

Oh Tannenbaum

Table des matières

1 Analyse préliminaire.....	4
1.1 Introduction	4
1.2 Terminologie.....	4
1.3 Objectifs.....	4
1.4 Planification initiale.....	5
2 Analyse.....	5
2.1 Cahier des charges détaillé.....	5
2.2 Stratégie de test.....	5
2.3 Budget	5
2.4 Etude de faisabilité.....	5
3 Conception.....	6
3.1 Dossier de conception.....	6
3.1.1 Matériel à disposition.....	6
3.1.2 Outils nécessaires:.....	6
3.1.3 Présentation de Sencha-touch :.....	6
3.1.4 Présentation de Phonegap :.....	6
3.1.5 Stockage des informations:.....	6
3.1.6 Fonctionnalités attendues.....	7
3.1.7 Envoi des notifications.....	8
3.2 Historique.....	8
4 Réalisation.....	9
4.1.1 Mise en place de l'environnement.....	9
4.1.2 Configuration du projet dans Eclipse.....	9
4.1.3 Le bootstrap.....	10
4.1.4 Création de cercles.....	10
4.1.5 Vue principale.....	12
4.1.6 Vue de détail d'un cercle.....	13
4.1.7 Liste des contacts du carnet d'adresse.....	15
4.1.8 Ajout manuel de membres.....	21
4.1.9 Informations organisateur.....	23
4.1.10 Tirage au sort.....	25
4.1.11 Résumé du tirage.....	26
4.1.12 Résolution du problème de sauvegarde des relations.....	27
4.1.13 Récolte et envoi des informations au serveur.....	28
4.1.14 Développement du backend.....	30
4.1.15 Renvoi des notifications.....	31
4.1.16 Arborescence de l'application web.....	33
4.2 Description des tests effectués.....	34
4.3 Sources de l'application.....	44
5 Mise en service.....	44
5.1 Liste des documents fournis.....	44
6 Conclusions.....	44

7 Annexes.....	45
7.1 Webographie.....	45
7.2 Journal de travail.....	45

1 Analyse préliminaire

1.1 Introduction

« Secret Santa » est une tradition occidentale dans laquelle chaque membre d'un groupe tire au sort anonymement un autre membre afin de lui offrir un cadeau.

Ce projet porte sur la réalisation d'une application web mobile permettant de simplifier la mise en place du processus de tirage au sort et la notification des résultats de celui-ci.

Il est réalisé dans le cadre de mon TPI en vue de l'obtention de mon CFC d'informaticien et se déroule au sein de l'entreprise Antistatique.net basée à Lausanne.

Afin de ne pas partir de zéro et au vu du temps imparti, Antistatique.net a mis à ma disposition un code de base avec une proposition d'architecture pour l'application.

1.2 Terminologie

Cercle : Groupe de membres qui participent au même tirage au sort

Organisateur : Utilisateur créant le tirage au sort

Contact : Personne dont le profil est présent dans le carnet d'adresse du périphérique

Membre : Personne ajoutée à un cercle

Ange (Angel): Personne désignée pour offrir un cadeau à un autre membre. Elle devient alors son ange

Chanceux (lucky) : Personne désignée pour recevoir un cadeau. Elle devient le chanceux vis-à-vis de son ange.

Notification : Email ou SMS envoyé aux anges pour les informer du membre auquel ils doivent faire un cadeau

1.3 Objectifs

L'application finale devra proposer les fonctionnalités suivantes :

- Création de cercles
- Sélection d'un cercle
- Ajout d'un membre en renseignant manuellement son nom ainsi que son numéro de téléphone ou son email
- Ajout de membres via le carnet d'adresse au cercle sélectionné
- Suppression de membres assignés préalablement à un cercle
- Contrôle du nombre de membres présents dans le cercle sélectionné

- Déclenchement du tirage au sort si il existe au moins trois membres présents dans le cercle courant
- Edition des informations concernant l'organisateur du tirage au sort
- Tirage au sort des relations entre les membres d'un cercle
- Envoi des résultats à un serveur
- Envoi de notifications aux différents membres par Email
- Envoi de notifications aux différents membres par SMS
- Accès à l'historique des tirages effectués

1.4 Planification initiale

Voir fichier annexe « planing_initial.pdf » présent dans le dossier « Annexes »

2 Analyse

2.1 Cahier des charges détaillé

Voir le fichier « cdc_ohtannenbaum.pdf » présent dans le répertoire « Annexes »

2.2 Stratégie de test

Comme l'application finale à pour but d'être multi-plateforme, toutes les fonctionnalités devront être testées sur chacune d'entre elles.

Pour se faire, une procédure de test sera élaborée.

Elle consistera à tester différentes fonctionnalités dans un ordre précis puis de comparer les résultats obtenus avec ceux attendus.

2.3 Budget

Le nombre total d'heures planifiées est de 120. Cela comprend la réalisation de l'application et la rédaction de la documentation

La somme totale des heures indiquées dans mon journal de travail est de 155. J'ai donc dépassé le temps imparti.

2.4 Etude de faisabilité

N'ayant jamais développé d'applications mobiles basées sur Sencha-Touch et Phonegap, je ne peux pas réellement avoir une vision précise des risques techniques.

Pour autant, La complexité ne me semble pas excessivement élevée d'après les objectifs définis.

De mon point de vue, le principal risque est donc mon inexpérience dans ce type de développement et le temps dont je vais avoir besoin pour prendre en main Sencha-touch et PhoneGap.

Le fait de disposer de la durée maximale (120 heures) pour réaliser ce travail permet de minimiser ces risques. De plus, je pourrai compter sur les connaissances et l'expérience de Monsieur Doge en cas de blocage ou de problèmes importants.

3 Conception

3.1 Dossier de conception

3.1.1 Matériel à disposition

Voici le matériel dont je dispose pour la réalisation du projet

- 1 HTC Desire
- 1 Mac sous OS X
- 1 Toshiba Satellite sous Ubuntu 11.04

3.1.2 Outils nécessaires:

Les outils nécessaires sont les suivants :

- Emulateur Android (pour plate-forme 4.0.3) fourni avec le SDK Android
- Outils de débogage Weinre
- Eclipse Indigo (ne fonctionne pas avec la version Galileo)
- Plugin ADT (Android Développement Tools) pour Eclipse
- Frameworks « Sencha-touch 1 » et « Phonegap 1.4.1 »

3.1.3 Présentation de Sencha-touch :

Sencha-touch est un framework destiné à manipuler les plate-formes mobiles.

Il propose

- Un moteur d'affichage en HTML5 / CSS3
- l'intégration de nombreuses icônes et transitions
- La prise en charge de la géolocalisation et du multitouch
- Une manipulation des données facilitée avec Ajax et JSON, et la possibilité de stockage en local

3.1.4 Présentation de Phonegap :

PhoneGap est une librairie qui permet d'accéder aux fonctionnalités du mobile en utilisant le SDK natif de la plateforme cible.

Elle permet aussi de générer une application native pour la plupart des plate-formes mobile. Un site web est d'ailleurs disponible afin d'effectuer cette tâche. Il est disponible à l'adresse <https://build.phonegap.com/>

3.1.5 Stockage des informations:

Sencha-touch fournit un proxy permettant l'utilisation du localStorage au sein de l'application. Afin de pouvoir stocker les données le nécessitant, cette technologie sera utilisée.

3.1.6 Fonctionnalités attendues

Le but de cette application est donc de gérer le processus de tirage au sort effectué dans le cadre de la tradition « Secret Santa ».

Voici le déroulement d'un tirage au sort :

L'utilisateur qui accède pour la première fois à l'application sera invité à créer un cercle dont il devra saisir un titre unique.

Un cercle a pour but de regrouper des membres qui participeront tous au même tirage au sort afin de définir à quel autre membre chacun devra offrir un cadeau.

Plusieurs cercles peuvent être créés en parallèle, mais ils doivent assurer le cloisonnement de leurs membres.

Une fois le premier cercle créé, il est affiché dans une liste regroupant par la suite tous les cercles existants.

Note : Si l'utilisateur qui accède à l'application a déjà créé des cercles, cette liste lui est directement affichée.

Cette liste doit regrouper et faire une distinction entre les cercles dont le tirage au sort a déjà été effectué et ceux qui sont en cours d'édition et dont le tirage au sort n'a pas encore été fait.

L'utilisateur peut ensuite cliquer sur une des entrées de la liste afin d'accéder à la gestion du cercle choisi.

Pour un cercle dont le tirage au sort n'a pas encore été effectué, cette gestion comprend les fonctionnalités suivantes:

- L'ajout de membres via le carnet d'adresse du téléphone
- L'ajout de membres de façon manuelle.
- Le listing des membres associés au cercle
- La suppression de membres associés au cercle
- Le lancement du tirage au sort

Pour chaque membre que l'on ajoute, on doit pouvoir sélectionner (uniquement) un mode de notification.

Il existe deux modes de notification : l'email ou le SMS.

Cela influera plus tard sur la façon dont le membre sera informé du tirage au sort et de la personne à qui il doit faire un cadeau

Un minimum de trois membres est requis afin de pouvoir effectuer le lancement du tirage au sort.

Une fois que l'utilisateur aura décidé d'effectuer le tirage, il sera invité à renseigner les informations suivantes:

- son nom
- son email
- son numéro de téléphone

Elles serviront à avertir les membres de la personne à l'initiative du tirage au sort.

Une fois cette étape passée, l'utilisateur verra apparaître un message d'avertissement l'informant que le tirage au sort est sur le point d'être effectué et que tous les membres présents dans le cercle recevront un message.

Si il valide, le tirage au sort est effectué et les notifications sont envoyées. L'utilisateur reçoit alors une alerte quant au déroulement de l'envoi.

La page de gestion du cercle est alors transformée en une page de résumé.

