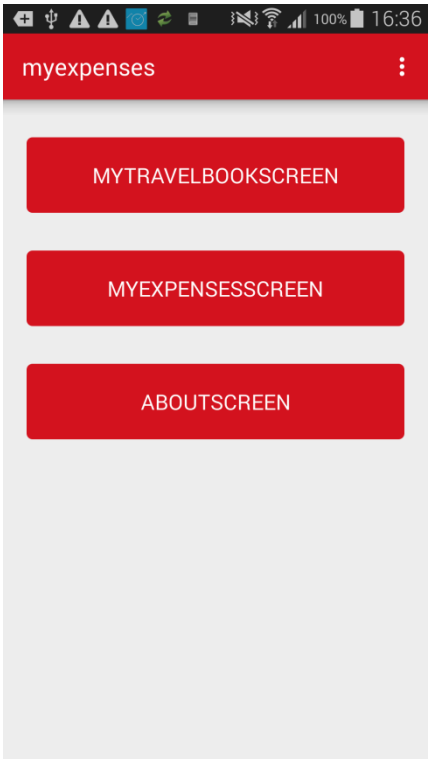

Figure 3 : Ecran principal de l'application générée à partir du modèle UML complet.

3. Cliquer sur le bouton *MYEXPENSESSCREEN*. L'écran qui s'affiche est totalement vide, car l'application ne possède pour le moment aucune donnée. Nous verrons à la section 3.3 comment ajouter des données à l'application générée

3.2. Analyse du modèle UML utilisé

Ouvrir le modèle UML de l'application (*myexpenses/commons/modelisation.xml*) dans *MagicDraw*

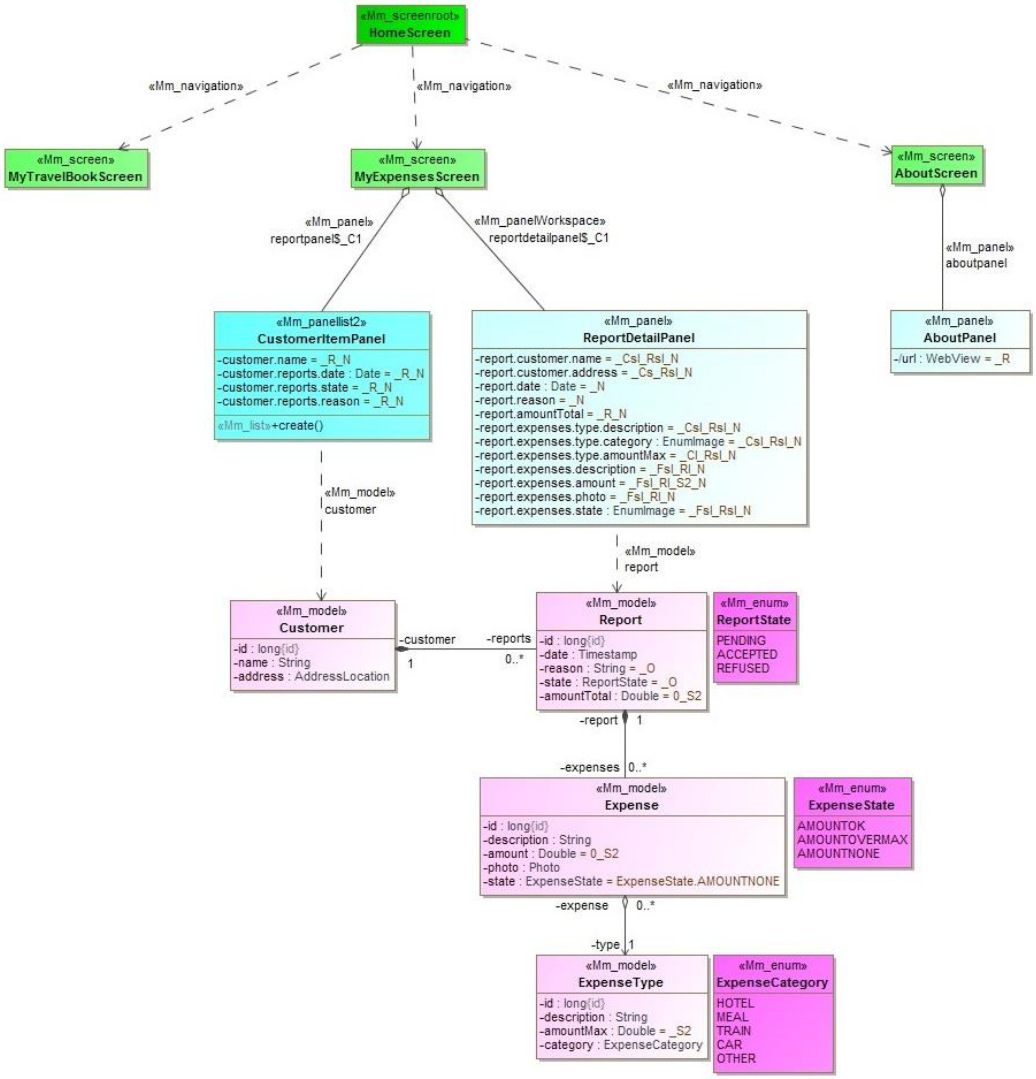

Figure 4 : Modèle UML complet.

