Sopra Steria

Hands On MDK iOS

Version 1.00 du mercredi 17 février 2016

État : Travail

Destinataire(s)

Historique

Version	Date	Origine de la mise à jour	Rédigée par	Validée par
1.00	17/02/2016	Refonte HandsOn iOS	Quentin Lagarde	Sébastien Maitre

Sommaire

1. Introduction 5
1.1. Présentation de l'application 5
1.2. Plan de la réalisation 6
1.3. Prérequis 7
2. Création du projet 8
3. Génération de l'application à l'aide du modèle UML 9
3.1. Génération à partir du modèle fourni 9
Analyse du modèle UML utilisé 10
3.2. 10
3.3. Jeu de données 13
4. Personnalisation de l'application 15
4.1. Personnalisation des libellés 15
4.2. Suppression d'un libellé 16
5. Surcharge métier de l'application 17
5.1. Calcul automatique du montant total 17
5.1.1. Présentation de la surcharge 17
5.1.2. Implémentation de la surcharge 18
5.2. Calcul automatique de l'état d'une dépense 20
5.2.1. Présentation de la surcharge 20
5.2.2. Implémentation de la surcharge 21
5.3. Masquer l'état de la dépense en fonction du type 23
5.3.1. Présentation de la surcharge 23
5.3.2. Implémentation de la surcharge 23
5.4. Rendu des montants : XX,XX € 25
5.4.1. Présentation de la surcharge 25
5.4.2. Implémentation de la surcharge 26
5.5. Contenu de l'écran « À propos » 28
5.5.1. Présentation de la surcharge 28
5.5.2. Implémentation de la surcharge 28
6. Design de l'application 30
6.1. Design général de l'application 30

1. Introduction

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

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.

L'application sera également en mesure de se synchroniser avec un serveur, réalisé à cet effet avec une application *JEE*. Grace au *MDK*, cette synchronisation sera incrémentielle et bidirectionnelle sur les classes de notre choix.

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.

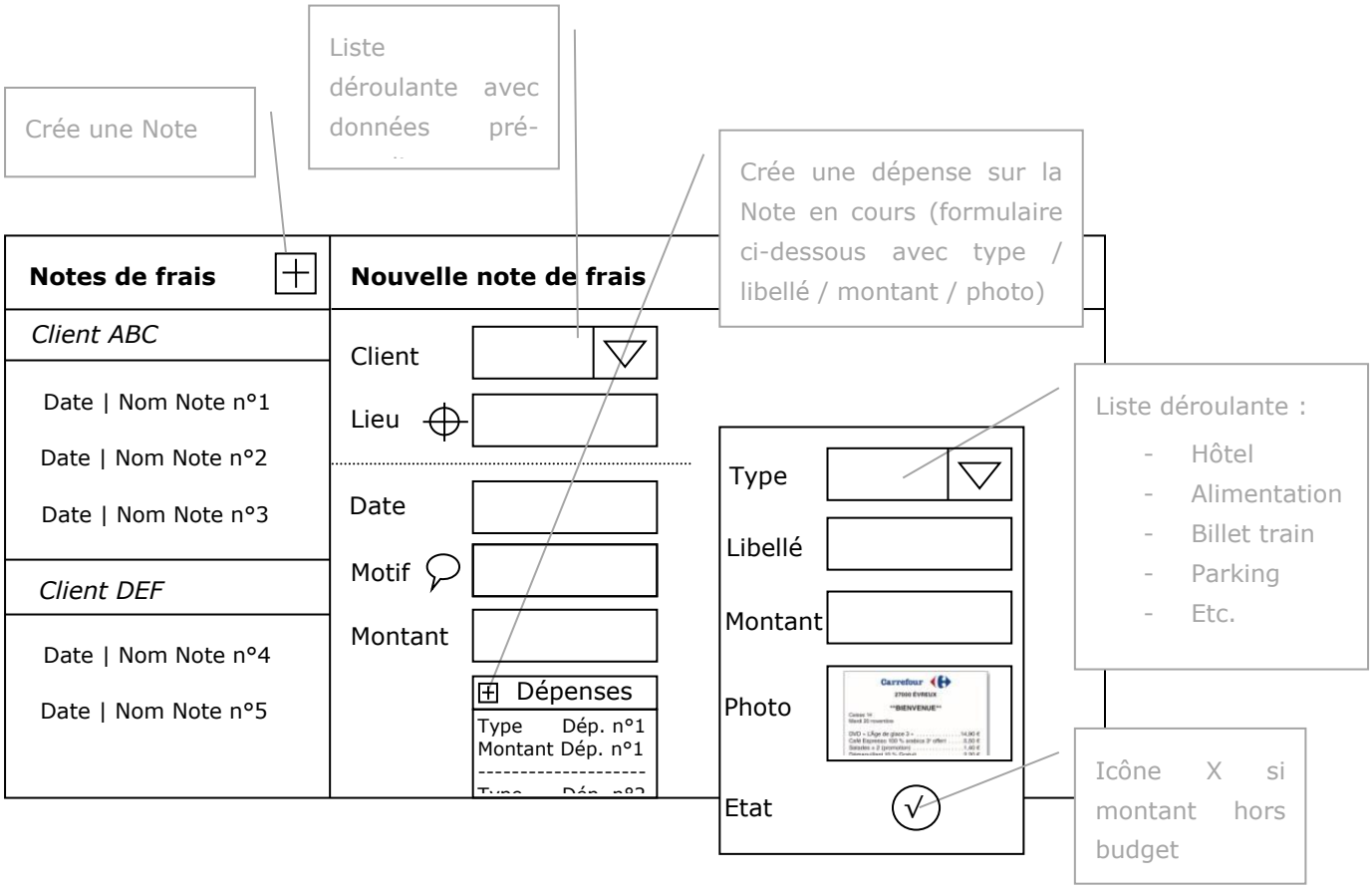

Figure 1 : Principe de l'écran secondaire « Notes de frais ».