Elle contient les éléments suivants :

- La date du tirage au sort
- Les membres qui y ont participé

Il n'est alors plus possible d'ajouter ou de supprimer de membres. De même, la fonctionnalité permettant d'effectuer un tirage au sort se voit remplacée par celle de renvoyer les messages de notification.

3.1.7 Envoi des notifications

L'envoi des notifications sera géré en PHP.

Un hébergement de type SimpleHosting small proposé par Gandi est mis à disposition.

Il est rattaché au domaine ohtannenbaum.ch.

Un sous-domaine (api.ohtannenbaum.ch) sur lequel est prévu le déploiement du script d'envoi de mails et de SMS est en place.

Les accès à celui-ci seront transmis de façon confidentielle si besoin.

3.2 Historique

Je n'ai pas eu le temps de mettre en place l'envoi de notifications par SMS.

L'application supporte cependant totalement cette fonctionnalité tant au niveau de l'ajout de contacts via leur numéro de téléphone qu'au niveau de l'envoi de cette information au serveur.

Une condition est en place dans le script PHP afin de pouvoir intégrer facilement cette fonctionnalité.

La plate-forme d'envoi de SMS que j'aurais utilisé si le temps me l'avais permis aurait été <http://www.aspsms.com/>.

4 Réalisation

4.1.1 Mise en place de l'environnement

La mise en place de l'environnement de développement demande un peu de configuration.

Celle que je décris ici est valable pour le déploiement de l'application sur un périphérique Android.

Les procédures de mise en place pour les autres plates-formes mobiles sont détaillées sur le site internet de phonegap à l'adresse <http://phonegap.com/start>.

Voici les principales étapes de mise en place à effectuer :

1. Téléchargement et installation de Eclipse indigo
2. Téléchargement et installation du SDK Android (procédure disponible sur <http://developer.android.com/sdk/installing.html>)
3. Création et configuration d'un dispositif virtuel android à l'aide des outils fournis avec le SDK
4. Installation du plugin ADT (<http://developer.android.com/sdk/eclipse-adt.html>) pour Eclipse
5. Téléchargement de la dernière version de phonegap.

4.1.2 Configuration du projet dans Eclipse

Voici les différentes étapes que j'ai réalisé afin de configurer Eclipse :

1. Démarrer un nouveau projet android dans eclipse.
2. Créer le répertoire www dans assets
3. Copier le fichier .jar fourni avec phonegap dans le répertoire libs
4. Ajouter le .jar de phonegap au buildpath
5. Copier le répertoire « xml » fourni avec phonegap dans le répertoire « res »
6. Modifier le fichier src/app.ohntannenbaum/OhtannenbaumActivity.java de la sorte :

```
package app.ohntannenbaum;
```

```
import android.os.Bundle;  
import com.phonegap.*;
```

```
public class OhtannenbaumActivity extends DroidGap {  
    /** Called when the activity is first created. */  
    @Override  
    public void onCreate(Bundle savedInstanceState) {
```

```
super.onCreate(savedInstanceState);  
super.loadUrl("file:///android_asset/www/mobile/index.html");  
super.setIntegerProperty("loadUrlTimeoutValue", 60000);  
}  
}
```

7. Récupération du bootstrap mis à disposition par Antistatique.net dans un autre répertoire
8. Création d'un lien symbolique entre /assets/www/mobile et le répertoire code/mobile du bootstrap.

4.1.3 Le bootstrap

Le bootstrap mis à disposition par Antistatique.net et récupéré à l'aide de git contient les fichiers suivants :

Filename	Description
code/android/README.md	
code/backend/index.php	
code/ios/README.md	
code/mobile/app/app.js	
code/mobile/app/controllers/circles.js	Contient les fonctions index, show & add
code/mobile/app/models/Circle.js	Contient un modèle pour les cercles
code/mobile/app/views/CircleDetail.js	Contient une vue de détail avec les informations du cercle courant (titre, id, date de création)
code/mobile/app/views/CirclesList.js	Liste les cercles présents dans le store
code/mobile/app/views/Viewport.js	
code/mobile/index.html	
code/mobile/lib/README.md	

Son code source est disponible sur à l'adresse

<https://github.com/antistatique/ohtannenbaum/commit/abf3854ca8021f2bd1ab85fc5fa2914ad4dc68ae>

4.1.4 Création de cercles

La vue affichée par défaut est définie dans le fichier [code/mobile/app/views/Viewport.js](#) de la façon suivante :

```
Ext.apply(this, {  
  items: [  
    ot.views.circlesList  
  ]  
});
```

Cette vue est affichée directement lors du chargement de l'application, sans passer par un contrôleur.

La vue générale étant différente en fonction de l'existence ou non d'un ou de plusieurs cercles, la première étape consiste à ajouter la fonctionnalité d'ajout de cercle à l'application.

Je crée donc le fichier `circleCreate.js` dans le répertoire `views`, je l'inclus dans `index.html` et j'instancie la nouvelle vue dans `ViewPort.js`.

Afin de permettre l'affichage de cette nouvelle vue, je remplace le contenu de la fonction `add` présente dans le contrôleur `ot.controllers.circles` par le code suivant :

```
add: function (options) {  
    var addView = ot.views.circleCreate;  
    ot.views.viewport.setActiveItem(  
        addView,  
        options.animation  
    );  
},
```

Lorsqu'on clique sur le bouton « add » depuis la vue `circleList`, une transition est maintenant effectuée vers la vue `circleCreate`.



Illustration 1: Wireframe de la vue `circleCreate`

Je place à l'intérieur de celle-ci une « Toolbar » que je spécifie en tant que `dockedItem`. Je lui ajoute ensuite un bouton de retour, un titre et une bouton permettant la sauvegarde du cercle.

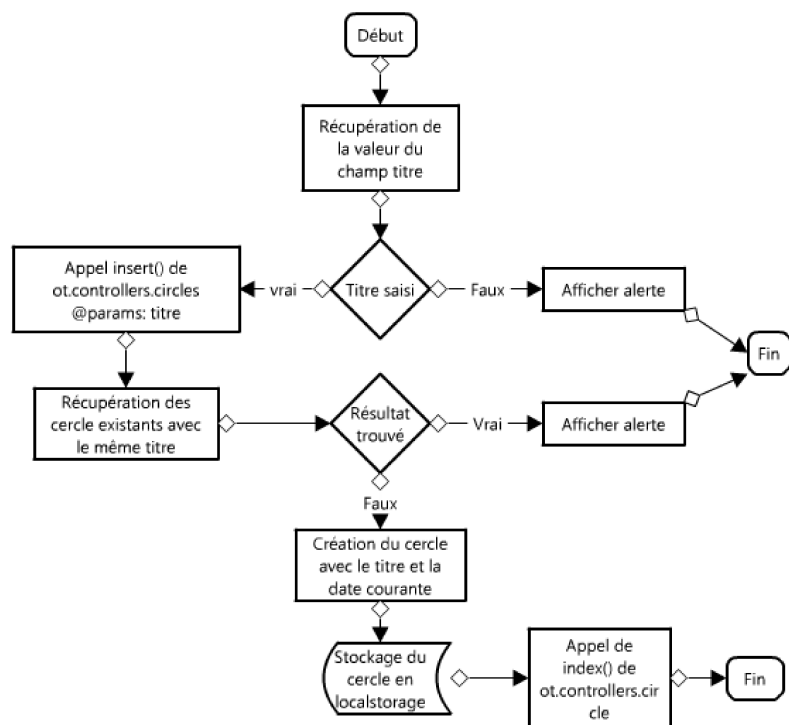
J'ajoute un champ de formulaire aux items de la vue. Celui-ci permet d'insérer un titre au cercle en cours de création.

La sauvegarde du cercle requiert la création d'une fonction spécifique à cette tâche dans le contrôleur `ot.controllers.circles`.

J'y ajoute donc une fonction que je nomme « insert »

J'assigne ensuite des actions aux deux boutons présents dans la toolbar :

1. Lors du clic sur le bouton « back », un appel est effectué à la fonction `index()` du contrôleur `ot.controllers.circles`. Celui-ci a pour effet d'afficher la vue principale.
2. Lors du clic sur le bouton « Sauver », un appel à la fonction « insert » est effectué. Voici son diagramme d'activité :



Made with lovelycharts.com

Illustration 2: Diagramme d'activité de l'action du bouton « Sauver » de la vue `ot.views.circleCreate`

L'ajout de nouveaux cercles est maintenant fonctionnel.

4.1.5 Vue principale

Afin de pouvoir gérer l'affichage de la page principale en fonction de l'existence ou non d'un cercle, je crée la vue `welcomeScreen`.

Cette vue affiche un bref message de bienvenue et invite l'utilisateur à créer son premier cercle.

Elle contient deux boutons permettant d'accéder à la vue de création d'un cercle. Lorsqu'on clique dessus, un appel à la fonction `add` du contrôleur `ot.controllers.circles` est effectué, ce qui déclenche une transition vers la vue `ot.views.circleCreate`.

Maintenant que les deux vues principales existent, il faut mettre en place leur affichage conditionnel.

J'ajoute une condition dans le fichier `ViewPort.js`. Elle permet d'afficher la bonne vue lors de l'initialisation de l'application.

