Guide

Sopra Steria

HandsOn MDK Android

Hands-on MDK Android - My Expenses

Version 1.02 of Monday, February 22nd 2016

Historique

Version	Date	Reason	Edited by	Approved by
1.00	2015/12/08	Document overhaul	Antoine Belliard	Sébastien Maitre
1.01	2016/02/05	Corrections	Quentin Lagarde	Sébastien Maitre
1.02	2016/02/22	English translation	Maxime Lumeau	Sébastien Maître

Table of contents

1. Introduction

Throughout this "Hands-On MDK" training session, you will learn to use the "Mobile Development Kit" in order to generate, override end customize an application for the *Android* platform.

The app that will serve as example throughout this session will be named "My Expenses". As its name suggests, it will allow managing expense reports and associated expenses.

1.1. Overview of the application

My Expenses consists of four screens:

- **The main screen**, allowing navigating to the three other screens using navigation buttons.
- **The "my expenses" screen** that gathers:
  - A list displaying expense reports per customer, as shown left of Figure 1.
  - A partially editable detailed display of an expense report and its expenses, illustrated in the middle of Figure 1.
  - The completely editable detail of an expense, right of Figure 1.
- **The "about" screen** that contains a Web view presenting the app.
- **The "my travel books" screen.** This one is empty for the moment.

1.2. Roadmap

As said earlier, the app development will be broken down in three parts:

1. Designing and generating the application.
2. Overriding the business model and customizing the display.
3. Achieving client/server synchronization.

Only the first step is mandatory. The other two can be achieved in any order.

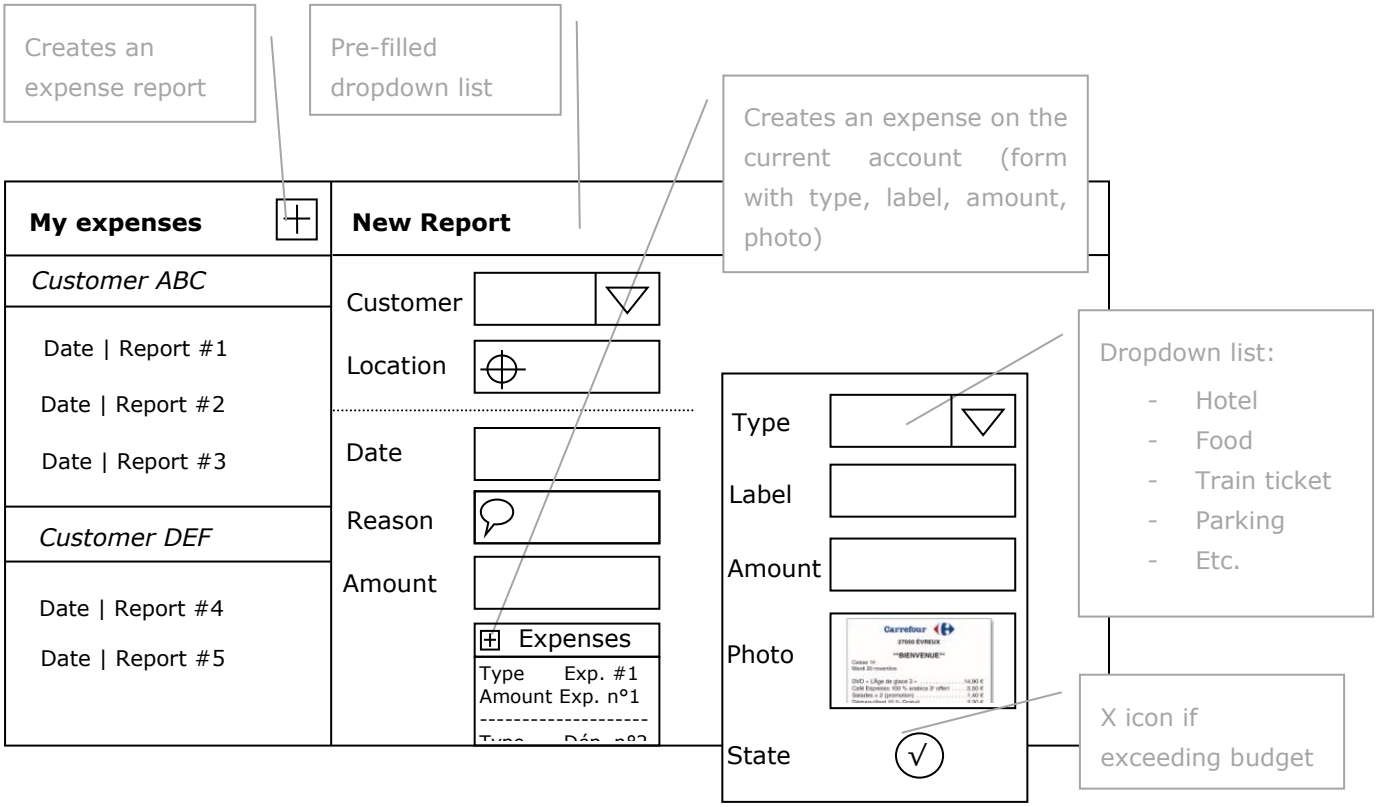

Figure 1 : the « My expenses » screen

1.3. Prerequisites

To follow this session you must:

- Install the *Command Line Interface* of MDK:
  - Install Node.js
  - Install mdk-cli:

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

- Install all the tools needed to develop an Android app with the MDK:

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

- Install Android Studio (**without the Android SDK**)
  - Then, launch Android Studio
  - File -> Settings
  - **Throughout the next steps *$HOME* is to be replaced by your own user directory (%USERPROFILE% on Windows, ~ on Mac/Linux)**
  - Locate the *Android SDK*:
    - Search for « Android SDK »
    - Replace the value of « Android SDK Location » with *$HOME/.mdk/tools/android-sdk*
  - Modify the *gradle* configuration :
    - Search for « gradle »
    - Replace the value of « Service directory path » with *$HOME/.mdk/tools/gradle*
- Install a UML modelling tool:
  - **MagicDraw** UML v17.0.5
    http://www.nomagic.com/products/magicdraw.html

For a better experience in testing the application we recommend you use the official Android emulator with a recent Intel x86 image and hardware acceleration enabled (http://developer.android.com/tools/devices/emulator.html#accel-vm).

You can also use your own physical Android device if you own one.

2. Initializing the project

Every MDK project uses a common source tree. Its creation is the first step.

In a command prompt:

1. Navigate to a folder that will contain the project root

> ⓘ *On Windows, it is recommended you use a root path with the fewest characters possible (e.g. **D:\dev**). Android SDK's generated paths can easily go over the system's limits.*

2. Initialize the project with the following command:

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

   a. *com.soprasteria.mdk.handson.myexpenses* is the unique id of the application
   b. The *–t* option is facultative. It allows using a template to generate the app. In that case, it will apply a visual theme and add a folder containing the overriding files.
   c. When it is asked, enter a login and a password.

3. Add the Android platform.

```
$ cd myexpenses

$ mdk platform-add android
```

   d. The « ***myexpenses*** » folder has been created. Its content must be as shown in Figure 2.

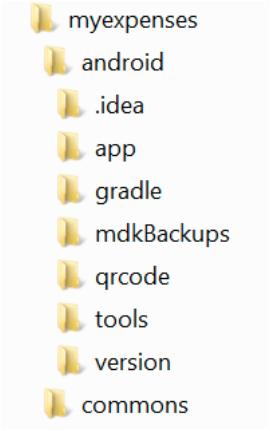

Figure 2 : « myexpenses » source tree.

3. Generating the application from a UML model

Once the project is initialized, it is necessary to design the model of the application using UML class diagrams. Then we can proceed to generating the app.

3.1. Generating the provided model

1. Overwrite *myexpenses/commons/modelisation.xml* with the file included in *myexpenses/hands-on/android/31 - Generation.*
2. Generate and build the app.

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

   a. The generated app will be found in *myexpenses/android/app/build/outputs/apk/*,
   b. It can be installed on a device/emulator

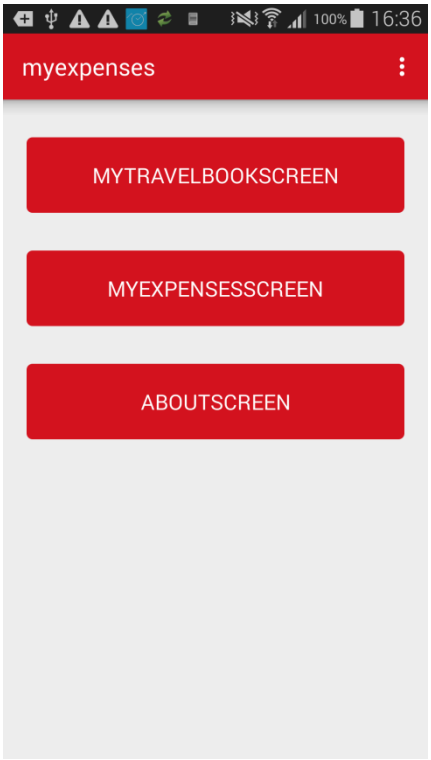

Figure 3 : Main screen of the generated app.

3. Click on *MY EXPENSES*. The following screen should be blank. We will see in section 3.3 how to add data to the generated app.