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 iOS :

```
$ mdk tools-install ios
```

- Installer XCode (la dernère version, non-bêta)
- 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 le simulateur *iPhone5/5S* afin d'obtenir un résolution correcte pour votre écran(les simulateurs pour iPhone 6, 6 Plus ont des résolutions trop hautes pour les écrans de formation).

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
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 iOS

```
$ cd myexpenses

$ mdk platform-add ios
```

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

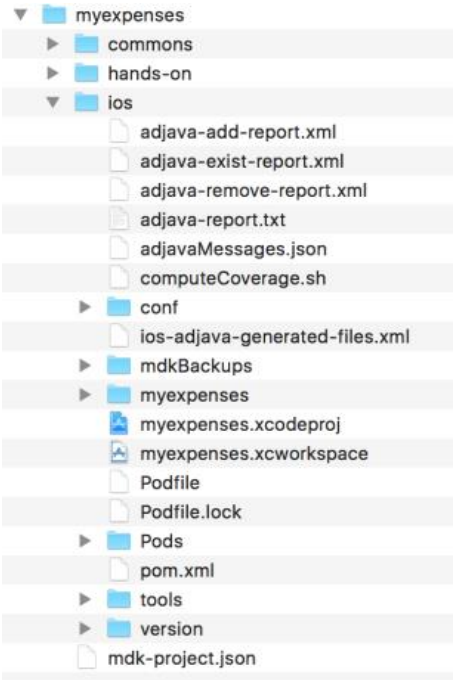

Figure 2 : Arborescence du dossier « myexpenses ».

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/ios/****31 - Generation.*
2. Générer et compiler l'application

```
$ mdk platform-build ios
```

a. L'application ainsi construite se trouve dans le répertoire ***Erreur ! Nom de propriété de document inconnu.****Debug-iphonesimulator/myexpenses.app*,
b. elle peut être installée sur un terminal/émulateur via la commande (depuis le répertoire du projet):

```
$ xcrun simctl install booted myexpenses/ios/build/Build/Products/Debug-iphonesimulator/myexpenses.app
```

c. l'application est alors installée sur l'écran d'accueil. Lancez-là pour obtenir l'écran d'accueil suivant :

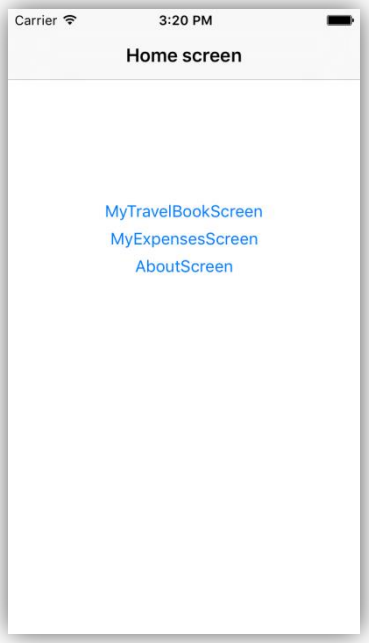

Figure 3 : Ecran principal de l'application générée à partir du modèle UML complet.