Voici son fonctionnement sous forme de diagramme :

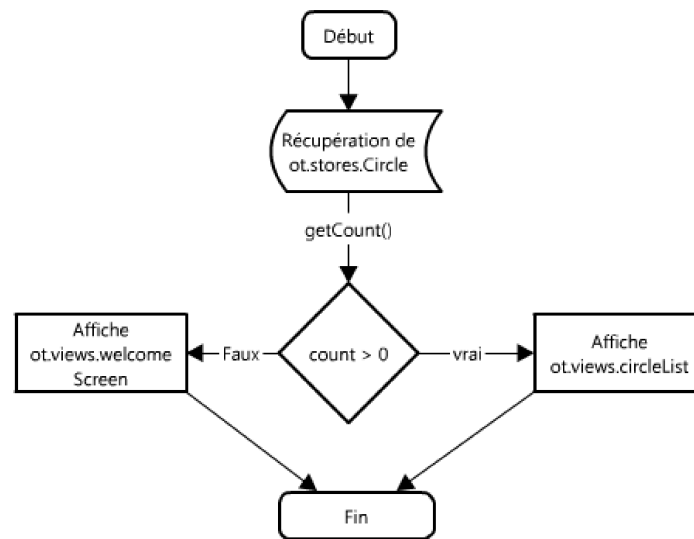


Illustration 3: Diagramme d'activité de la condition d'affiche de la vue d'accueil

Il reste maintenant à effectuer la même condition dans la fonction index du contrôleur `ot.controllers.circles`.

A ce stade, la navigation entre les deux vues principales et la vue de création d'un cercle est fonctionnelle.

4.1.6 Vue de détail d'un cercle

Je peux donc mettre en place la vue `circleDetail` qui servira à effectuer la majorité des actions possibles pour un cercle choisi.



Illustration 4: Wireframe de la vue `circleDetail` ne contenant que l'actionsheet

Je vais dans un premier temps me concentrer sur l'ajout d'une actionsheet dans cette vue. C'est elle qui contiendra les différents boutons d'ajout de membres.

Sa mise en place est un peu différente de celle des autres composants puisqu'elle doit être cachée par défaut et n'apparaître que lorsqu'un clic est effectué sur le bouton « + » présent dans la toolbar.

Pour comprendre la différence, voici une brève explication sur la façon dont un composant est ajouté à une vue :

Chacune des vue de l'application est en réalité une classe héritant de `Ext.Panel`.

Cette classe possède différents attributs qui vont permettre de lui ajouter des composants.

Parmi ceux-ci, on trouve les attributs « items » et « dockedItems ».

Items sert à créer des éléments dont la position est relative, tandis que dockedItems permet de créer des éléments en position fixe (comme les toolbar).

Dès qu'un composant est ajouté à un de ces attributs, il est affiché lors du rendu de la vue.

La solution pour la gestion de l'affichage de l'actionSheet est d'ajouter un attribut à la classe `ot.views.CircleDetail` qui contiendra l'actionSheet. De cette façon, je pourrai y accéder via un autre composant afin d'interagir avec, évitant ainsi qu'elle n'apparaisse par défaut.

Voici le code épuré de la mise en place de l'actionSheet :

```
ot.views.CircleDetail = Ext.extend(Ext.Panel, {
    actionSheet : null,
    initComponents: function() {
        var titlebar, moreButton;
        this.actionSheet = new Ext.ActionSheet({
            items: [{ /* Boutons présents dans l'actionSheet */}]
        });
        moreButton = {
            handler: function(){this.actionSheet.show()},
            scope: this
        };
        titlebar = {
            dock: 'top',
            xtype: 'toolbar',
            title: 'Oh Tannenbaum',
            items: [moreButton]
        };
        Ext.apply(this, {
            scroll: 'vertical',
            dockedItems: [ titlebar ],
            items: []
        });
        ot.views.CircleDetail.superclass.initComponent.apply(this, arguments);
    }
});
```

Il reste ensuite à mettre en place les trois boutons à l'intérieur de l'actionSheet.

Ils s'ajoutent en remplissant le tableau `actionsheet.items` avec des objets formatés de la sorte :

```
{
    text : 'texte du bouton',
    scope: me, //dans la fonction initComponents, me = this. Il fait donc référence à la classe de la vue
    handler : function(){
        //Action au clic
    }
}
```

Afin de pouvoir renseigner les handler, il faut que je mette en place les vues qui seront affichées lors du clic sur les boutons.

4.1.7 Liste des contacts du carnet d'adresse

J'ajoute donc la vue `contactsList` qui servira à afficher les contacts du carnet d'adresse et permettra d'en choisir un afin de l'ajouter au cercle. Je crée par la même occasion un contrôleur nommé `contacts`.

La vue `contactsList` se compose d'une liste groupée par nom. Lors du clic sur un élément, une actionsheet doit apparaître en proposant de choisir une méthode de notification en fonction des informations dont on dispose sur le contact : soit son email, soit son numero de téléphone.

Il faut dans un premier temps récupérer les contacts du carnet d'adresse afin de pouvoir populer la liste.

C'est à ce stade que PhoneGap entre en jeu en mettant à disposition la fonction `navigator.contacts.find(contactFields, contactSuccess, contactError, contactFindOptions);` pour récupérer les contacts du carnet d'adresse.

J'ai rencontré un problème lors de sa mise en place car elle s'effectue de manière asynchrone et le résultat n'est visible que dans la fonction de rappel. Après de multiples tentatives et beaucoup de temps perdu, Monsieur Doge m'a aiguillé sur le code présent à l'adresse <http://www.sencha.com/learn/a-sencha-touch-mvc-application-with-phonegap/>.

J'ai donc mis en place un nouveau modèle nommé « `ot.models.Contact` » servant à gérer les contacts. Voici sa structure :

ot.models.Contact		
<i>Modèle permettant la création et la gestion d'objets représentants les organisateurs</i>		
Fields		
Name	Type	Description
id	Int	Identifiant unique du contact
givenName	String	Prénom du contact
FamilyName	String	Nom de famille du contact
emails	String	Emails du contact
phoneNumbers	int	Numéros de téléphone du contact
proxy		
Type		
contactstorage		

La particularité de ce modèle est son proxy personnalisé.
Il est défini dans le même fichier que le modèle :

```
Ext.data.ProxyMgr.registerType("contactstorage",
Ext.extend(Ext.data.Proxy, {
    create: function(operation, callback, scope) {
    },
    read: function(operation, callback, scope) {
        var thisProxy = this;
        navigator.contacts.find(
            ['id', 'name', 'emails', 'phoneNumbers', 'addresses'],
            function(deviceContacts) {
                //loop over deviceContacts and create Contact model instances
                var contacts = [];
                for (var i = 0; i < deviceContacts.length; i++) {
                    var deviceContact = deviceContacts[i];
                    var contact = new thisProxy.model({
                        id: deviceContact.id,
                        givenName: deviceContact.name.givenName,
                        familyName: deviceContact.name.familyName,
                        emails: deviceContact.emails,
                        phoneNumbers: deviceContact.phoneNumbers
                    });
                    contact.deviceContact = deviceContact;
                    contacts.push(contact);
                }
                //return model instances in a result set
                operation.resultSet = new Ext.data.ResultSet({
                    records: contacts,
                    total : contacts.length,
                    loaded : true
                });
                //announce success
                operation.setSuccessful();
                operation.setCompleted();
                //finish with callback
                if (typeof callback == "function") {
                    callback.call(scope || thisProxy, operation);
                }
            },
            function(e) { console.log('Error fetching contacts'); },
            {multiple: true}
        );
    },
    update: function(operation, callback, scope) {
    },
    destroy: function(operation, callback, scope) {
    }
});
```

Dans sa fonction « read », nous pouvons voir un appel à la fonction `navigator.contacts.find` qui est fournie par `phonegap`. C'est elle qui permet la récupération des contacts du carnet d'adresse.

Le premier argument de cette fonction est un tableau contenant les attributs à récupérer, dans notre cas :

['id', 'name', 'emails', 'phoneNumbers', 'addresses']

Le second argument est la fonction de callback en cas de succès. Les contacts récupérés lui sont passés en paramètre.

Voici un diagramme résumant l'activité de la fonction read du proxy :

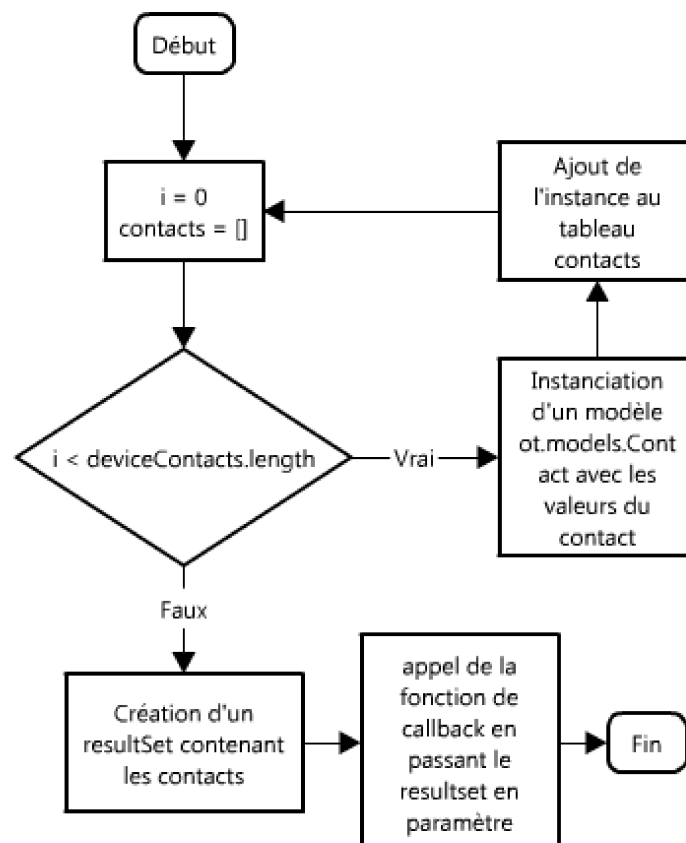


Illustration 5: Diagramme d'activité de la fonction read du proxy contactstorage

Un store est aussi associé au modèle :

```
ot.stores.contacts = new Ext.data.Store({
    model: "ot.models.Contact",
    sorters: 'familyName',
    getGroupString : function(record) {
        if("==" record.get('familyName')) {
            return '-';
        }
        return record.get('familyName')[0];
    }
});
```

C'est grâce à lui que la récupération des contacts depuis la vue va pouvoir se faire.