La modélisation pour le *MDK* se base sur une modélisation *UML* classique, à laquelle est associée plusieurs mécanismes dont :

- L'usage de stéréotypes UML sur les classes et navigations.
- L'usage des types du *MDK* pour les attributs.
- L'usage de valeurs par défaut avec une syntaxe particulière, afin de paramétrer les attributs.

Le modèle de l'application se lit en couches avec, de bas en haut :

2. La couche modèle métier », correspondant aux entités Métier de l'application : Elle contient les classes stéréotypées « Mm_model » illustrées en Figure 5. Ici, on retrouve :
    a. Un client (***Customer***) qui occasionne des notes de frais (***Report***).
    b. Chaque note se compose de dépenses (**Expense**).
    c. Tout dépense est typée (***ExpenseType***)
    d. Certaines classes possèdent un ou plusieurs attributs de type énumérés :
        - Les énumérations ***NoteState*** et ***ExpenseState*** et ***ExpenseCategory*** représentent ces types particuliers, et sont stéréotypées en « ***Mm_enum*** ».
        - Contrairement aux entités Métier, les énumérations définissent un ensemble fini et figé au moment de la modélisation

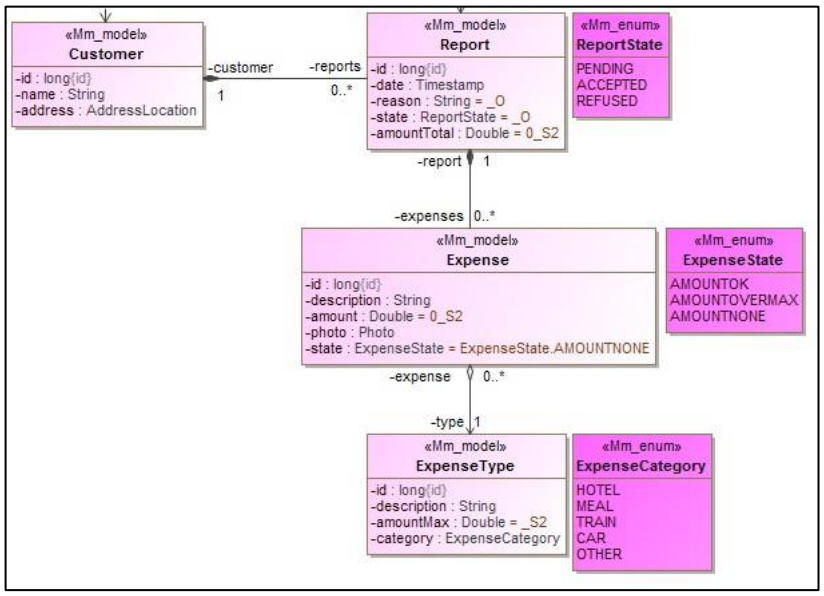

Figure 5: Classes stéréotypées « Mm_model ».

3. « La couche ***panel*** », décrivant des parties d'écrans de l'application : Elle contient les classes stéréotypées « ***Mm_panel***[…] », illustrées en Figure 6. On retrouve ici trois classes :
    a. **NotePanel** présente une liste de certains attributs de la classe Client et Note.
    b. **NoteDetailPanel** présente le détail d'une Note de frais, permettant de consulter et/ou de modifier certains attributs des classes Client, Note, Expense et ExpenseType.
    c. **AboutPanel** présente une page Web accessible depuis l'application.

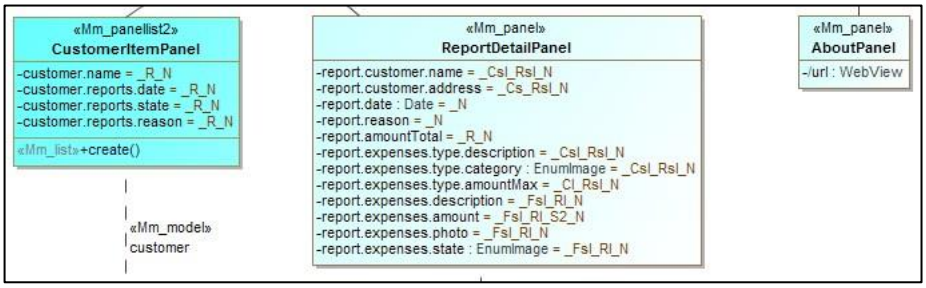

Figure 6 : Classes stéréotypées en « Mm_panel[…] ».

4. « La couche ***screen*** », permettant de modéliser le contenu des écrans et la navigation eux :
   a. Elle contient les classes stéréotypées « ***Mm_screen*** » et une classe stéréotypée « ***Mm_screenroot*** », illustrées en Figure 7.

On retrouve ici trois classes stéréotypées « ***Mm_screen*** », correspondant à des écrans (qui comprennent les « ***Mm_panel*** » abordés ci-dessus) :

   b. **MonCarnetDeVoyageScreen** est un écran vide, car il n'est composé d'aucun panel.
   c. **MesNotesScreen** se compose d'un panel de type liste, NotePanel. En choisissant un élément de la liste, on affiche un panel de détail, NoteDetailPanel.
   d. **AboutPanel** se compose d'un panel qui affiche une page Web.