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*

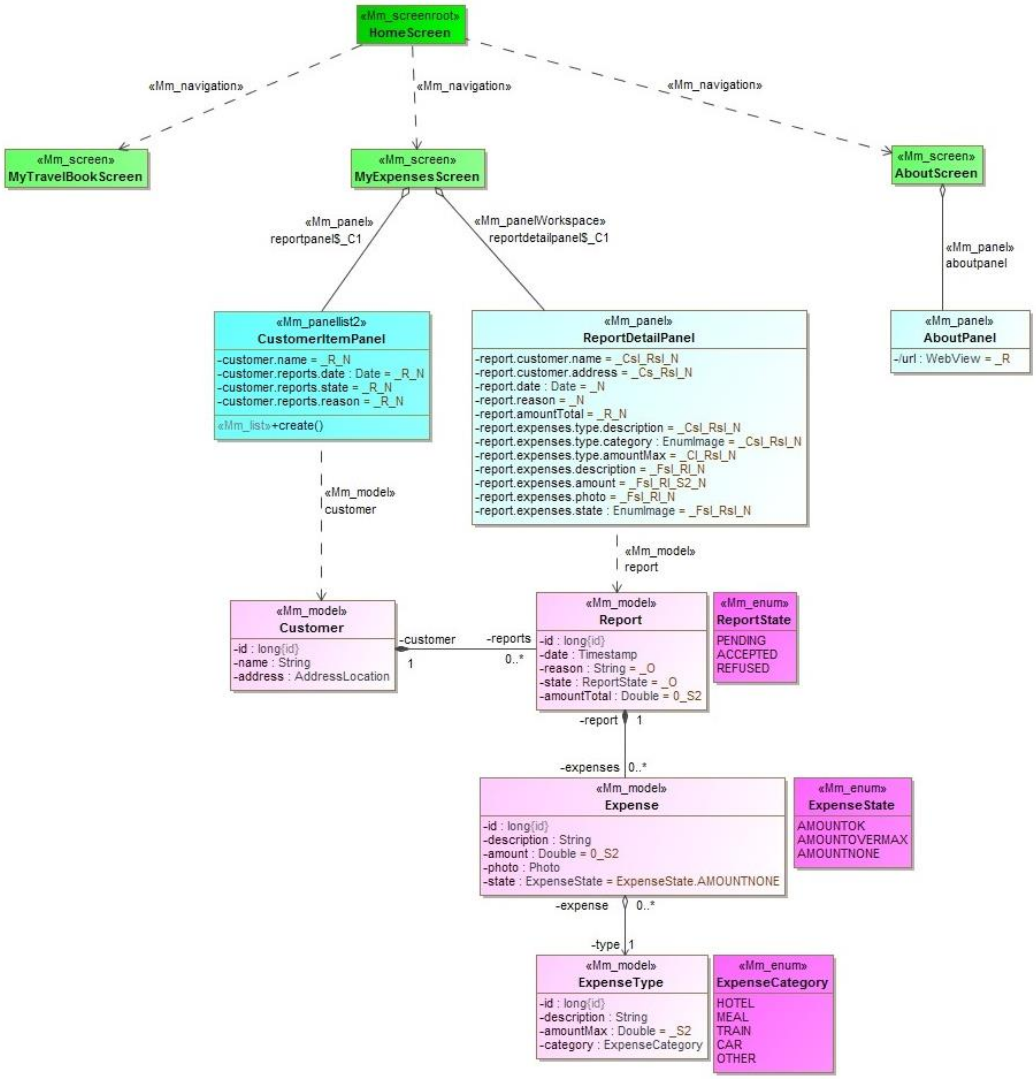

Figure 4 : Modèle UML complet.

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

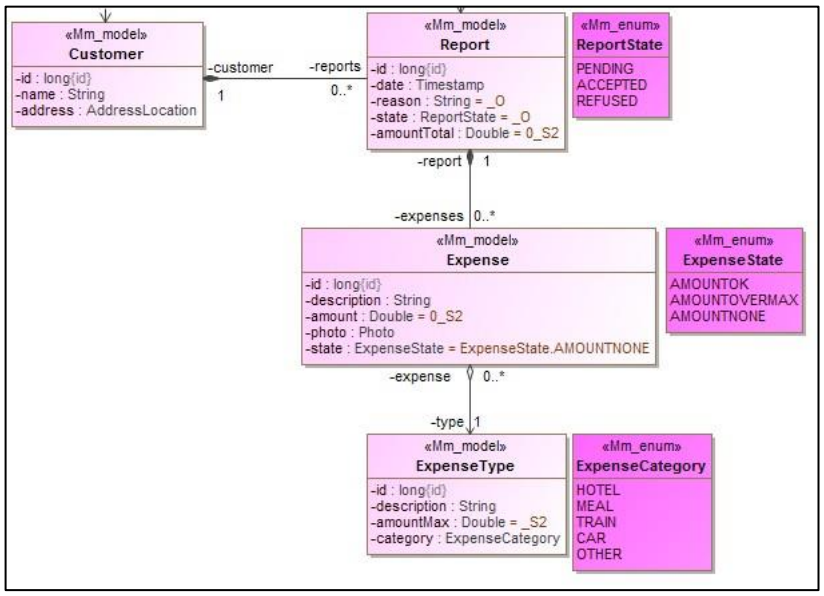

Figure 5: Classes stéréotypées « Mm_model ».

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.

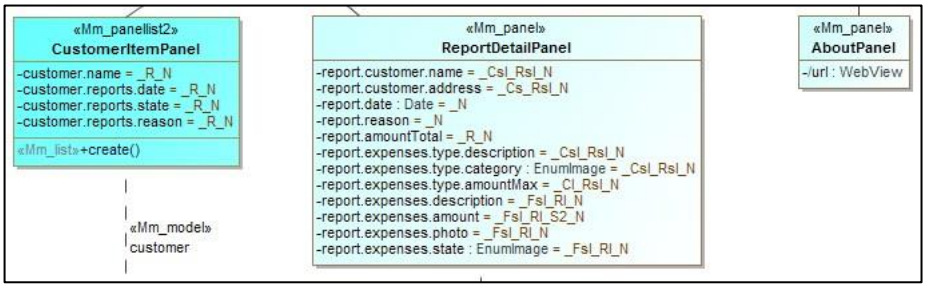

Figure 6 : Classes stéréotypées en « Mm_panel[…] ».

1. « 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.

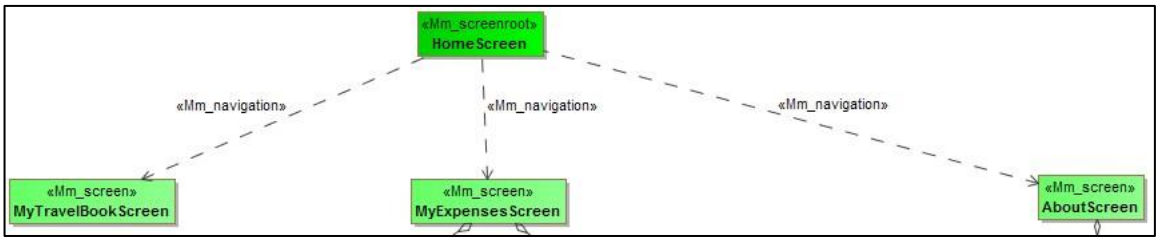

Figure 7 : Classes stéréotypées en « Mm_screen » et « Mm_screenroot ».