Je crée donc une liste à l'intérieur de la vue contactsList de la manière suivante :

```
items: [{
    xtype: 'list',
    store: ot.stores.contacts,
    itemTpl: '{givenName} {familyName}',
    grouped: true,
    indexBar: true
}]
```

Celle-ci va automatiquement être peuplée avec les enregistrements présents dans le store `ot.stores.contacts`.

J'ajoute ensuite une fonction « `onItemDisclosure` » à la liste. Cela va générer un bouton sur chaque enregistrement présent dans la liste qui appellera une fonction en lui passant en paramètre l'enregistrement du contact choisi.

Cette fonction sera chargée d'afficher une `actionSheet` possédant des boutons permettant de choisir le mode de notification désiré pour le contact.

Ces boutons doivent donc être générés dynamiquement en fonction des informations disponibles dans l'enregistrement reçu en paramètre.

Voici son fonctionnement :

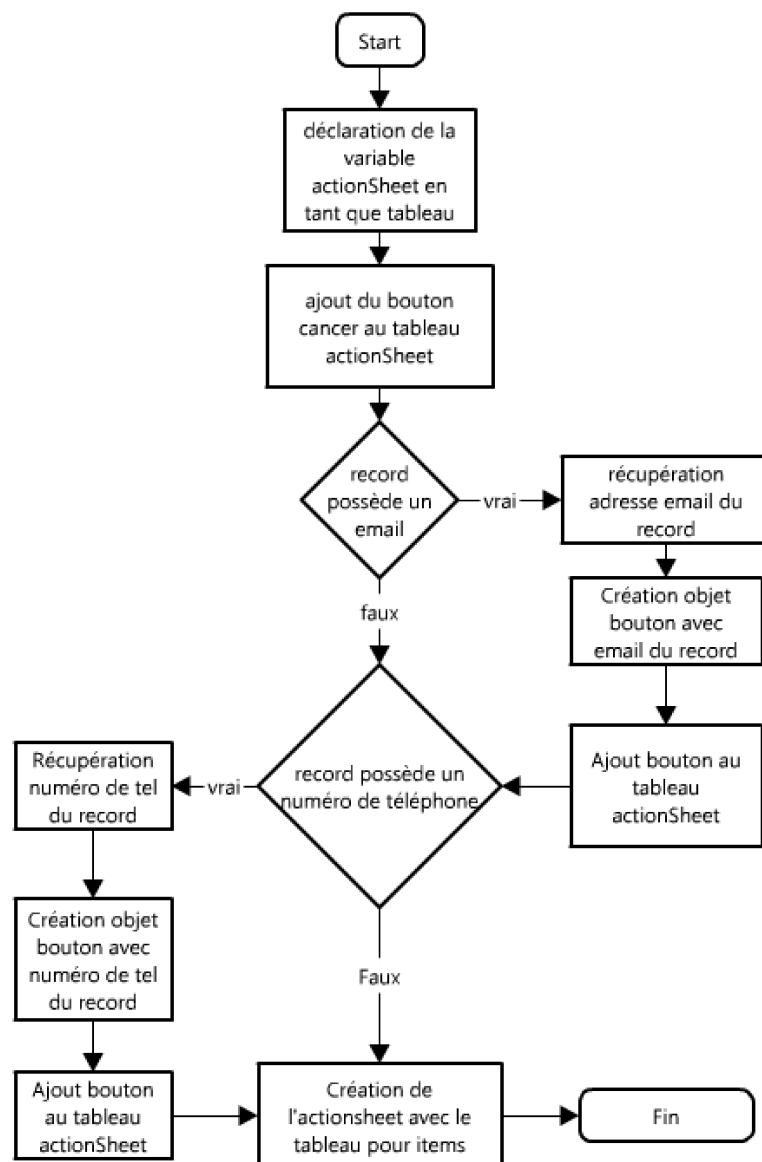


Illustration 6: Diagramme d'activité permettant l'affichage dynamique des boutons de l'actionSheet de la vue `ot.views.contactsList`

Je crée ensuite une fonction `add` dans le contrôleur `ot.controllers.contacts`.

Celle-ci sert à récupérer l'enregistrement et le mode de notification choisi par l'utilisateur dans la liste des contacts.

Elle instancie ensuite un modèle avec ces informations et l'associe au cercle courant.

Elle se charge ensuite de stocker les objets en `localStorage`, de rafraîchir la vue `circleDetail` et d'effectuer une transition afin de l'afficher.

Voici son fonctionnement détaillé :

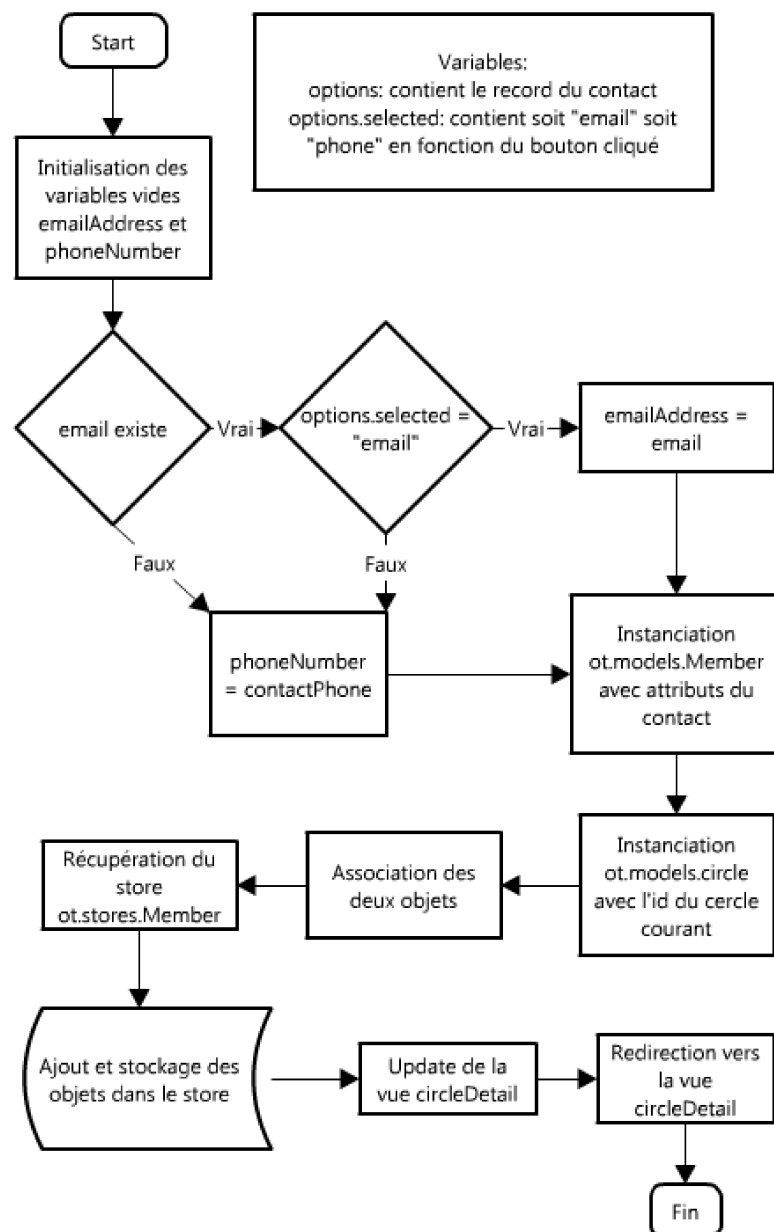


Illustration 7: Diagramme d'activité de la fonction add du contrôleur `ot.controllers.contacts`

L'ajout de membres via le carnet d'adresse est maintenant fonctionnel. Il reste à développer la fonctionnalité d'ajout manuel de membres.

4.1.8 Ajout manuel de membres



Illustration 8: Wireframe de la vue manuallyAddMember

Je crée donc une nouvelle vue que je nomme « manuallyAddMember » et une fonction nommée manuallyAdd dans le contrôleur `ot.controllers.members`.

Cette fonction se chargera d'afficher la vue manuallyAddMember.

Celle-ci se compose de 2 champs de texte et d'une toolbar avec les boutons « Back » et « Ajouter »

L'action du bouton back est un appel à la fonction `show` du contrôleur `ot.controllers.circles`. Celle-ci va se baser sur la valeur de la variable `ot.currentCircleId` pour mettre à jour la vue `circleDetail`.

Voici les quelques lignes concernées :

```
var id = parseInt(options.id) || ot.currentCircleId,  
var circle = store.getByid(id);  
detailView.updateWithRecord(circle);
```

La mise en place des champs dans la vue est la même que dans la vue `circleCreate`.

Par manque de temps, je n'ai pas pu faire une validation complète des données (formatage de l'adresse email ou du numéro de téléphone). La seule vérification que j'effectue repose sur le fait qu'aucun des champs ne peut être vide.

Afin de pouvoir traiter les données insérées par l'utilisateur, je crée une autre fonction que je nomme « insertManually » dans le contrôleur `ot.controllers.members`.

Celle-ci va recevoir en paramètre la valeur des deux champs de la vue. Une expression régulière vérifie la présence du caractère « @ » dans le champ « Email ou mobile » afin de savoir s'il s'agit d'un email ou d'un mobile.

Le reste du traitement effectué par ce contrôleur est le même que celui du contrôleur traitant l'ajout via le carnet d'adresse. Je ne re-détaillerai donc pas son comportement.