Enfin, on retrouve l'unique classe stéréotypée « Mm_screenroot », correspondant à l'écran principal de l'application affiché au lancement. Ici, il s'agit de la classe **MesNotesDeFrais**. Cet écran principal permet de naviguer vers différents autres écrans, abordés ci-dessus.

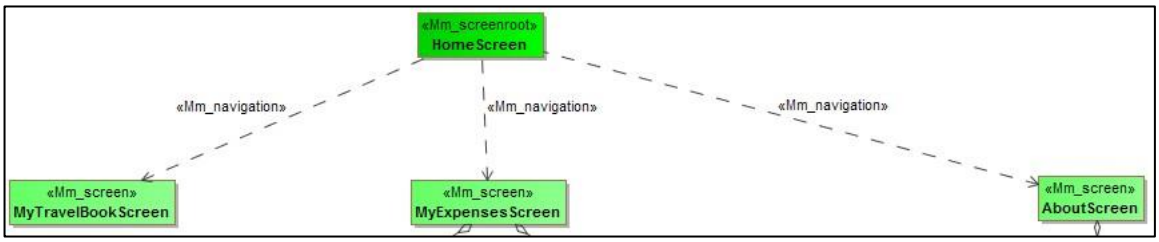

Figure 7 : Classes stéréotypées en « Mm_screen » et « Mm_screenroot ».

3.3. Jeu de données

La mise en place d'un jeu de données est une étape facultative lors de la création d'une application. Cela peut cependant s'avérer utile lors des premières itérations de développement si aucun service web ne peut être interrogé pour alimenter l'application.

Le *MDK* propose un outil permettant d'initialiser un jeu de données à l'aide d'un classeur *Excel* : *myexpenses/commons/MOV-MM-Referentiel_de_donnees-V1.0.xls*.

En quelques étapes, ce dernier permet de constituer un fichier reprenant l'ensemble des entités métiers du modèle UML et de renseigner pour chacune d'entre elle des données qui seront injectées lors du premier lancement de l'application.

Il permet également d'importer et d'exporter les données au format CSV.

Dans le cadre de ce hands-on un fichier Excel a déjà été pré-rempli :

5. Ouvrir *myexpenses/hands-on/android/33 – Datas/MOV-MM-Referentiel_de_donnees-V1.0.xls*
6. Dans l'onglet « Admin », cliquer sur « Exporter »
7. Un fichier nommé *sqlitecreate_userdata* est généré
8. Remplacer le fichier *myexpenses/android/app/src/main//res/raw/sqlitecreate_userdata* par celui-ci
9. Compiler l'application

```
$ mdk platform-compile android
```

10. Désinstaller la précédente version de l'application sur votre terminal/émulateur ou supprimer ses données
11. Installer puis exécuter l'application ainsi construite qui se trouve dans le répertoire *myexpenses/android/app/build/outputs/apk/*
12. Cliquer sur le bouton *MyExpensesScreen*
    a. La liste des notes de frais est maintenant alimentée. Son affichage doit être similaire à celui illustré en Figure 8.

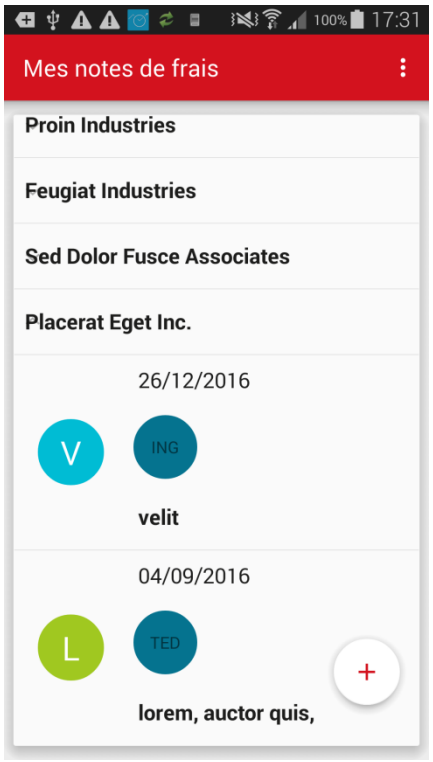

Figure 8 : Panel présentant le jeu de données (Clients et Notes de frais).

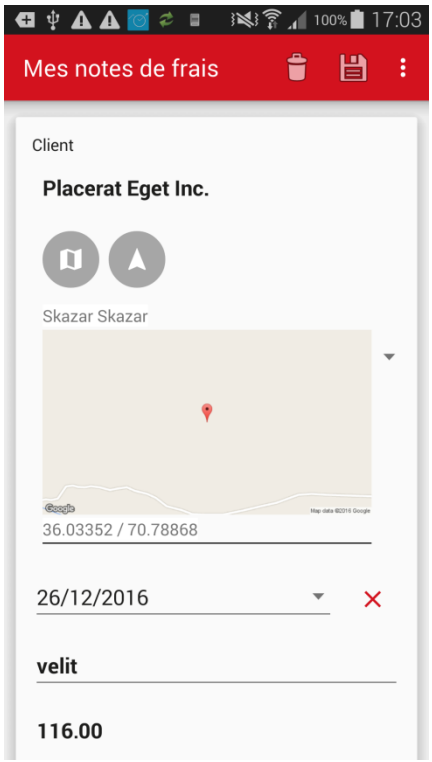

Figure 9 : Panel présentant le détail d'une Note de frais.