3.2. UML model

Open the app's UML model (*myexpenses/commons/modelisation.xml*) inside *MagicDraw*

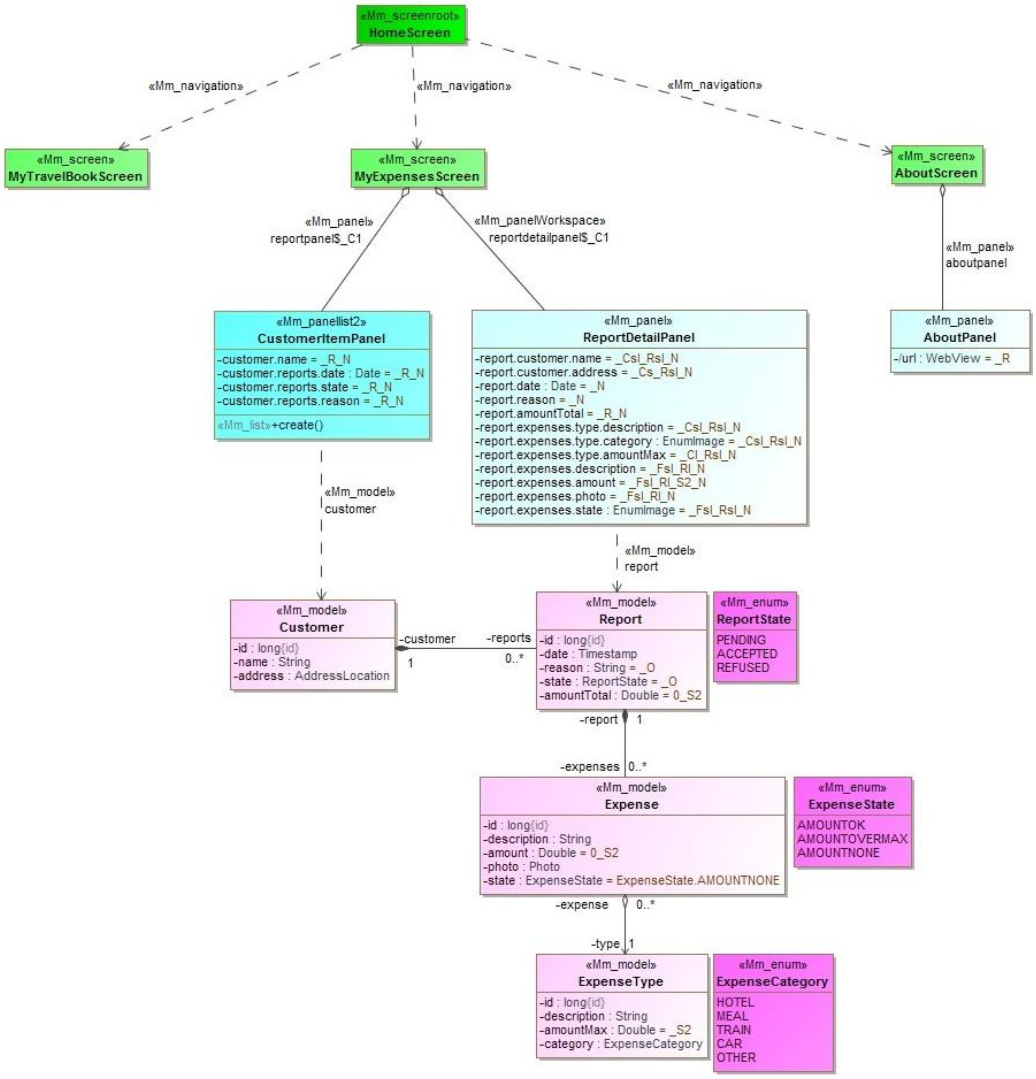

Figure 4 : Complete UML model.

The MDK modelling builds atop a classic UML modelling with a few additional mechanisms:

- The usage of UML stereotypes on classes and navigations.
- The usage of MDK types for attributes.
- Default values with the MDK syntax, to configure the attributes.

The application model is layered, from bottom to top:

1. The business model layer: containing the "**Mm_model**" stereotyped classes, as shown in Figure 5. It consists of:
    a. A ***Customer*** that brings ***Report***s.
    b. Each report breaks down in **Expense**s.
    c. Each expense has an ***ExpenseType***
    d. Some classes can have attributes with enumerated types:
        - ***NoteState, ExpenseState*** and ***ExpenseCategory*** are enumerations, flagged with « ***Mm_enum*** ».