Les fonctionnalités de listing des membres d'un cercle, d'ajout de membres via le carnet d'adresse et d'ajout manuel de membres sont maintenant en place.

J'ajoute donc un bouton « Tirage au sort » dans la vue `circleDetail`. Celui-ci doit être inactif par défaut puisqu'il faut au minimum trois membres dans un cercle pour pouvoir effectuer le tirage au sort.

Afin de pouvoir accéder facilement à l'objet contenant le bouton au moment où je devrai l'activer, je crée l'attribut « drawButton » dans la classe `ot.views.circleDetail`.

```
this.drawButton = new Ext.Button({  
  applyTo: 'button-div',  
  text: 'Tirage au sort',  
  handler: function() {  
    //Action au clic  
  },  
  scope: this  
});
```

J'insère ensuite celui-ci dans un panel spécialement créé pour lui :

```
var drawPanel = new Ext.form.FormPanel({applyTo: Ext.getBody(),  
  xtype : 'panel',  
  title : 'draw Panel',  
  frame : true,  
  items : [this.drawButton]  
});
```

drawPanel est ensuite ajouté à la liste d'items de la vue afin qu'il apparaisse. Le bouton n'est pas encore désactivé pour le moment. Afin de le faire, j'ajoute « `this.drawButton.disable()`; » après sa déclaration.

Il faut maintenant que je récupère le nombre de membres associé au cercle lors de chaque ajout/suppression de membre afin de pouvoir activer et désactiver le bouton « tirage au sort » si le nombre de membre est supérieur ou égal à 3.

Comme la vue `circleDetail` est mise à jour à l'aide de la fonction `updateWithRecord` à chaque ajout de membre, je vais effectuer l'activation et la désactivation du bouton à l'intérieur.

Voici le code ajouté dans la fonction `updateWithRecord` :

```
updateWithRecord: function(record) {  
  /* ... code précédent ... */  
  if(memberListLength >= 3){  
    this.drawButton.enable();  
  }else{  
    this.drawButton.disable();  
  }  
},
```

L'activation et la désactivation du bouton en fonction du nombre de membres est maintenant effective.

Afin de renseigner l'utilisateur sur cette condition, je crée un panel qui servira à contenir le texte concernant le nombre de membres restant à ajouter afin de pouvoir lancer le tirage au sort.

Voici comment il est défini dans la vue :

```
var infoPanel = this.infoPanel = new Ext.Panel({
```

```
tpl: new Ext.XTemplate('{infoText}'),  
style: "text-align:center"  
});
```

Ce texte est mis à jour depuis la fonction `updateWithRecord` de la façon suivante :

```
if(memberListLength >= 3){  
    this.infoPanel.update({  
        infoText: 'Vous pouvez ajouter d'autres participants ou faire le tirage au sort'  
    });  
    this.drawButton.enable();  
}else{  
    this.infoPanel.update({  
        infoText: 'Ajoutez encore ' + (3 - memberListLength) + ' membre(s) minimum pour faire le tirage au  
sort'  
    });  
    this.drawButton.disable();  
}
```

Le bouton « Tirage au sort » étant est en place, il faut que je développe la vue qui sera affichée lorsque l'utilisateur cliquera dessus.

4.1.9 Informations organisateur

Afin de rendre le processus de tirage au sort plus logique, j'ai décidé d'inverser l'affichage de deux vues par rapport aux wireframes initiales ;

L'utilisateur sera invité à entrer ses informations organisateur avant que l'alerte de lancement du tirage au sort ne soit affichée.

Je crée donc la vue `drawOwner` ainsi qu'une fonction « `drawOwnerEdit` » dans le contrôleur `ot.controllers.circle`.

Afin de pouvoir stocker les informations sur l'organisateur, je dois préalablement créer un nouveau modèle :

ot.models.Owner		
Modèle permettant la création et la gestion d'objets représentant les organisateurs		
Fields		
Name	Type	Description
id	Int	Identifiant unique de l'organisateur du tirage
name	String	Nom de l'organisateur
email	String	Email de l'organisateur
phone	String	Numéro de téléphone de l'organisateur
circle_id	int	ID du cercle auquel l'organisateur est lié
Associations		
Type	Model	Description

belongsTo	ot.models.Circle	Association de type belongsTo (appartient à) liée au modèle ot.models.Circle
-----------	------------------	--

Problème (voir <http://www.sencha.com/forum/showthread.php?125218-LocalStorage-and-hasMany-association>) : J'ai défini un attribut circle_id car je me suis rendu compte que sencha-touch ne stocke pas les relations en localStorage. Une fois l'application fermée, elles sont donc toutes perdues. Ce champ me permettra de les gérer manuellement.

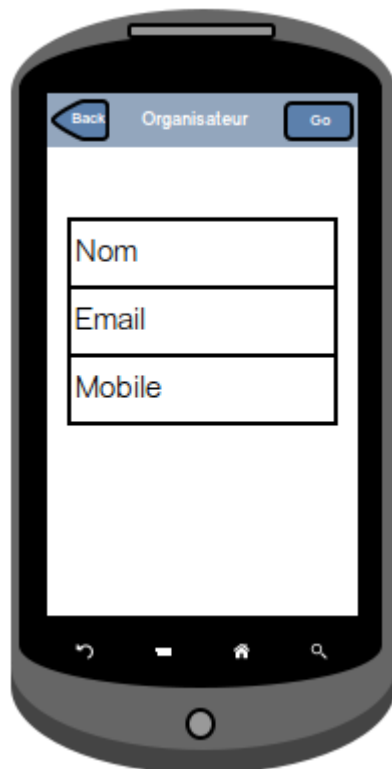


Illustration 9: Wireframe de la vue drawOwner

Il me reste maintenant à créer le formulaire permettant à l'organisateur de renseigner ses informations dans la vue drawOwnerEdit.

Celui-ci se compose de trois champs et sa mise en place est la même que celui de la vue manuallyAddMember.

Lors du clic sur le bouton « Go », une vérification est effectuée afin que les champs ne puissent pas être laissés vides.

Si celle-ci est passée, une demande de confirmation est affichée afin de prévenir l'utilisateur des actions effectuées s'il accepte de lancer le tirage au sort, à savoir l'envoi de mails ou de SMS aux différentes membres.

Si il valide, la fonction doDraw du contrôleur ot.controllers.circles est appelée en envoyant les données de l'organisateur en paramètre.

Cette fonction va dans un premier temps se charger de stocker les informations relatives à l'organisateur :

```
var store = Ext.getStore('ot.stores.Owner'),
    newRecord;
```

```
//Add the owner of the draw
newRecord = {
    name: options.ownerName,
    email: options.ownerEmail,
    phone: options.ownerPhone,
    circle_id: ot.currentCircleId
};
```

```
store.add(newRecord);
store.sync();
```


Elle va ensuite récupérer le store `ot.stores.Member` et réinitialiser les éventuels filtres actifs avant d'en ajouter un.

Celui-ci aura pour but d'exclure tous les enregistrements qui ne possèdent pas un attribut `ot.models.circle_id` de la valeur de l'id du cercle courant.

Voici sa structure :

```
var memberStore = Ext.StoreMgr.get('ot.stores.Member');  
memberStore.clearFilter();
```

```
memberStore.filter({  
  property: 'ot.models.circle_id',  
  value: ot.currentCircleId,  
  exactMatch: true  
});
```

```
var membersArray = memberStore.data.items;
```

La variable `membersArray` contient maintenant un tableau avec les enregistrements des membres rattachés au cercle.

4.1.10 Tirage au sort

Afin de réaliser un tirage au sort de façon simple, j'ai décidé de mettre en place le système suivant :

Je récupère la liste des membres du cercle dans un tableau avant de le mélanger. Les relations sont ensuite définies de la façon suivante :

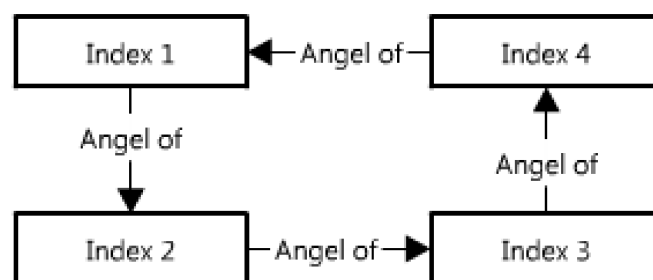


Illustration 10: Définition des relations pour un cercle de 4 membres

Voici la fonction permettant de mélanger le tableau :

```
membersArray.sort(function(){return Math.round(Math.random())-0.5});
```

Il faut maintenant remplir les attributs `angelOf` et `luckyOf` des différents enregistrements et mettre à jour l'attribut `drawDate` du cercle.