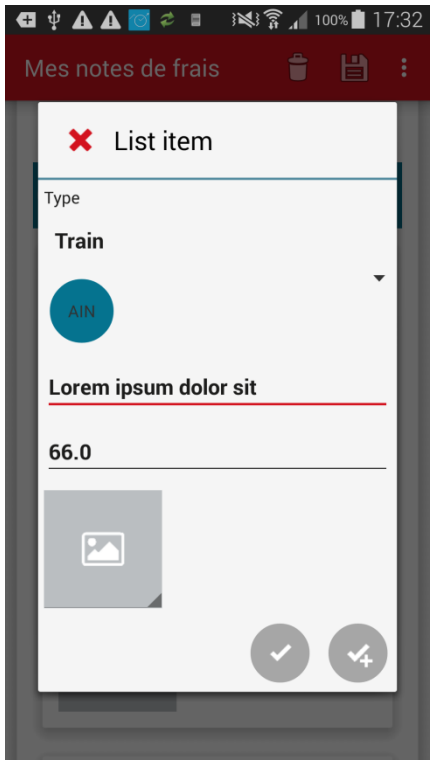

Figure 10 : Panel de modification d'une Expense.

13. Cliquer sur un Client de la liste/ Les Notes de frais associées au Client apparaissent en dessous.
14. Cliquer sur une Note de frais pour en afficher le détail. Son affichage doit être similaire à celui illustré en Figure 9.
15. Cliquer sur une dépense (*Expense*) de la note de frais en bas de l'écran. L'écran de modification de la dépense apparaît. Son affichage doit être similaire à celui illustré en Figure 10.

4. Personnalisation de l'application

Une fois l'application générée avec le *MDK*, il est possible de personnaliser certains éléments afin de se rapprocher d'une application finalisée.

4.1. Personnalisation des libellés

Le projet généré possède un fichier nommé *dev__project_labels.xml* dans le répertoire *myexpenses/android/app/src/main/res/values*. Le contenu de ce fichier est déduit du modèle UML :

- Un libellé par écran
- Un libellé par bouton généré
- Un libellé pour chaque attribut de chaque panel ne possédant pas, dans sa valeur par défaut, l'option « _N » (No label)
- Un libellé pour chaque littéral d'une énumération
- …

De plus, il est possible, comme pour toute application Android, de définir ces libellés pour de multiples langues.

1. Copier
    a. Le contenu du répertoire *myexpenses/hands-on/android/41 – Labels/*
    b. Dans *myexpenses/android/app/src/main/*
2. Compiler l'application avec la commande

```
$ mdk platform-compile android
```

3. Déployer et exécuter l'application
4. Sur le terminal, vérifier que les libellés ont été pris en compte.

4.2. Suppression d'un libellé

Le but ici est de démontrer qu'une modification du modèle UML n'aboutit pas à une perte totale de la surcharge des libellés.

1. Écraser le fichier *myexpenses/commons/modelisation.xml* en utilisant le fichier présent dans le répertoire *myexpenses/hands-on/android/42 – Delete label.*
    a. Ce modèle diffère du précédent par l'ajout de l'option *_N* sur le champ *url* du panel nommé *AboutPanel*
2. Générer et compiler l'application

```
$ mdk platform-build android
```

    b. L'application ainsi construite se trouve dans le répertoire myexpenses/android/app/build/outputs/apk/, elle peut être exécutée sur un terminal/émulateur
3. Installer et exécuter l'application
4. Vérifier que le libellé de l'unique champ de l'écran « À propos » a disparu
    a. Dans le fichier *myexpenses/android/app/src/main/res/values/dev__project__labels*

b. Lors de l'affichage de l'écran « À propos »

5. Surcharge métier de l'application

Une fois l'application générée avec le *MDK*, il est possible d'y ajouter tout type de fonctionnalités via des surcharges. Ce chapitre permet d'intégrer les différentes surcharges métier à l'application.

Les surcharges métiers sont réalisées en java, à partir de cette section nous conseillons d'importer le projet sous Android Studio.

5.1. Suppression d'une note de frais

5.1.1. Présentation de la surcharge

L'objectif de la surcharge est de générer une nouvelle action de suppression d'une note de frais. Lorsque l'action de suppression s'est déroulée avec succès, l'utilisateur doit en être notifié.