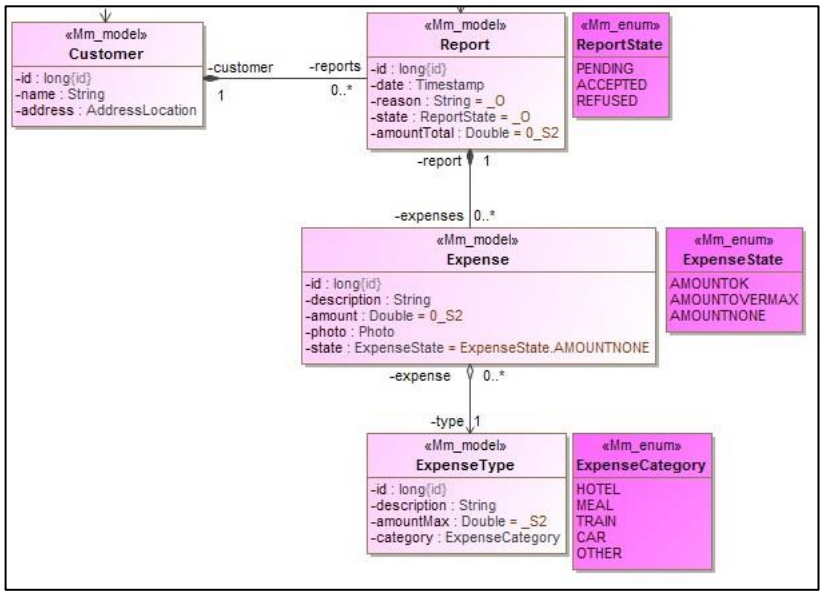

Figure 5: « Mm_model » stereotyped classes.

2. The panel layer, describing screen fragments: containing the "***Mm_panel***[…]" stereotyped classes, as shown in Figure 6. Three classes have been modelled here:
    a. **CustomerItemPanel** presents attributes of the *Report* and *Customer* classes.
    b. **ReportDetailPanel** presents the detail of an expense report, and allows displaying and/or updating some attributes of *Customer*, *Report*, *Expense* and *ExpenseType*.
    c. **AboutPanel** presents a Web view accessible from the app.

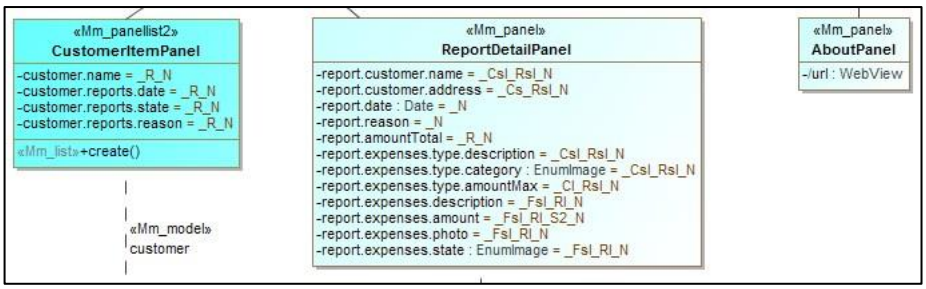

Figure 6 : « Mm_panel[…] » stereotyped classes.

3. The screen layer, describing screen contents and navigation: containing the
   "***Mm_screen***" stereotyped classes, along with one "***Mm_screenroot***", as shown in Figure 7.
   We find here all the "***Mm_screen***" stereotyped classes, each corresponding to one screen (and containing the "***Mm_panel***" above):
   a. **MyTravelBookScreen** is an empty screen, containing no panel.
   b. **MyExpensesScreen** has a list panel, *CustomerItemPanel*. Choosing a list item will bring us to the second panel, *ReportDetailPanel*
   c. **AboutPanel** is composed of a panel that displays a Web view.

   At last, we find here the "**Mm_screenroot**" classcorresponding to the main screen shown at the application startup. This screen is used to navigate to the other screens.

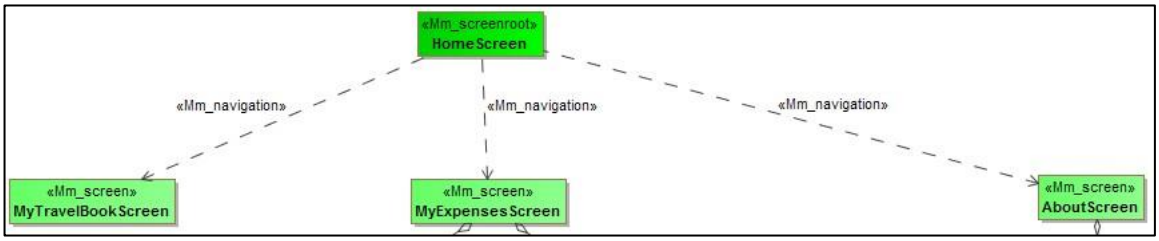

Figure 7 : « Mm_screen » and « Mm_screenroot » stereotyped classes.

3.3. Datasets

Using a dataset is a facultative step in the process of creating an application. It can be useful during the first development iterations if no web service can be consumed to populate the application data.

The MDK provides a tool that can initialize a dataset from an Excel worksheet: *myexpenses/commons/MOV-MM-Referentiel_de_donnees-V1.0.xls*.

With a few steps, it can build a file containing, for each business entity in the UML model, data that will be injected at first launch of the application.

It can also import and export data in the CSV format.