Voici le mécanisme qui va s'en charger :

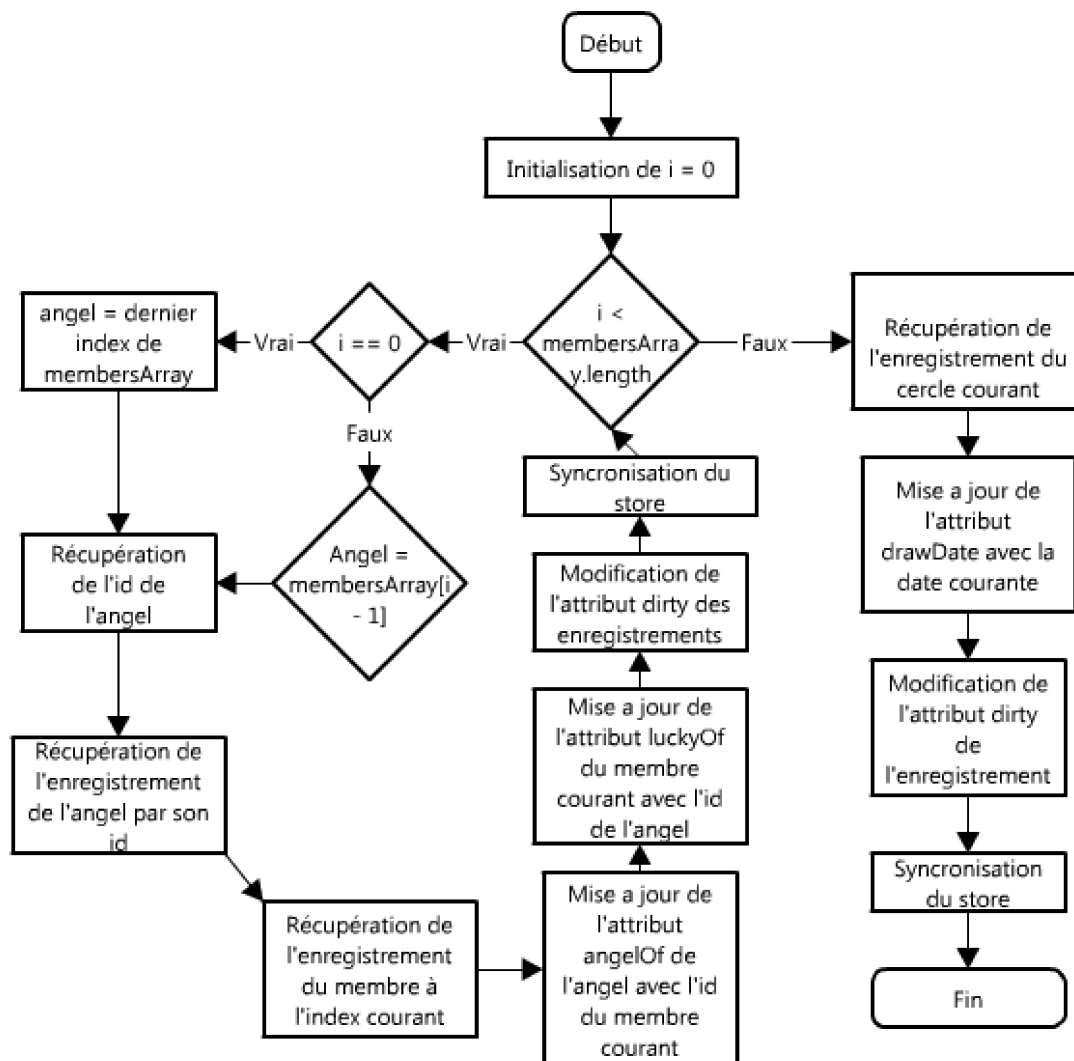


Illustration 11: Définition et stockage des relations entre les membres d'un cercle lors du tirage au sort

Une fois le tirage au sort accompli et le cercle mis à jour, un appel est effectué à la fonction show du contrôleur `ot.controllers.circle`. Celui-ci va faire apparaître la vue `circleDetail`.

Il faut maintenant la modifier puisque le tirage au sort à été effectué.

4.1.11 Résumé du tirage

J'ai donc décidé de créer une autre vue, nommée `circleDrawSummary`. J'ajoute ensuite une condition dans la fonction show de `ot.controllers.circles` afin d'afficher cette nouvelle vue si le tirage au sort à été effectué :

```

if(store.getByld(id).get('drawDate') != null){
    drawSummaryView.updateWithRecord(circle);
    ot.views.viewport.setActiveItem(
        drawSummaryView,

```

```

        options.animation
    );
} else {
    detailView.updateWithRecord(circle);
    ot.views.viewport.setActiveItem(
        detailView,
        options.animation
    );
}

```



Illustration 12: Wireframe de la vue circleDrawSummary

Cette vue est composée d'un texte précisant la date du tirage au sort, d'un bouton permettant de renvoyer les notifications aux membres et la liste des membres qui ont participé au tirage au sort.

La liste des membres est générée de la même façon que celle de la vue circleDetail, et le texte précisant la date du tirage est un panel dans lequel le texte est affiché :

```

var infoPanel = this.infoPanel = new Ext.Panel({
    tpl: new Ext.XTemplate('Tirage au sort effectué le
    {FormattedDrawDate}')
});

```

Un token est placé dans le texte, il servira à afficher la date.

Il est mis à jour depuis la fonction updateWithRecord de la façon suivante :

```

updateWithRecord: function(record) {
    /* ... */
    var recordDate = record.get('drawDate');
    var recordDate = recordDate.format('d/m/Y à H:i');
    this.items.items[0].update({
        FormattedDrawDate: recordDate
    });
},

```

4.1.12 Résolution du problème de sauvegarde des relations

Maintenant que toutes les vues de l'application sont en place, je vais m'occuper de régler le problème de sauvegarde des relations entre les cercles et les membres.

Je commence par modifier le modèle ot.models.member en lui ajoutant le champ « circle_id » de type « int ».

Il servira à stocker l'id du cercle auquel le membre est rattaché.

Afin de renseigner ce champ lors de l'enregistrement des membres, je modifie la fonction « add » du contrôleur « ot.controllers.contacts » en ajoutant le paramètre « circle_id : ot.currentCircleId » lors de l'instanciation de la classe « ot.models.Member ».

Le résultat est le suivant :

```
var myUser = new ot.models.Member({name: options.record.data.givenName + ' ' +
options.record.data.familyName, email: emailAddress, phone: phoneNumber, circle_id:
ot.currentCircleId });
```

Je modifie ensuite la fonction insertManually du contrôleur ot.controller.members de la même façon.

Je peux maintenant me baser sur ce champ pour récupérer les relations.

La prochaine étape consiste à filtrer la liste des membres d'un cercle en fonction de ce champ. Le filtre est appliqué dans la fonction « show » du contrôleur ot.controllers.circles :

```
var memberStore = Ext.StoreMgr.get('ot.stores.Member');
memberStore.clearFilter();
memberStore.filter({
  property: 'ot.models.circle_id',
  value: id,
  exactMatch: true
});
```

Je modifie la ligne « property: 'ot.models.circle_id', » en remplaçant la propriété 'ot.models.circle_id' par 'circle_id' pour obtenir le résultat escompté.

Maintenant que ce bug important est fixé, je vais poursuivre par la mise en place de l'envoi des notifications par email.

4.1.13 Récolte et envoi des informations au serveur

Il faut pour cela que je génère un JSON avec les informations dont j'aurai besoin au niveau du serveur.

Je décide de le structurer de la façon suivante :

```
cercle
  circle_id
  title
  created_date
  drawDate
organisateur
  id
  name
  email
```

```
    phone
    circle_id
membres
  1
    id
    name
    email
    phone
    angelOf
    luckyOf
    circle_id
  2
    ...
  3
    ...
```

Afin de le générer, j'ajoute le code suivant dans la fonction « doDraw » du contrôleur « ot.controllers.circles » :

```
var owner = Ext.StoreMgr.get('ot.stores.Owner');
owner.clearFilter();
owner.filter('circle_id', ot.currentCircleId);
owner = owner.getAt(0).data;
var jsonData = {
    members: new Array(),
    circle: Ext.StoreMgr.get('ot.stores.Circle').getById(ot.currentCircleId).data,
    owner: owner
};
```

Explications :

var owner = Ext.StoreMgr.get('ot.stores.Owner'); me permet de récupérer le store « ot.stores.Owner » dans la variable owner. J'effectue ensuite une réinitialisation des filtres potentiellement présents dessus.

Cela me permet d'être sûr que le filtre que j'appliquerai par la suite soit le seul actif sans quoi le résultat pourrait-être incorrect.

J'applique donc un nouveau filtre sur le store grâce à owner.filter('circle_id', ot.currentCircleId);

Le seul enregistrement que le store me retournera sera celui de l'organisateur du cercle courant.

Je le récupère à l'aide de la fonction getAt : owner.getAt(0).data;

J'initialise ensuite la variable « jsonData » avec un objet possédant 3 attributs. « members » est un tableau vide. Il servira à contenir les enregistrements des membres du cercle

« circle » est initialisé avec l'enregistrement du cercle courant.

« owner » est initialisé avec la variable owner qui contient l'enregistrement de l'organisateur du tirage.

Il reste maintenant à populer le tableau members. Comme une boucle effectuant une itération sur la liste des membres du cercle courant est déjà présente dans la fonction doDraw, je décide d'y placer le code suivant :

```
jsonData.members[i] = membersArray[i].data;
```

Une fois la boucle terminée, le tableau « *membersArray* » contient donc les enregistrement de tous les membres.

Ma variable `jsonData` contient maintenant toutes les informations dont j'ai besoin.

Je peux donc mettre en place la requête Ajax qui va me permettre de la transmettre au serveur.

Afin de ne pas écrire l'URL du serveur directement dans le code, je crée le fichier `config.js` dans le répertoire `app`.

Voici son contenu :

```
ot.config = {  
  notificationServerUrl: 'http://api.ohntannenbaum.ch'  
}
```

Je place ensuite le code suivant à la fin de la fonction `doDraw`, juste avant l'appel à la fonction `show` qui effectue une redirection vers la vue `circleDetail`.

```
Ext.Ajax.request({  
  url: ot.config.notificationServerUrl,  
  method: 'GET',  
  params: {json: JSON.stringify(jsonData)},  
  failure : function(response){  
    alert('failure')  
  },  
  success: function(response, opts) {  
    navigator.notification.alert('Un message à été envoyé aux participants leur indiquant à  
qui ils devront offrir leurs cadeaux.', function() {}, 'Bravo', 'Super!');  
  }  
});
```

On peut y voir que la variable `jsonData` sera envoyée en GET au serveur.

La fonction `JSON.stringify()` que j'applique dessus permet de créer une chaîne de texte JSON à partir d'une structure de données JavaScript.