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* :

Commons/MOV-MM-Referentiel_de_donnees-V1.XX.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 :

1. Ouvrir ***myexpenses/hands-on/ios/****33 – Datas :* Ce dossier comporte les fichiers CSV déjà générés par la feuille Excel de référentiel de données.
2. Copier tous les fichiers CSV dans le dossier *myexpenses/ios/myexpenses/resources/csvdatas/*
3. Ouvrir le projet XCode en double cliquant sur *myexpenses.xcworkspace* et déplier l'arborescence : *myexpenses/myexpenses/resources/csvdatas/*
4. Il faut référencer les fichiers CSV dans le projet XCode : faire un clic-droit sur le dossier « csvdatas » puis « Add files to "myexpenses" » et ajouter les nouveaux fichiers CSV.
5. Désinstaller la précédente version de l'application sur votre terminal/émulateur ou faire effectuer une réinitialisation complète du simulateur avec l'option « *Simulator > Reset Contents and Settings…* »
6. Lancer la compilation et l'exécution de l'application sur le simulateur depuis XCode
7. Cliquer sur le bouton *MyExpensesScreen*
    d. La liste des notes de frais est maintenant alimentée. Son affichage doit être similaire à celui illustré en Figure 8

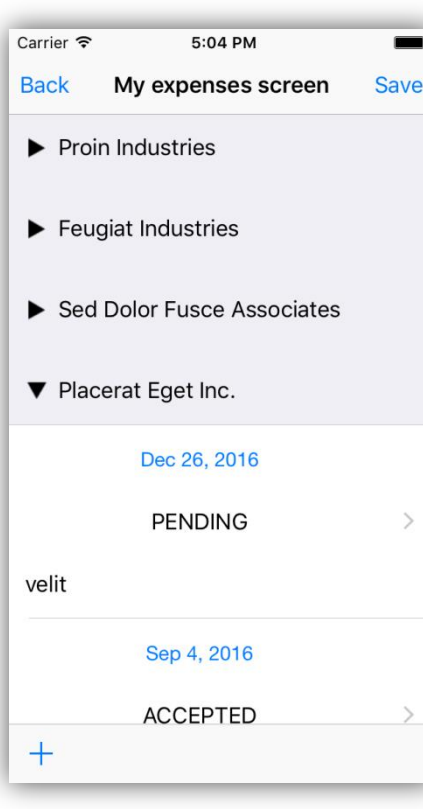

Figure 8 : Panel présentant le jeu de données (Clients et Notes de frais).

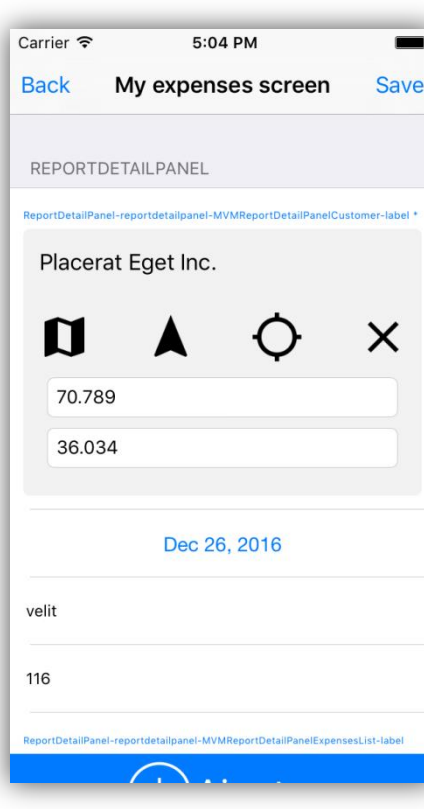

Figure 9 : Panel présentant le détail d'une Note de frais.

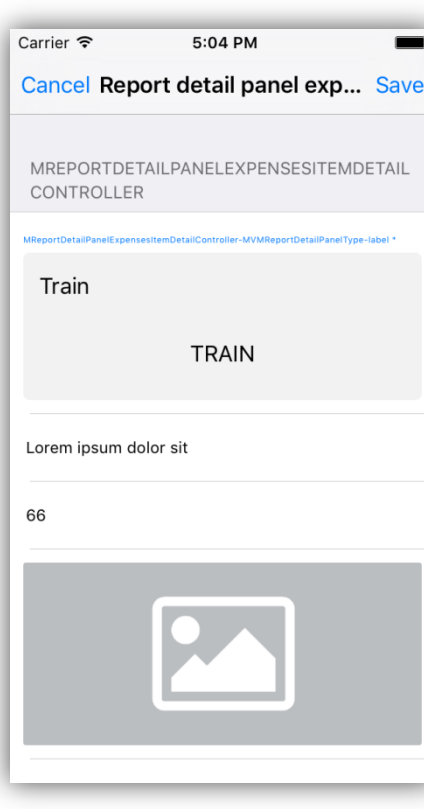

Figure 10 : Panel de modification d'une Expense.

8. Cliquer sur un Client de la liste/ Les Notes de frais associées au Client apparaissent en dessous.
9. Cliquer sur une Note de frais pour en afficher le détail. Son affichage doit être similaire à celui illustré en Figure 9.
10. 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 des fichiers localisés nommés Localizable-project.*strings* dans le répertoire *myexpenses/ios/myexpenses/resources/strings/XX.lproj*. où « XX » représente le code la langue dans le format international (*fr* pour Français (France) et *en* pour Anglais (Royaume-Uni))

- 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
- …

1. Copier
   a. Le contenu du répertoire ***myexpenses/hands-on/ios/****41 – Labels/*
   b. Dans ***myexpenses/ios/resources/strings/*** *(choisir l'option « Remplacer », et non pas « Fusionner »)*
2. Dans XCode, nettoyer le projet avec les raccourcis Cmd+Maj+K (Clean project) et Cmd+Alt+Maj+K (Clean Build folder). En effet, la modification de ressources hors du projet XCode n'étant pas prise en compte par XCode, il est nécessaire de nettoyer le cache de compilation du projet.
3. Lancer l'application à nouveau depuis XCode (Cmd+R) et observez la modification des libellés.