For this session an Excel worksheet has already been filled:

1. Open *myexpenses/hands-on/android/33 – Datas/MOV-MM-Referentiel_de_donnees-V1.11.xls*
2. Tab « Admin », click on « Export »
3. A file named *sqlitecreate_userdata* is generated
4. Replace *myexpenses/android/app/src/main/res/raw/sqlitecreate_userdata* with this file
5. Build

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

6. Uninstall the previous version of the app or wipe its data from your device.

7. Install and launch the apk file found in *myexpenses/android/app/build/outputs/apk/*
8. Click on *MY EXPENSES*
    a. The reports list is now populated, as shown in Figure 8.

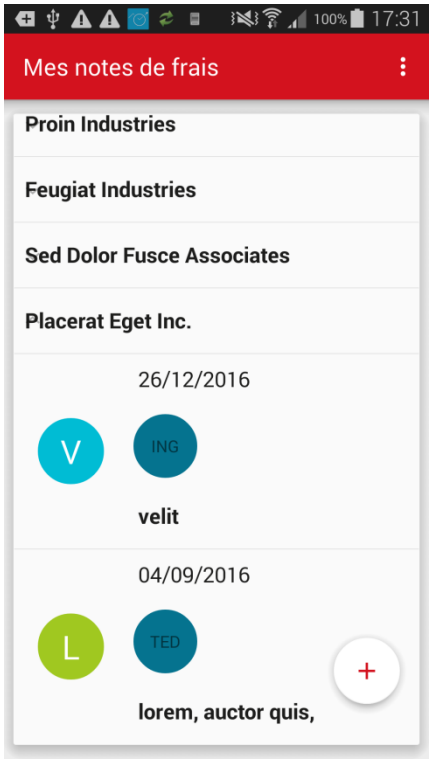

Figure 8 : Panel presenting the list of reports.

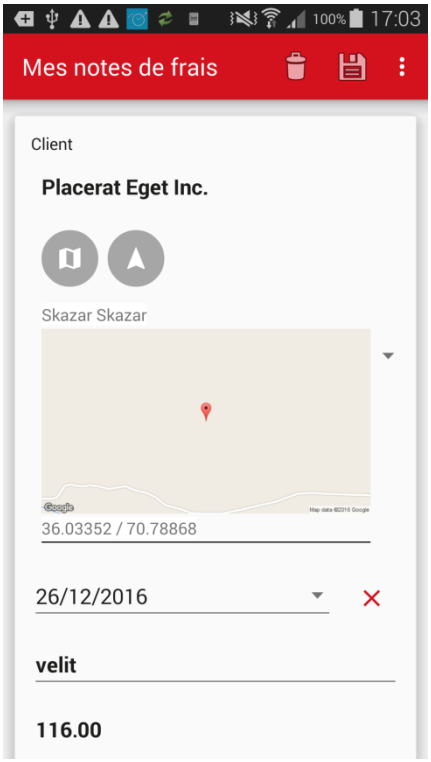

Figure 9 : Panel presenting the detail of a report.

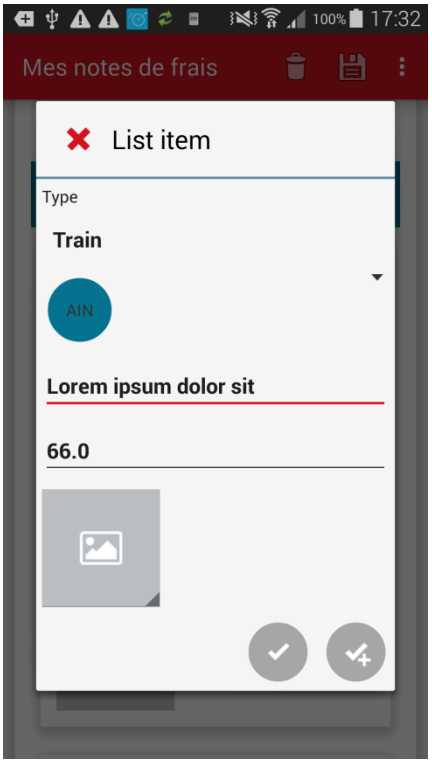

Figure 10 : Panel allowing the modification of an expense.

9. Click on a customer: the associated reports are displayed below the customer's name.
10. Click on a report to see the detail as shown in Figure 9.
11. Click on an expense below the report detail. The expense modification screen is displayed as shown in Figure 10.

4. Customizing

Once the app is generated, it is possible to customize some elements to finalize the project.

4.1. Customizing labels

The generated project contains a file named *dev__project_labels.xml* inside the directory *myexpenses/android/app/src/main/res/values*. The content of this file comes from the UML model:

- One label per screen
- One label per button
- One label for each attribute that doesn't have the default value option "_N" (No label)
- One label for each element of an enumeration
- …

Furthermore, it is possible to redefine these labels in multiple languages, in order to internationalize the application.