Deux fonctions de callback sont aussi définies dans ce code : `failure` et `success`. L'une ou l'autre sera appelée en fonction du code HTTP retourné par le serveur.

La fonction `success` affiche une notification via `phonegap` pour alerter l'utilisateur que le processus s'est déroulé avec succès.

La fonction `failure` s'occupe du cas contraire.

Je peux maintenant passer au développement du script d'envoi de notifications

4.1.14 Développement du backend

Je commence par télécharger la librairie Swift (site officiel : <http://swiftmailer.org/>) qui permet d'envoyer des emails.

Je récupère ensuite le contenu de la variable `$_GET['json']` en la passant dans la fonction `json_decode`. Cela a pour effet de me retourner le contenu de la variable sous la forme d'un objet PHP.

J'initialise ensuite un tableau vide que servira à accueillir les membres. Le subtilité réside dans le fait que l'index auquel ils seront stockés correspondra à leur ID.

Voici le code qui s'en charge :

```
<?php
$result = json_decode($_GET['json']);
$members = $result->members;
$sortedMembers = array();

foreach($members as $member){
    $sortedMembers[$member->id] = $member;
}
```

J'effectue ensuite une boucle sur le tableau contenant les membres et je formate à l'intérieur l'email qui sera envoyé au membre.

Le fait d'avoir stocké les membres dans un tableau en fonction de leur ID me permet de facilement accéder aux informations de l'ange ou du chanceux du membre en cours d'itération.

Par exemple, cette ligne suffit pour récupérer le nom de l'ange du membre :

```
$sortedMembers[$member->angelOf]->name
```

Une fois l'email formaté, il est envoyé via Swift.

4.1.15 Renvoi des notifications

Il me reste maintenant une dernière fonctionnalité à implémenter. Il s'agit du renvoi des notifications.

Le bouton servant à effectuer cette tâche est déjà présent dans la vue `ot.views.circleDrawSummary`.

Comme le script qui effectue l'appel Ajax est contenu dans la fonction `doDraw`, j'ai besoin de créer une nouvelle fonction dans le contrôleur.

Cette partie pourrait être ré-écrite de façon plus propre. Il faudrait créer une fonction ne s'occupant que d'effectuer la requête Ajax. Cela permettrait d'éviter de devoir dupliquer du code.

Par manque de temps, je me contenterai du fait que cette partie de mon application est fonctionnelle au détriment des bonnes pratiques... D'avance Je tiens à m'en excuser...

J'ajoute donc la fonction « `resendNotifications` » au contrôleur `ot.controllers.circle`

Je modifie ensuite l'action effectuée en cas de clic sur le bouton de renvoi des notifications afin d'appeler cette nouvelle fonction.

J'ajoute ensuite une pop-up de confirmation dont l'affichage intervient au moment de l'appel à la fonction `resendNotifications`. Celle-ci a pour but de demander une confirmation à l'utilisateur quant au renvoi des notifications.

Le reste du script n'est déclenché que si l'utilisateur clique sur le bouton « Oui ». Dans le cas contraire, la pop-up disparaît et aucune action n'est effectuée.

Si l'utilisateur accepte, les informations sur le tirage sont récupérées et le JSON est reconstruit.

Une requête Ajax est ensuite effectuée à destination du serveur, lequel se charge de renvoyer les notifications.

4.1.16 Arborescence de l'application web


Voici le contenu de l'arborescence de l'application web ainsi qu'une brève description de l'utilité de chaque fichier :

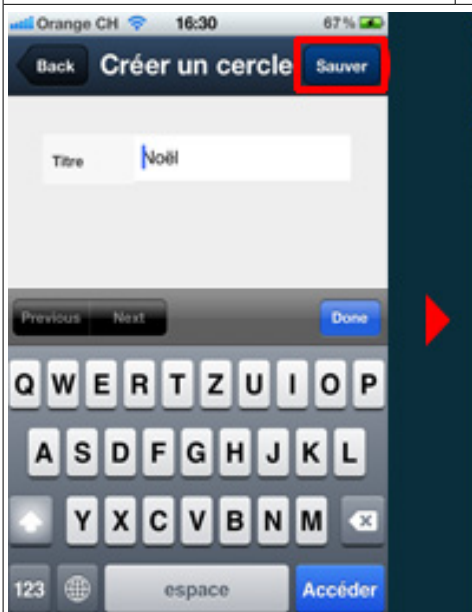

▼ app	
▼ controllers	Dossier contenant tous les fichiers de contrôleur de l'application
circles.js	Contrôleur pour la gestion des cercles
contacts.js	Contrôleur pour la gestion des membres ajoutés manuellement
members.js	Contrôleur pour la gestion des membres ajoutés via le carnet d'adresse
▼ models	Dossier contenant les fichiers de définition des modèles
Circle.js	Fichier content la définition du modèle et du store pour les cercles
Member.js	Fichier content la définition du modèle et du store pour les membres
contacts.js	Fichier permettant la gestion des contacts récupérés dans le carnet d'adresse
owners.js	Fichier content la définition du modèle et du store pour les organisateurs
▼ views	Dossier content les fichiers dans lesquels sont définies les différentes vues
CircleCreate.js	Vue de création d'un cercle
CircleDetail.js	Vue de détail du contenu d'un cercle avant le tirage au sort
CircleDrawSummary.js	Vue de résumé du contenu du cercle après le tirage au sort.
CirclesList.js	Vue principale si un cercle existe. Liste les cercles existants.
ContactsList.js	Vue affichant la liste des contacts récupérés dans le carnet d'adresse
DrawOwner.js	Vue affichant le formulaire servant à insérer les informations organisateur
ManuallyAddMember.js	Vue affichant le formulaire d'insertion de membre de façon manuelle
Viewport.js	Fichier servant à déclarer et initialiser les vues au chargement
WelcomeScreen.js	Page d'accueil si aucun cercle n'existe. Invite à en créer un premier
app.js	Fichier permettant l'initialisation de l'application
config.js	Contient des variables de configuration utilisées dans l'application
▼ lib	Dossier contenant les différentes ressources externes
▶ touch	Dossier contenant les fichiers du framework sencha-touch
README.md	Fichier Readme
moment.js	Librairie de manipulation des dates. Non utilisé.
phonegap-1.4.1.js	Fichier javascript pour le framework phonegap
▼ ressources	Contient les différentes ressources utilisées par l'application (images, css, ...)
▼ images	Dossier content les images utilisées par l'application
homepage.png	Image de fond de la vue welcomeScreen
sapin.jpg	Image du sapin visible dans la vue WelcomeScreen.js
trash-can.png	Image de la poubelle servant d'icônes pour la suppression d'un membre
index.html	Fichier d'index, inclus les différents fichiers javascript et lance l'init de l'application.



4.2 Description des tests effectués



Afin de vérifier le bon fonctionnement de l'application, effectuez les test suivants dans l'ordre.

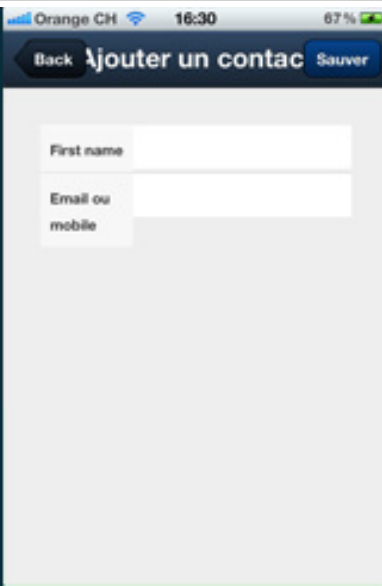
Note : Les captures d'écrans présentes dans ces tests ont été effectués sur un périphérique de type Iphone 4.

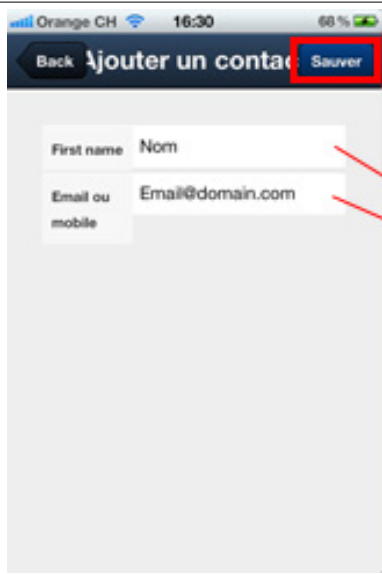

Test à effectuer	Résultat attendu	Description
		<p>Appuyez sur le bouton « + » ou sur le bouton « Créer mon premier cercle ».</p> <p>Vous devez être redirigé vers la vue de création d'un cercle.</p>

Test à effectuer	Résultat attendu	Description
		<p>Remplissez le champ titre avec la chaîne de caractère « Noël » puis appuyez sur le bouton « Sauver ».</p> <p>Vous devez être redirigé sur une vue affichant une liste dans laquelle est présente une entrée ayant pour titre « Noël »</p>


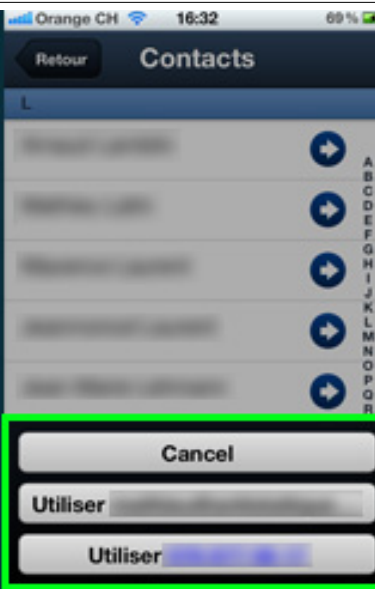
Test à effectuer	Résultat attendu	Description
		<p>Cliquez sur l'entrée ayant pour titre « Noël » dans la liste.</p> <p>Vous devez être redirigé vers la vue de détail du cercle « Noël »</p>