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/ios/****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 l'application avec la commande suivante depuis le répertoire du projet MDK

```
$ mdk platform-gensrc ios
```

3. Recompiler et exécuter l'application depuis XCode (Cmd+R)
4. Vérifier que le libellé de l'unique champ de l'écran « À propos » a disparu
   a. Lors de l'affichage de l'écran « À propos »

5. Surcharge métier de l'application

5.1. Calcul automatique du montant total

5.1.1. 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*.

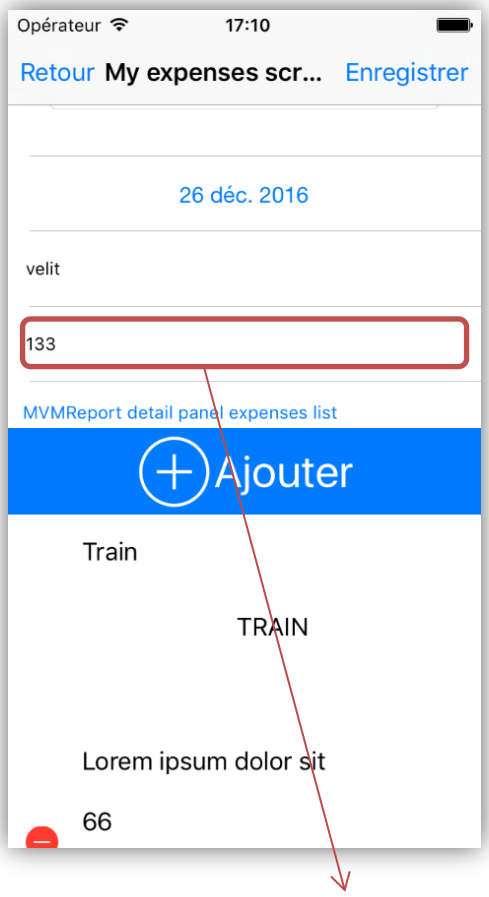

Figure 11 : Champ amountTotal.

5.1.2. 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* : ***MVM****ReportDetailPanel* (***MVM****ReportDetailPanel* définit le contrat du *ViewModel*) :

1. Ouvrir le fichier M*VMReportDetailPanel*.m présent dans le répertoire ***myexpenses/ios/viewmodel/***
2. Rechercher le commentaire *//@non-generated-start[other-methods][X]*
   a. Supprimer le suffixe *[X]* afin que le code surchargé ne soit pas écrasé à la prochaine génération du projet.
   b. Il est à mettre en relation avec le commentaire *//@non-generated-end*
   c. 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)
   d. Cette mécanique est présente dans l'ensemble du code généré.
3. Copier le code ci-dessous entre les deux commentaires :

```
-(void) createViewModelConfiguration {
    MFViewModelConfiguration *config = [MFViewModelConfiguration configurationForViewModel:self];

    MFListenerDescriptor *propertyChangedListener = [MFListenerDescriptor
listenerDescriptorForType:MFListenerEventTypeViewModelPropertyChanged withFormat:
                                                    @"computeTotalAmount :
mVMReportDetailPanelExpensesList",
                                                    nil];
    [config addListenerDescriptor:propertyChangedListener];
}
-(void)computeTotalAmount {
    float total = 0.0;
    for(MVMReportDetailPanelExpensesItemCell *itemVM in
self.mVMReportDetailPanelExpensesList.viewModels) {
        total += [itemVM.amount floatValue];
    }
    self.amountTotal = @(total);

}
```

   a. Une configuration de ViewModel est créée pour ce ViewModel
   b. Un descripteur de listener est créé pour écouter les changements de valeur du champ « *mVMReportDetailPanelExpensesList* » ; la méthode associée « *computeTotalAmount* » sera appelée en conséquence
   c. Le descripteur de listener est ajouté à la configuration du ViewModel
   d. L'implémentation de la méthode « *computeTotalAmount* » recupère la liste des *Expenses* pour faire la somme de leur montant (*amount)*. Le montant total est ainsi mis à jour.
   e. Le binding bi-directionnel a pour effet de mettre à jour visuellement instantanément le champ représentant le montant total de la note de frais
4. Compiler l'application
   a. Soit avec XCode
   b. Soit avec la commande

```
$ mdk platform-compile ios
```

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.

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

5.2.1. 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.

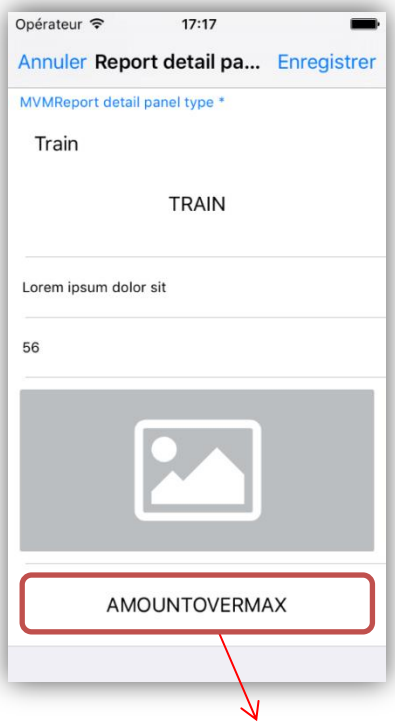

Figure 12 : Champ state.

5.2.2. 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* ***MVM****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* (***MVM****ReportDetailPanel****ExpensesList***). Chaque élément de cette liste est de Type ***MVM****ReportDetailPanel****ExpensesItemCell*** où ***Expenses*** est déduit du nom de l'extrémité de l'association entre *Report* et *Expense* présente dans le modèle UML. ***ItemCell*** indique le type du ViewModel associé : il s'agit là d'un *ViewModel* d'une cellule représentant un item de liste.