5.1.2. Implémentation de la surcharge

Première étape : génération de l'action

1. Écraser le fichier *myexpenses/commons/modelisation.xml* en utilisant le fichier *modelisation.xml* présent dans le répertoire *myexpenses/hands-on/android/51 – Delete report.*
   c. Ce modèle diffère du précédent par l'ajout de l'opération *deleteDetail* au panel *ReportDetailPanel*
2. Générer l'application à l'aide de la commande

```
$ mdk platform-gensrc android
```

   a. Une fois le code généré, vérifier l'existence de la classe *DeleteReportDetailPanel* qui gère la suppression de la note affichée

Deuxième étape : écouter la fin de l'action et afficher la notification

3. Ouvrir le fichier nommé *ReportDetailPanel.java* présent dans le répertoire :
   a. *myexpenses/android/app/src/main/java/com/soprasteria/mdk/handson/myexpenses/panel/*
   b. *ReportDetailPanel.*java représente un panel du modèle UML sous la forme d'un fragment Android
4. Rechercher le commentaire *//@non-generated-start[methods]*
5. Copier le code ci-dessous à sa suite :

```
@ListenerOnActionSuccess(action = DeleteReportDetailPanel.class)
public void
onDeleteReportSuccess(ListenerOnActionSuccessEvent<EntityActionParameterImpl<Report>>
event) {
      Toast.makeText(this.getContext(), R.string.delete_report_confirm,
Toast.LENGTH_LONG).show();
}
```

a. L'annotation *@ListenerOnActionSuccess* indique que la méthode sur laquelle elle est définie réagit au succès de l'exécution de l'action dont le nom est précisé grâce à son paramètre action
   i. Il existe également une annotation *@ListenerOnActionFail* pour réagir à l'échec de l'exécution d'une action
b. Le paramètre de la méthode dépend de l'annotation et du type en sortie de l'action :
   i. *ListenerOnActionSuccessEvent* car l'annotation est *@ListenerOnActionSuccess*
   ii. *EntityActionPArameterImpl<Report>* car l'action *DeleteReportDetailPanel* retourne une instance de cette classe.

6. Déclarer la ressource ***delete_report_confirm*** dans les fichiers nommés *dev__project__labels* :
   a. myexpenses/android/app/src/main/res/res/values/dev__project__labels.xml :

```
<string name="delete_report_confirm">Report has been deleted</string>
```

   b. myexpenses//android/app/src/main/res/res/values-fr/dev__project__labels.xml :

```
<string name="delete_report_confirm">Confirmation de la suppression de la note</string>
```

7. Compiler l'application
   a. Soit avec *Android Studio*
   b. Soit avec la commande

```
$ mdk platform-compile android
```

8. Déployer et exécuter l'application
   a. Sur le terminal, vérifier l'affichage du message à la suppression d'une note.

5.2. Calcul automatique du montant total

5.2.1. Présentation de la surcharge

Mettre à jour automatiquement le champ *amountTotal* (mis en évidence en Figure 11) du détail d'une Note, en calculant la somme des champs *amount* de ses objets *Expense* (situés dans la liste en dessous). Cette mise à jour doit se faire à chaque ajout, modification ou suppression d'un objet *Expense*.

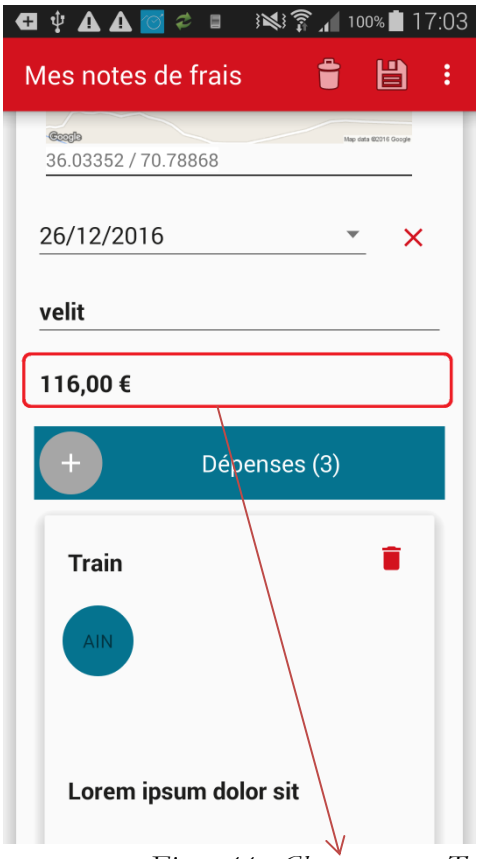

Figure 11 : Champ amountTotal.

5.2.2. Implémentation de la surcharge

En consultant le modèle UML de l'application, il est possible de constater que le montant total d'une note de frais est affiché dans le *panel* nommé *ReportDetailPanel*. L'implémentation de la règle métier doit ainsi être réalisée dans l'implémentation du *ViewModel* de ce *panel* : ***VM****ReportDetailPanel****Impl*** (***VM****ReportDetailPanel* définit le contrat du *ViewModel*) :

1. Ouvrir le fichier *VMReportDetailPanelImpl*.java présent dans le répertoire *myexpenses/android/app/src/main/java/com/soprasteria/mdk/handson/myexpenses/viewmodel/*
2. Rechercher le commentaire *//@non-generated-start[methods]*
   a. Il est à mettre en relation avec le commentaire *//@non-generated-end*
   b. Entre ces deux commentaires, il est possible d'ajouter du code spécifique au projet qui sera conservé lors d'une nouvelle génération (suite à une modification du modèle par exemple)
   c. Cette mécanique est présente dans l'ensemble du code généré.
3. Copier le code ci-dessous entre les deux commentaires :

```
@ListenerOnCollectionModified(fields = { KEY_LSTVMREPORTDETAILPANELEXPENSES })
public void onChangeExpense(ViewModel subVm, Action action, int itemId, ViewModel
newOrCurrentOrDeletedObject) {

	double total = 0d;
	for (int index = 0; index < this.lstVMReportDetailPanelExpenses.getCount(); index++)
{
		VMReportDetailPanelExpenses vm =
this.lstVMReportDetailPanelExpenses.getCacheVMByPosition(index);
		total += vm.getAmount();
	}
	setAmountTotal(total); // updates the view
}
```