1. Copy the content of *myexpenses/hands-on/android/41 – Labels/* into *myexpenses/android/app/src/main/*
2. Build the app

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

3. Deploy and execute the app

4.2. Removing a label

The goal here is to demonstrate that modifying the UML model does not necessary cause a total loss of the custom labels.

1. Overwrite *myexpenses/commons/modelisation.xml* using the file in *myexpenses/hands-on/android/42 – Delete label.*

This model differs from the previous one by adding the option *_N* on the *url* field of *AboutPanel*

2. Generate and build

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

3. Deploy and execute
4. The label of the unique field in the *About* screen is gone
    a. In the file *myexpenses/android/app/src/main/res/values/dev__project__labels*
    b. When displaying the *About* screen.

5. Overriding

Once the application is generated, it is possible to override it to add any feature. In this chapter we will see how to override the business model of an application.

To achieve this, we will need to dive into the Java code. We recommend you import the project into Android Studio.

5.1. Removing an expense

By overriding the application we aim to add a new feature: removing an expense. The user must be notified when the action is complete.

First step: generating the action

1. Overwrite *myexpenses/commons/modelisation.xml* using *modelisation.xml* in *myexpenses/hands-on/android/51 – Delete report.*

   This model differs from the previous one *by adding deleteDetail* to *ReportDetailPanel*

2. Generate the application

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

Once the app is generated, you can see that the *DeleteReportDetailPanel* class that handles the action of removing the displayed expense.

Second step: Listening to the action completion and displaying the notification.

3. Open *ReportDetailPanel.java* inside

   myexpenses/android/app/src/main/java/com/soprasteria/mdk/handson/myexpenses/panel/

   *ReportDetailPanel.*java is a UML Panel in the form of an Android Fragment.

4. Look for the commented lines: *//@non-generated-start[methods]* and *//@non-generated-end*
5. Copy the code below between these lines (the code between the two markups won't be overwritten with a new generation):

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

@ListenerOnActionSuccess indicates that this method is called upon the success of the action defined in parameter.

Likewise, *@ListenerOnActionFail* is used as a failure callback.

6. Declare the ***delete_report_confirm*** resource inside *dev__project__labels*.

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

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

7. Build the app, either inside Android Studio or using the command line:

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

8. Deploy, execute and test.

5.2. Automatic calculation of the total amount

We want to automatically update the field *amountTotal* (shown in Figure 11) in the detail of a report, by calculating the sum of all *amount* fields of its expenses. This update must be done at every expense *adding, removal* or *modification*.

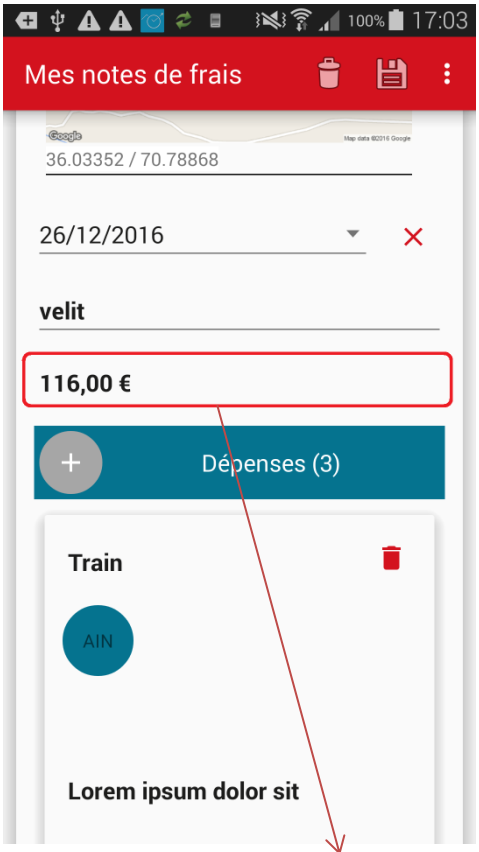

Figure 11 : amountTotal field.

A quick glance at the UML model reveals that the total amount of a report is displayed in *ReportDetailPanel*. The business rule must be implemented inside the *view model* of this *panel*: ***VM****ReportDetailPanel****Impl*** (***VM****ReportDetailPanel* defines the methods of the *view model*):

1. Open *VMReportDetailPanelImpl*.java inside *myexpenses/android/app/src/main/java/com/soprasteria/mdk/handson/myexpenses/viewmodel/*

2. Look for the commented lines: *//@non-generated-start[methods]* and *//@non-generated-end*

3. Copy the code below between these lines (the code between the two markups won't be overwritten with a new generation):

```
@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
}
```

@ListenerOnCollectionModified indicates that this method is called upon any action (adding, removing or updating) affecting a list. (*FixedList*)

LSTVMREPORTDETAILPANELEXPENSES helps us identifying the expense list inside a report. This key is generated.

4. Build the app, either within Android Studio or using the command line:

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

5. Deploy, execute and test.

5.3. Automatic calculation of the state of an expense

We want to automatically update the state of an expense, when modifying the type or the amount, by comparing the current amount with the maximum allowed for this type of expense.