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

1. Ouvrir le fichier *MVMReportDetailPanelExpensesItemCell.m* présent dans le répertoire ***myexpenses/ios/viewmodel/***
2. Rechercher le commentaire *//@non-generated-start[other-methods][X]* et enlever l'indicateur *[X]* pour que le code contenu entre cette balise et la balise *//@non-generated-end* ne soit pas écrasé lors de la prochaine génération
3. Copier le code ci-dessous à sa suite :

```
-(void) createViewModelConfiguration {
        MFViewModelConfiguration *config = [MFViewModelConfiguration configurationForViewModel:self];

        MFListenerDescriptor *propertyChangedListener = [MFListenerDescriptor
listenerDescriptorForType:MFListenerEventTypeViewModelPropertyChanged withFormat:
                                    @"updateExpenseState : selectedMVMReportDetailPanelTypeItem,
amount" ,
                                    nil];
   [config addListenerDescriptor:propertyChangedListener];
}

-(void) updateExpenseState {
   if(self.selectedMVMReportDetailPanelTypeItem && self.selectedMVMReportDetailPanelTypeItem.amountMax
&& [self.selectedMVMReportDetailPanelTypeItem.amountMax intValue] >0) {
      if([self.amount intValue] > [self.selectedMVMReportDetailPanelTypeItem.amountMax intValue] ) {
         self.state = MEXPENSESTATE_AMOUNTOVERMAX;
      }
      else {
         self.state = MEXPENSESTATE_AMOUNTOK;
      }
   }
   else {
      self.state = MEXPENSESTATE_AMOUNTNONE;
   }
}
```

   a. La première méthode « *createViewModelConfiguration* » permet de créer une configuration de ViewModel : c'est un objet spécifique Movalys MDK iOS permettant de configurer ce ViewModel, notamment en ajoutant des écouteurs de champs.

b. La configuration est créée pour ce ViewModel. On y ajoute ensuite un écouteur (*MFListenerDescriptor)* qui associe l'appel de la méthode « *updateExpenseState* » aux changement de valeur des champs « *selectedMVMReportDetailPanelTypeItem* » et « amount ».
c. Il convient alors d''implémenter la méthode « *updateExpenseState* », qui en fonction du type et du montant de la note, va mettre à jour son état (OK, OVERMAX ou NONE).

4. Compiler l'application
   a. Soit avec XCode
   b. Soit avec la commande

```
$ mdk platform-compile ios
```

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.

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

5.3.1. 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

5.3.2. Implémentation de la surcharge

7. Ouvrir le fichier *MVMReportDetailPanelExpensesItemCell*.m présent dans le répertoire ***myexpenses/ios/viewmodel/***
8. Rechercher le commentaire *//@non-generated-start[methods]*
9. Copier le code ci-dessous à la suite du code déjà ajouté :

```
-(void) updateExpenseStateVisibility {
   BOOL hideState = !self.selectedMVMReportDetailPanelTypeItem ||
!self.selectedMVMReportDetailPanelTypeItem.amountMax;
   [[NSNotificationCenter defaultCenter]
postNotificationName:@"MReportDetailPanelExpensesItemDetailController_Notification" object:self
userInfo:@{@"hideState":@(hideState)} identifier:@"expenseStateVisibility"];
}
```

   i. La méthode calcule un booléen en fonction de l'était du montant maximum dans le type de dépense choisi.
   ii. Ce booléen est envoyé par notification au controller qui gère l'écran. La notification de classe porte le nom de ce controller suffixé par « _Notification ». Afin d'identifier qui est l'auteur de la notification, l'identifiant « *expensesStateVisibility* » est passé en tant que paramètre « identifier ».

10. Ajouter l'écouteur sur le champ « *selectedMVMReportDetailPanelTypeItem* » pour appeler cette méthode à chaque changement du type de la dépense (dans la méthode « *createViewModelConfiguration* » déjà créée auparavant) :

```
@"updateExpenseStateVisibility: selectedMVMReportDetailPanelTypeItem"
```

11. Il faut désormais traiter la notification dans le *ViewController* qui affiche le détail. Il s'agit du ViewController « ***M****ReportDetailPanelExpensesItem****DetailController*** » qui se trouve dans le dossier ***myexpenses/ios/controller/***
    a. Rechercher la balise *@//non-generate-start[other-methods][X]*
    b. Enlever l'indicateur *[X]* pour que le code ajouté entre cette balise et la balise *@//non-generated-end* ne soit pas écrasé à la prochaine génération

c. Ajouter le code suivant :

```
-(void) didReceiveClassNotification:(NSNotification *)classNotification {
    //Check notification identifier
    if([classNotification.identifier isEqualToString:@"expenseStateVisibility"]) {
        //Retrieve descriptor to hide
        MFBindingAbstractDescriptor *descriptor = [self bindingDescriptorAtIndexPath:[NSIndexPath
indexPathForRow:4 inSection:0]];
        descriptor.hidden = [classNotification.userInfo[@"hideState"] boolValue];
        //Reload data
        [self reloadData];
    }
}
```

d. La méthode doit obligatoirement être implémentée, car des notifications sont envoyées à « *MReportDetailPanelExpensesItemDetailController* » via le nom « *MReportDetailPanelExpensesItemDetailController*_Notification ».
    i. L'identifiant de la notification est testé
    ii. La position du descripteur représentant la cellule à cacher est récupérée
    iii. Sa propriété *hidden* prend la valeur donnée dans la notification
    iv. A la fin, la tacle affichant les données est rechargée

12. Compiler l'application
    a. Soit XCode
    b. Soit avec la commande

```
$ mdk platform-compile ios
```

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

5.4. Rendu des montants : XX,XX €

5.4.1. 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**.