   a. L'annotation *@ListenerOnCollectionModified* permet d'écouter les modifications (ajout, suppression, modification) apportées à un élément d'une liste contenue dans un formulaire (*FixedList*)
   b. ***LSTVMREPORTDETAILPANELEXPENSES*** permet d'identifier de manière unique au sein du formulaire la liste des dépenses d'une note de frais. Cette clé est générée
4. Compiler l'application
   a. Soit avec *Android Studio*
   b. Soit avec la commande

```
$ mdk platform-compile android
```

5. Déployer et exécuter l'application
6. Sur le terminal, vérifier que le champ amountTotal d'une note de frais se met automatiquement à jour si une dépense est ajoutée, modifiée ou supprimée.

5.3. Calcul automatique de l'état d'une dépense

5.3.1. Présentation de la surcharge

Mettre à jour automatiquement l'état d'une dépense, à chaque modification du montant et du type de la dépense, en comparant le montant de la dépense avec le montant maximal autorisé par son type.

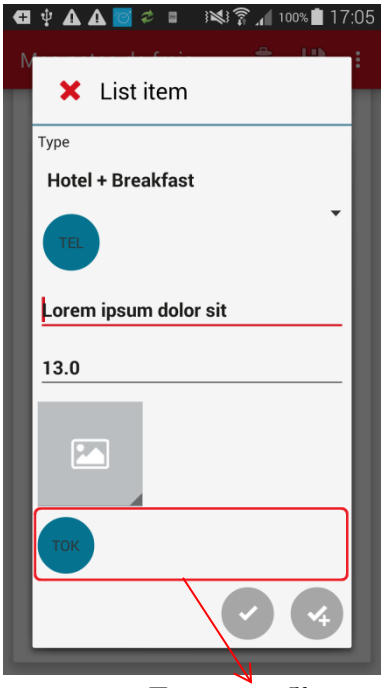

Figure 12 : Champ state.

5.3.2. Implémentation de la surcharge

En consultant le modèle UML de l'application, il est possible de constater que les dépenses d'une note de frais sont affichées dans le panel nommé *ReportDetailPanel*.

La surcharge précédente a montré qu'au panel *ReportDetailPanel* était associé un *ViewModel* ***VM****ReportDetailPanel*.

Ce *ViewModel* est une représentation des données à afficher dans le panel. On y trouve donc une représentation des dépenses sous forme d'une liste de sous-*ViewModel*. Chaque élément de cette liste est de Type ***VM****ReportDetailPanel****Expenses*** où ***Expenses*** est déduit du nom de l'extrémité de l'association entre *Report* et *Expense* présente dans le modèle UML.

L'écoute des modifications du montant d'une dépense doit se faire dans l'implémentation de ce sous-*ViewModel* :

1. Ouvrir le fichier *VMReportDetailPanelExpensesImpl*.java présent dans le répertoire *myexpenses/android/app/src/main/java/com/soprasteria/mdk/handson/myexpenses/viewmodel/*
2. Rechercher le commentaire *//@non-generated-start[methods]*
3. Copier le code ci-dessous à sa suite :

```
@ListenerOnFieldModified(fields = { KEY_VMREPORTDETAILPANELTYPE, KEY_AMOUNT })
public void onChangeAmountOrType(String field, Object oldValue, Object newValue) {
	// If the type and amount are valid
	if (oVMReportDetailPanelType != null && oVMReportDetailPanelType.getAmountMax()
!= null && this.amount > 0) {
		// Then we can compare the amount and modify the state
		if (this.amount > this.oVMReportDetailPanelType.getAmountMax()) {
			setState(ExpenseState.AMOUNTOVERMAX);
		} else {
			setState(ExpenseState.AMOUNTOK);
		}

	} else {
		setState(ExpenseState.AMOUNTNONE);
	}
}
```

   a. L'annotation *@ListenerOnFieldModified* permet d'écouter les apportées à un attribut du *ViewModel* (autre qu'une collection/liste)
   b. ***KEY_VMREPORTDETAILPANELTYPE*** (resp. ***KEY_AMOUNT***) permet d'identifier de manière unique au sein du formulaire le type (resp. le montant) de la dépense. Cette clé est générée.
4. Compiler l'application
   a. Soit avec *Android Studio*
   b. Soit avec la commande

```
$ mdk platform-compile android
```

5. Déployer et exécuter l'application
6. Sur le terminal, vérifier que l'état de la dépense est mis à jour à chaque modification du type et du montant, en adéquation avec le jeu de données.