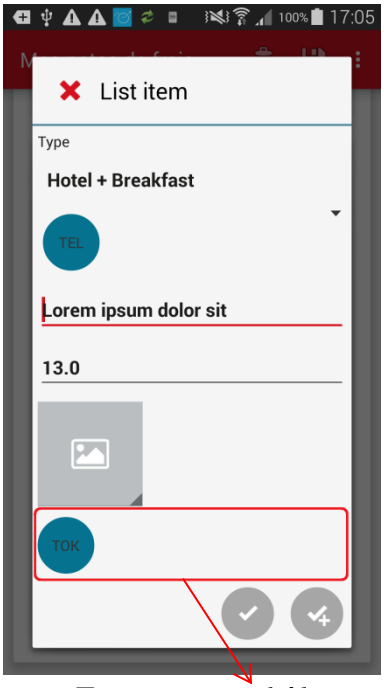

Figure 12 : state field.

A quick glance at the UML model reveals that the expenses of a report are displayed in *ReportDetailPanel*.

The previous override showed that *ReportDetailPanel* was associated to the *view model* ***VM****ReportDetailPanel*.

This view model represents the data displayed in the *Panel*. Each expense item will be associated with a ***VM****ReportDetailPanel****Expenses*** (*Expenses* is the name of the association end in the UML model)

The expense amount modification listener will be implemented through this *view model*:

1. Open *VMReportDetailPanelExpensesImpl*.java in *myexpenses/android/app/src/main/java/com/soprasteria/mdk/handson/myexpenses/viewmodel/*
2. Look for the commented lines: *//@non-generated-start[methods]* and *//@non-generated-end*

3. Copy the code below between these lines:

```
@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);
      }
}
```

@ListenerOnFieldModified indicates that the method is called upon changing the value of any attribute of the *ViewModel* (other than a list)

KEY_VMREPORTDETAILPANELTYPE (respectively, ***KEY_AMOUNT***) helps us identifying the type (respectively, the amount) of the expense. These keys are generated.

4. Build the app, either inside Android Studio or using the command line:

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

5. Deploy, execute and test

5.4. Hide the state of an expense based on its type

The aim here is to display the state of the expense only if a maximum amount has been defined on the expense type.

1. Open *VMReportDetailPanelExpensesImpl*.java in *myexpenses/android/app/src/main/java/com/soprasteria/mdk/handson/myexpenses/viewmodel/*
2. Look for the commented lines: *//@non-generated-start[methods]* and *//@non-generated-end*
3. Copy the code below between these lines:

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

@BusinessRule is used to apply a calculation on a field property:

Here we target the visibility (PropertyTarget.***HIDDEN***) of the state field (***KEY_STATE***), the rule being applied each time (***KEY_VMREPORTDETAILPANELTYPE***) is modified

4. Build the app, either inside Android Studio or using the command line:

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

5. Deploy, execute and test

5.5. Amount formatting: XX,XX €

Throughout this session, the Euro currency will be used for all amounts. Our aim here is to format all read-only amounts with two decimals and the € symbol, while respecting the locale of the device.

This formatting must be implemented for all fields in the report detail panel and the expense entry panel as shown in Figure 13, Figure 14 and Figure 15.

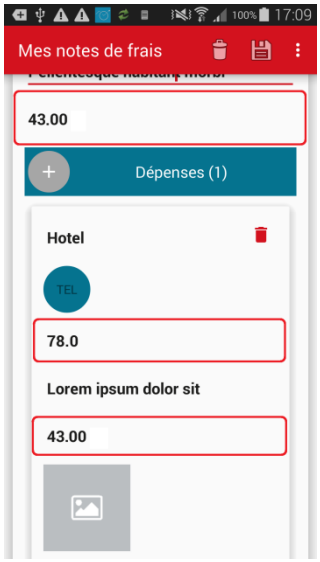

Figure 13 : field to format in the report detail panel

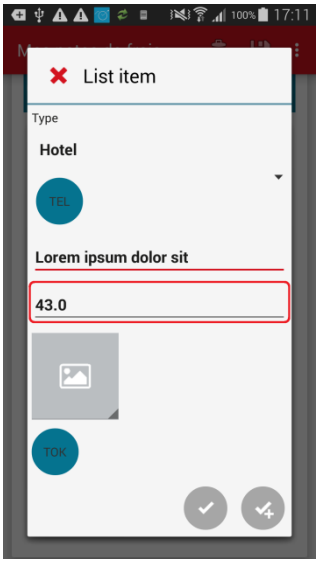

Figure 14 : field to format in the expense entry panel.