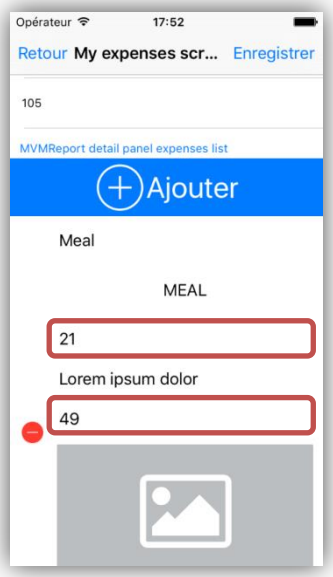

Figure 13 : Champs à formater dans le panel de détail d'une Note.

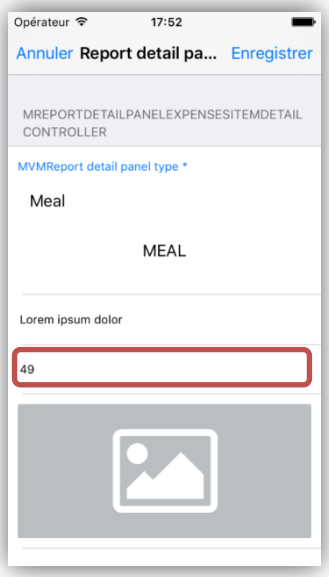

Figure 14 : Champ à formater dans le formulaire d'une Expense.

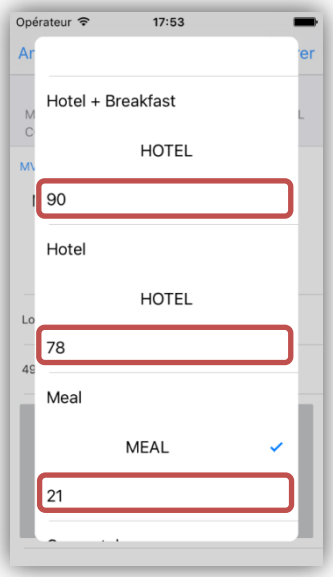

Figure 15 : Champ à formater dans le formulaire d'une Expense.

5.4.2. Implémentation de la surcharge

1. Copier le contenu du répertoire ***myexpenses/hands-on/ios/55 – Amount/viewmodel*** **dans le répertoire** ***myexpenses/ios/myexpenses/src/viewmodel/***
   a. Les éléments copiés représentent la catégorie « *AmountHelper* » sur la classe *MFUIBaseViewModel.* Cette catégorie propose deux méthodes permettant de convertir un nombre en chaîne monétaire et vice-versa. Une fois copiés, référencer dans le projet en cliquant-droit sur « *myexpenses/myexpenses/src/viewmodel* », en choisissant « Add "myexpenses" … » et en sélectionnant les deux fichiers de la nouvelle catégorie
2. Ouvrir les fichiers d'implémentation (.m) des 3 ViewModel suivants :
   a. *MVMReportDetailPanel* (ViewModel représentant une note de frais)
   b. *MVMReportDetailPanelExpensesItemCell* (ViewModel représentant une dépense d'une note de frais)
   c. *MVMReportDetailPanelTypeItem* (ViewModel représentant un type de dépense)
3. Dans chacune de ces classes :
   a. Repérer le commentaire *//@non-generated-start[custom-imports][X]*
   b. Supprimer le *[X]*
      i. Cela indique au générateur que le code généré par défaut a été surchargé
   c. Ajoutez la ligne suivante avant la balise *//@non-generated-end :*

```
#import "MFUIBaseViewModel+AmountHelper.h"
```

   d. Cet import nous permet désormais d'utiliser la catégorie sur chacun des ViewModel cités ci-dessus
   e. Repérer le commentaire *//@non-generated-start[other-methods]*
   f. Ajouter à sa suite les méthodes :

```
-(NSString *) humanReadableAmount {
    return [self humanReadableAmountFromNumber:self.amount];
}
-(void)setHumanReadableAmount:(NSString *)humanReadableAmount {
    self.amount = [self numberFromHumanReadableAmount:humanReadableAmount];
}
```

   g. En fonction du ViewModel, « *self.amount* » devra être remplacé par « *self.amountMax* » ou « *self.amountTotal*
4. Afin d'accepter le symbole « € », les composants doivent être transformés en MDKTextField. Pour cela, remplacer les fichiers suivants :
   a. ***myexpenses/ios/myexpenses/resources/storyboard/MyExpensesScreen.storyboard*** par ***myexpenses/hands-on/ios/55 – Amount/MyExpensesScreen.storyboard***
   b. ***myexpenses/ios/myexpenses/resources/xib/ReportDetailPanelExpensesItemCell.xib*** *par* ***myexpenses/hands-on/ios/55 – Amount/ReportDetailPanelExpensesItemCell.xib***
   c. ***myexpenses/ios/myexpenses/resources/xib/ReportDetailPanelTypeListItem.xib*** *par* ***myexpenses/hands-on/ios/55 – Amount/ ReportDetailPanelTypeListItem.xib***