5.4. Masquer l'état de la dépense en fonction du type

5.4.1. Présentation de la surcharge

L'objectif est d'afficher l'état de la dépense uniquement lorsqu'un montant maximal est défini sur le type de la dépense

5.4.2. Implémentation de la surcharge

7. Ouvrir le fichier *VMReportDetailPanelExpensesImpl*.java présent dans le répertoire *myexpenses/android/app/src/main/java/com/soprasteria/mdk/handson/myexpenses/viewmodel/*
8. Rechercher le commentaire *//@non-generated-start[methods]*
9. Copier le code ci-dessous à sa suite :

```
@BusinessRule(fields = KEY_STATE, propertyTarget = PropertyTarget.HIDDEN, triggers = {
KEY_VMREPORTDETAILPANELTYPE })
public boolean isStateHidden() {
      return this.oVMReportDetailPanelType == null ||
this.oVMReportDetailPanelType.getAmountMax() == null;
}
```

   a. L'annotation *@BusinessRule* permet d'appliquer une règle de calcul sur une propriété d'un champ de l'écran :
      i. Ici, la règle s'applique sur la visibilité (PropertyTarget.***HIDDEN***) de l'état (***KEY_STATE***), la règle est appelée à chaque fois que le type (***KEY_VMREPORTDETAILPANELTYPE***) est modifié
10. Compiler l'application
    a. Soit avec *Android Studio*
    b. Soit avec la commande

```
$ mdk platform-compile android
```

11. Déployer et exécuter l'application
12. Sur le terminal, vérifier que l'état de la dépense est masqué lorsque le type n'a pas montant maximal.

5.5. Rendu des montants : XX,XX €

5.5.1. Présentation de la surcharge

Dans le cadre de ce Hands-on, la devise de l'ensemble des montants de l'application est l'euro. L'objectif de la surcharge présentée ici est d'afficher tout montant de l'application, en lecture seule, avec deux décimales suivies du symbole €.

Afficher une valeur correspondant à un montant selon un formatage qui respecte la localisation du terminal, avec un séparateur, deux chiffres après le séparateur et le symbole « € ».

Ce formatage doit être effectué dans différents champs du panel de détail d'une note et du formulaire de saisie d'une dépense, mis en évidence en Figure 13, Figure 14 et Figure 15.

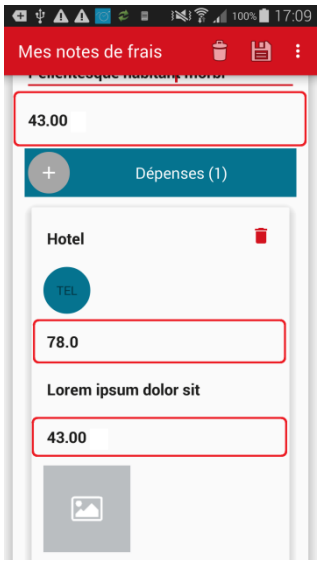

Figure 13 : Champs à formater dans le panel de détail d'une Note.

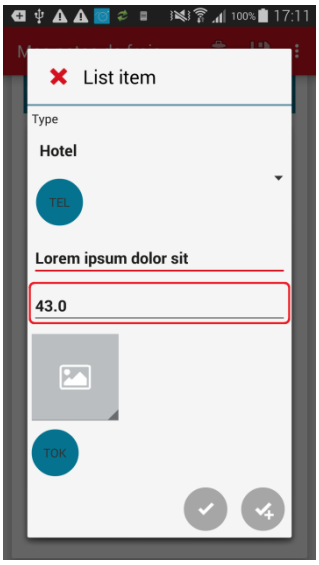

Figure 14 : Champ à formater dans le formulaire d'une Expense.

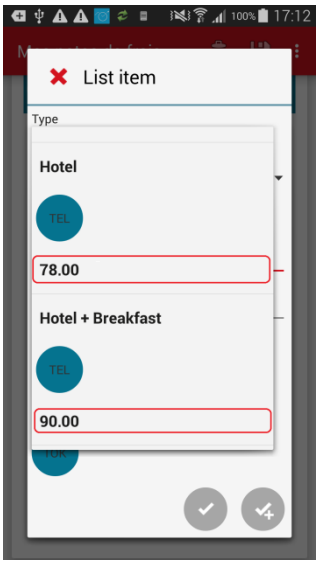

Figure 15 : Champ à formater dans le formulaire d'une Expense.

5.5.2. Implémentation de la surcharge

9. Copier le contenu du répertoire *myexpenses/hands-on/android/55 – Amount* dans le répertoire *myexpenses/android/app/src/main/java/com/soprasteria/mdk/handson/myexpenses/viewmodel/*
    a. Les éléments copiés sont les suivants
        i. AmountViewModel.java : Interface commune à l'ensemble des *ViewModels* affichant un montant
        ii. AbstractAmountViewModel.java : Classe abstraite des *ViewModels* affichant un montant
10. Ouvrir l'interface des trois *ViewModels* à modifier :
    a. VMReportDetailPanel
    b. VMReportDetailPanelExpenses
    c. VMReportDetailPanelType
11. Dans chacune de ces interfaces
    a. Repérer le commentaire *//@non-generated-start[class-signature]*
    b. Ajouter, en dessous du commentaire, le code suivant (ne pas oublier la virgule:

```
, AmountViewModel
```