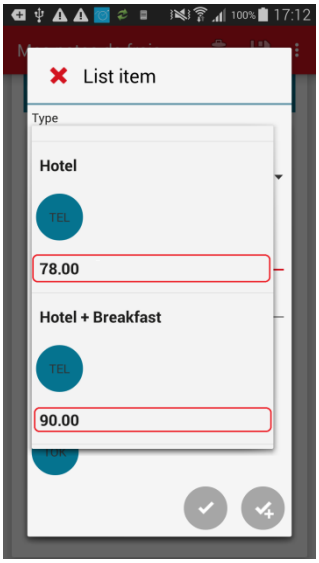

Figure 15 : field to format in the expense type dropdown list.

1. Copy the content of *myexpenses/hands-on/android/55 – Amount* inside *myexpenses/android/app/src/main/java/com/soprasteria/mdk/handson/myexpenses/viewmodel/*
2. Open the interface of the three *ViewModels* to modify:
   - *VMReportDetailPanel*
   - *VMReportDetailPanelExpenses*
   - *VMReportDetailPanelType*
3. For each interface
   a. Look for the commented lines: *//@non-generated-start[class-signature]* and *//@non-generated-end*
   b. Copy the code below (including the comma) between these lines:

```
, AmountViewModel
```

4. Open the implementation of the *ViewModels*:
   - *VMReportDetailPanelImpl*
   - *VMReportDetailPanelExpensesImpl*
   - *VMReportDetailPanelTypeImpl*
5. For each class:
   a. Look for this commented line *//@non-generated-start[class-signature-extends][X]*
   b. Remove the *[X].* It indicates that the generated code has been overloaded

c. Replace *AbstractItemViewModelId* with *AbstractAmountViewModel*
d. Look for the commented lines: *//@non-generated-start[methods]* and *//@non-generated-end*
e. Copy the code below between these lines:

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

For each ViewModel:

- ***KEY_AMOUNT*** must be replaced with ***KEY_AMOUNTTOTAL*** or ***KEY_AMOUNTMAX***
- **this.amount** with **this.amountTotal** or **this.amountMax**

6. Open the layout files displaying the amounts
   - *greportdetailpanel__screendetail__master*
   - *greportdetailpanel__flistitemvmreportdetailpanelexpenses__master*
   - *greportdetailpanel__spinitemvmreportdetailpaneltype__master*
7. Look for the elements displaying amounts:
   - **@+id/reportdetailpanel__amountTotal__value**
   - **@+id/lstreportdetailpanel__amount__value**
   - **@+id/lstreportdetailpanel__amountMax__value**
8. Replace **amount** / **amountTotal** / **amountMax** with **humanReadableAmount**

WARNING - In order to keep the layout modifications, we must tell the generator not to overwrite the layout files:

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

6. Build the app, either inside Android Studio or using the command line:

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

9. Deploy, execute and test

5.6. « About » screen

The aim here is to display an HTML page embedded in a Web view in the "About" screen, as shown in Figure 17.

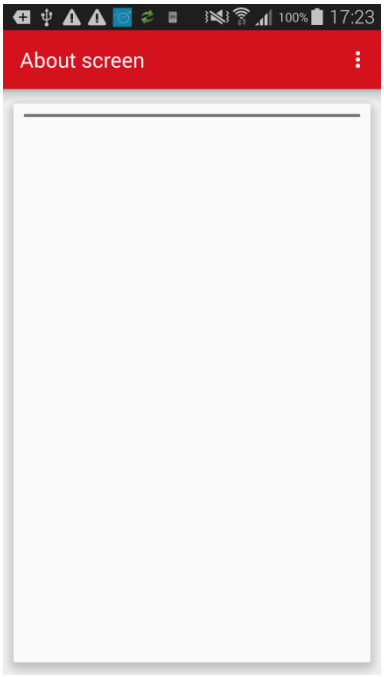

Figure 16 : AboutPanel without Web view.

Figure 17 : AboutPanel showing the « About » web page.

We won't be detailing the making of the HTML page.

1. Copy the content of *myexpenses/hands-on/android/56 – About* into *myexpenses/android/app/src/main/assets*
2. Open the layout file of the "About" *Panel*. *myexpenses/android/app/src/main/res/layout/gaboutpanel__screendetail__master.xml*

Define the attribute *movalys:url* of the *MMWebView* item this way:

```
movalys:url="file:///android_asset/about.html"
```

3. Build the app, either inside Android Studio or using the command line:

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

4. Deploy, execute and test.

6. Theming

Once the application is generated, it is possible to customize its visual aspect entirely.

6.1. Customizing visual components

See the mdk-android-widget library documentation.

6.2. Customizing the overall app theme

No particular MDK-specific mechanism is necessary here. We only need to copy the Android theme resources into our project to reach the expected results.

1. Copy the content of *myexpenses/hands-on/android/62 - Design* into *myexpenses/android/app/src/main/*

   Caution: on a Mac, copying from the Finder can cause issues. Use this command instead:

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

2. Build the app, either inside Android Studio or using the command line:

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

3. Deploy, execute and test.