5. Il ne reste plus qu'à modifier le binding des formulaires pour binder le champ visuel affichant un montant sur le nouveau champ associé (ajouté grâce à la catégorie). Ouvrir les implémentations des 4 classes suivantes :
   a. *MMyExpensesScreenDetailColumn1Controller* (ViewController représentant la colonne de détail d'une notes de frais)
   b. *ReportDetailPanelExpensesView* (Delegate d'une vue représentant une dépense dans la liste des dépenses d'une note de frais)
   c. *MReportDetailPanelExpensesItemDetailController* (ViewController représentant le détail d'une dépense)
   d. *ExpensesTypeListItemBindingDelegate (*Delegate d'une vue représentant un type de dépense)
6. Chacune de ces classes déclarent un binding dans la méthode « *createBindingStructure* ». Un binding est déclaré en créant une configuration sur l'objet qui le déclare. Dans cet objet sont ajoutés des descripteurs ; chacun de ces descripteurs contient des informations liant des propriétés des composants graphiques aux propriétés du ViewModel associé. Pour cette surcharge, il faut modifier la propriété du ViewModel bindée au composant graphique affichant un montant.
   a. Dans la balise ouvrante *@//non-generated-start[…][X]* débutant la déclaration du binding, supprimer l'indicateur *[X]* afin que les surcharges ne soient pas écrasées lors de la prochaine génération.
   b. Pour chacune des binding, repérer les occurrences de : **c.data<->vm[.xxxx].amount** où [.xxxx] représente une éventuelle étape dans le chemin de la proriété du ViewModel
   c. Remplacer la dernière partie du chemin vers la prorpéiété du ViewModel (amount, amountMax ou amountTotal) par notre nouvelle propriété : *humanReadableAmount*
   d. Le résultat suivant devrait être obtenu : **c.data<->vm[.xxxx].humanReadableAmount**
7. Compiler l'application
   a. Soit avec XCode
   b. Soit avec la commande

```
$ mdk platform-compile ios
```

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

5.5. Contenu de l'écran « À propos »

5.5.1. 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

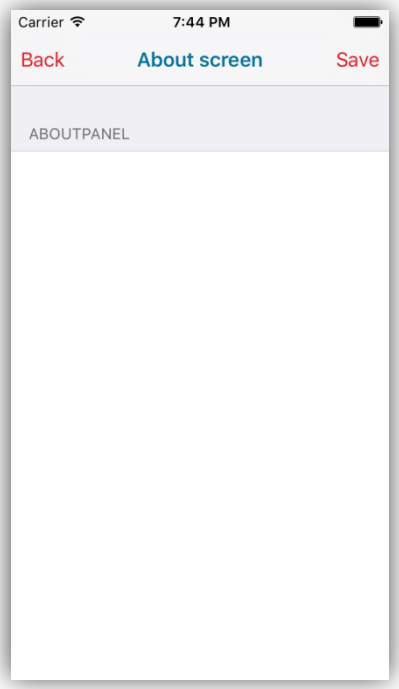

Figure 16 : AboutPanel sans page Web.

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

5.5.2. 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/ios/56 – About***
   b. Dans ***myexpenses/ios/html/***
   c. Ajouter le dossier « html » au projet depuis XCode en créan des « Folder references » (les dossiers ajoutés doivent être colorés en bleu)
2. Dans le ViewModel lié à l'écran *A propos*, surcharger la valeur de l'attribut url :
   a. Identifier la balise *//@non-generated-start[other-methods][X]*
   b. Supprimer l'indicateur *[X]* afin que la surcharge ne soit pas écrasée à la prochaine génération.

c. Avant la balise *//@non-generated-end*, ajouter le code suivant :

```
-(NSString *) url {
    return [[NSBundle mainBundle] URLForResource:@"about" withExtension:@"html" subdirectory:@"html"];
}
```

d. La valeur retournée par l'attribut *url* est désormais le chemin local de la page web à afficher.

3. Compiler l'application
   a. Soit avec XCode
   b. Soit avec la commande

```
$ mdk platform-compile ios
```

4. Déployer et exécuter l'application
5. 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. Design général de l'application

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

1. Exécuter le script de recopie des ressources en exécutant la commande suivante depuis le répertoire du projet MDK

```
$ sh "hands-on/ios/62 - Design/applyDesign.sh"
```

   Ce script recopie toutes les ressources (nouvelles ou modifiées) et les surcharges des classes existantes afin de personnaliser le design de l'application. A la fin de la copie, il ouvre une nouvelle fenêtre Shell dans laquelle il faut saisir la commande affichée lors du premier script. Cette seconde commande exécutée dans le contexte mdk-cli iOS pour la version du MDK spécifiée par le projet en cours, permet de référencer automatiquement toutes les nouvelles ressources dans le projet XCode.
2. A la fin, ouvrir de nouveau le Workspace du projet et exécuter l'application sur le Simulateur iOS depuis XCode pour constater les modifications :

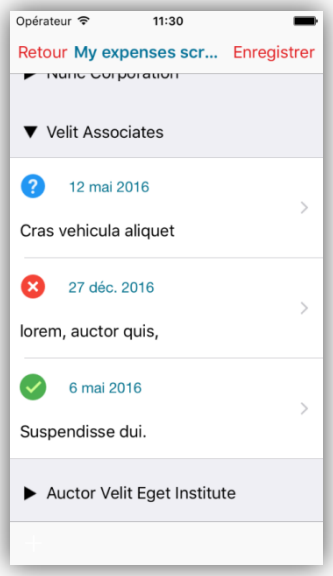

Figure 18 : Application avec

Figure 19 : Application avec

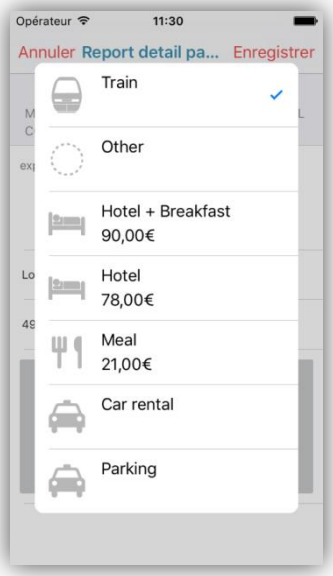

Figure 20 : Application avec

design *design* *design*