12. Ouvrir l'implémentation des *ViewModels* considérés :
    a. VMReportDetailPanelImpl
    b. VMReportDetailPanelExpensesImpl
    c. VMReportDetailPanelTypeImpl
13. Dans chacune de ces classes :
    a. Repérer le commentaire *//@non-generated-start[class-signature-extends][X]*
    b. Supprimer le *[X]*

i. Cela indique au générateur que le code généré par défaut a été surchargé

c. Remplacer *AbstractItemViewModelId* par *AbstractAmountViewModel*

d. Repérer le commentaire *//@non-generated-start[methods]*

e. Ajouter à sa suite la méthode :

```
@ListenerOnFieldModified(fields = { KEY_AMOUNT })
public void onChangeAmount(String field, Object oldValue, Object newValue) {
    this.defineHumanReadableAmountFrom(this.amount);
}
```

i. En fonction du ViewModel :

1. ***KEY_AMOUNT*** devra être remplacé par ***KEY_AMOUNTTOTAL*** ou ***KEY_AMOUNTMAX***
2. **this.amount** par **this.amountTotal** ou **this.amountMax**

14. Ouvrir le layout affichant des montants
    a. *greportdetailpanel__screendetail__master*
    b. *greportdetailpanel__flistitemvmreportdetailpanelexpenses__master*
    c. *greportdetailpanel__spinitemvmreportdetailpaneltype__master*
15. Repérer les composants affichant des montants :
    a. **@+id/reportdetailpanel__amountTotal__value**
    b. **@+id/lstreportdetailpanel__amount__value**
    c. **@+id/lstreportdetailpanel__amountMax__value**
16. Remplacer **amount** / **amountTotal** / **amountMax** par **humanReadableAmount**
    a. Attention : Pour ne pas perdre les modifications apportées aux *layouts* il convient d'indiquer au générateur qu'il ne faut pas écraser les layouts à chaque génération via la commande suivante :

```
$ mdk platform-config-set android adJavaAppendGeneratorForceOverwrite false
```

17. Compiler l'application
    a. Soit avec *Android Studio*
    b. Soit avec la commande

```
$ mdk platform-compile android
```

18. Déployer et exécuter l'application
19. Sur le terminal, vérifier que les montants s'affichent comme attendus

5.6. Contenu de l'écran « À propos »

5.6.1. Présentation de la surcharge

L'objectif est d'afficher une page html embarquée dans le panel de l'écran *« À propos »*, illustré en Figure 17.

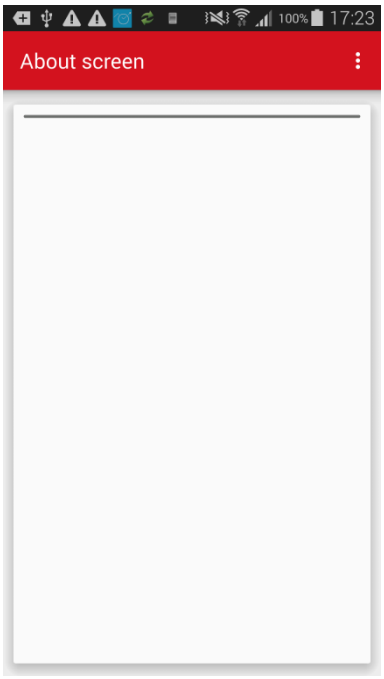

Figure 16 : AboutPanel sans page Web.

Figure 17 : AboutPanel avec la page Web « A propos ».

5.6.2. Implémentation de la surcharge

Ici nous ne nous intéressons pas à la conception de la page HTML que l'on suppose réalisé en amont. Nous abordons uniquement son intégration dans l'application

1. Copier
   a. Le contenu du répertoire *myexpenses/hands-on/android/56 – About*
   b. Dans *myexpenses/android/app/src/main/assets*
2. Ouvrir le layout de l'unique panel composant l'écran « A propos »
   a. *myexpenses/android/app/src/main/res/layout/gaboutpanel__screendetail__master.xml*
3. Définir l'attribut *movalys:url* du composant *MMWebView* de la façon suivante :
   a. **movalys:url="file:///android_asset/about.html"**
4. Compiler l'application
   a. Soit avec *Android Studio*
   b. Soit avec la commande

```
$ mdk platform-compile android
```

5. Déployer et exécuter l'application
6. Sur le terminal, vérifier la prise en compte de la surcharge.

6. Design de l'application

Une fois l'application générée avec le *MDK*, il est possible d'y ajouter tout type de fonctionnalité via des surcharges, mais également de personnaliser totalement son aspect visuel.

6.1. Personnalisation du rendu d'un composant

Les MDK Widgets ne permettent pas la surcharge voulue : libellé en bas et erreur en haut

6.2. Design général de l'application

Il s'agit ici d'appliquer un thème général à l'application. Il s'agit de personnalisation purement Android (aucune mécanique spécifique au MDK Android), cette section ainsi à la recopie de ressources permettant d'arriver au rendu souhaité.

1. Copier le contenu du répertoire *myexpenses/hands-on/android/62 - Design* dans le répertoire *myexpenses/android/app/src/main/* avec la commande (attention, la copie depuis le Finder ne fonctionne pas) depuis le répertoire du projet MDK :

```
$ cp -R "./hands-on/android/62 - Design/res" "./android/app/src/main/"
```

2. Compiler l'application
   a. Soit avec *Android Studio*
   b. Soit avec la commande

```
$ mdk platform-compile android
```

3. Déployer et exécuter l'application
   a. Sur le terminal, la prise en compte du design