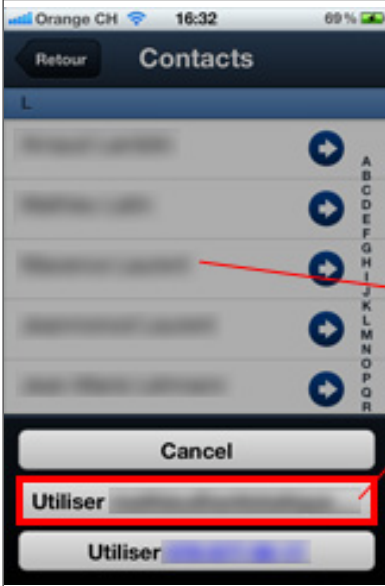

Test à effectuer	Résultat attendu	Description
		<p>Cliquez sur le bouton « + » situé dans la toolbar.</p> <p>Une actionSheet doit apparaître avec 3 boutons : « Ajouter via le carnet d'adresse », « Ajouter manuellement » et « Cancel »</p>

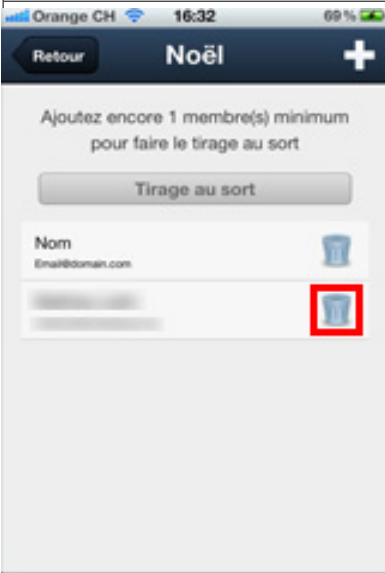
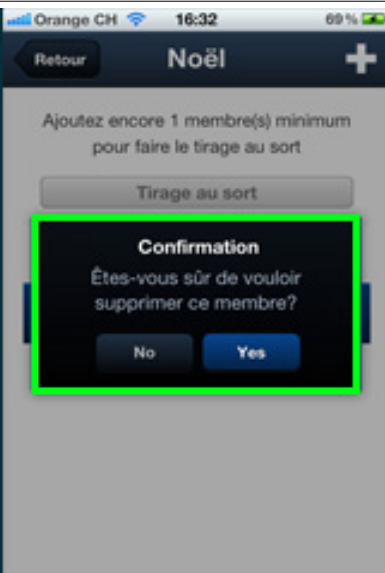
Test à effectuer	Résultat attendu	Description
		<p>Cliquez sur le bouton « Ajouter manuellement » présent dans l'actionDateSheet.</p> <p>Vous devez être redirigé vers un formulaire comportant les champs « First name » et « Email ou mobile »</p>

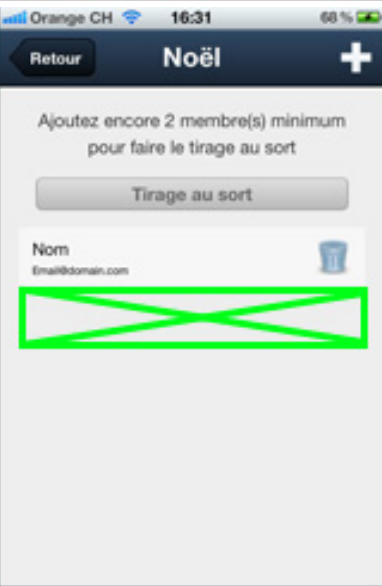
Test à effectuer	Résultat attendu	Description
		<p>Remplissez le champ « First name » avec la valeur « Nom » et le champ « Email ou mobile » avec la valeur « Email@domain.com ».</p> <p>Vous devez être redirigé vers la vue de détail du cercle et voir le nouveau membre apparaître dans la liste des membres.</p> <p>Le texte qui affiche le nombre de membres à ajouter pour pouvoir lancer le tirage au sort doit afficher « 2 ».</p>



Test à effectuer	Résultat attendu	Description
		<p>Cliquez sur le bouton « + » et cliquez ensuite sur le bouton « Ajouter via le carnet d'adresse » situé dans l'actionSheet qui est apparue.</p> <p>Vous devez être redirigé sur une page listant les contacts de votre carnet d'adresse.</p>

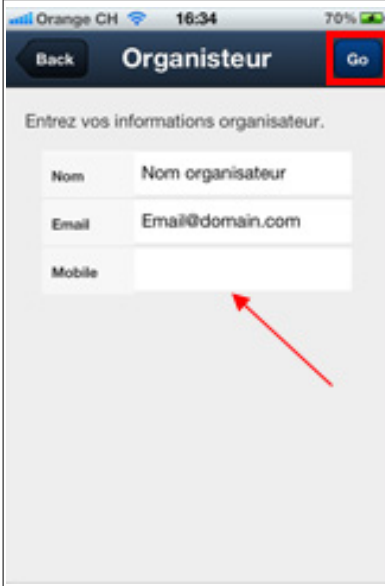
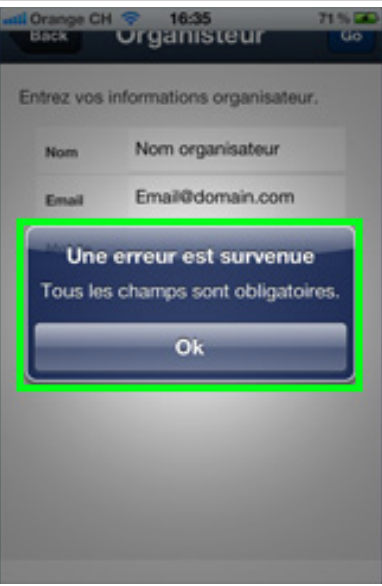
Test à effectuer	Résultat attendu	Description
		<p>Cliquez sur la flèche présente sur la ligne d'un enregistrement.</p> <p>Une actionSheet doit apparaître avec au maximum 3 boutons : « Cancel », « Utiliser {Email} », « Utiliser {Mobile} » et au minimum le bouton « Cancel ».</p> <p>Cela dépend des informations disponible pour le contact sélectionné.</p> <p>Prérequis: Pour effectuer ce test, vous devez disposer d'au moins un contact dans le carnet d'adresse de votre périphérique.</p>


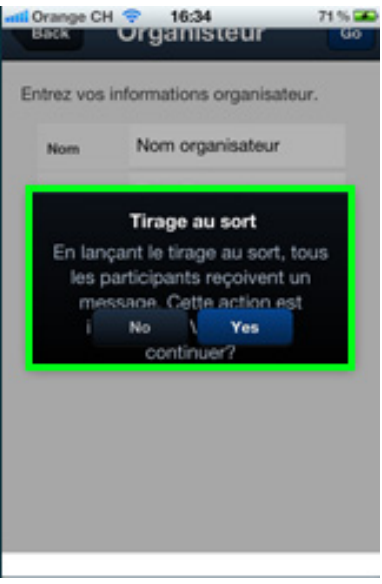
Test à effectuer	Résultat attendu	Description
		<p>Cliquez sur le bouton « Utiliser {Email} ».</p> <p>Vous devez être redirigé vers la vue de détail du cercle et voir apparaître un nouvel enregistrement correspondant au membre qui vient d'être ajouté.</p> <p>Prérequis: Pour effectuer ce test, vous devez sélectionner un contact dont l'adresse email est renseignée dans votre carnet d'adresse</p>

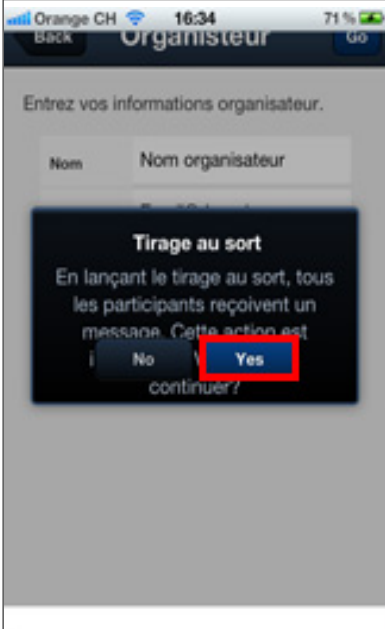

Test à effectuer	Résultat attendu	Description
		<p>Cliquez sur l'icône représentant une corbeille situées sur l'entrée de la liste contenant le membre venant d'être ajouté.</p> <p>Une pop-up doit apparaître pour vous demander confirmation de la suppression de l'élément.</p>

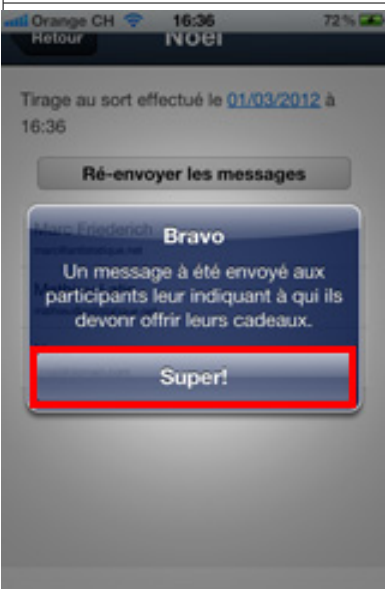

Test à effectuer	Résultat attendu	Description
		<p>Cliquez sur le bouton « Yes » présent dans la pop-up de confirmation.</p> <p>Elle doit disparaître et l'enregistrement de la liste associé à l'action venant d'être effectuée ne doit plus être présent.</p>

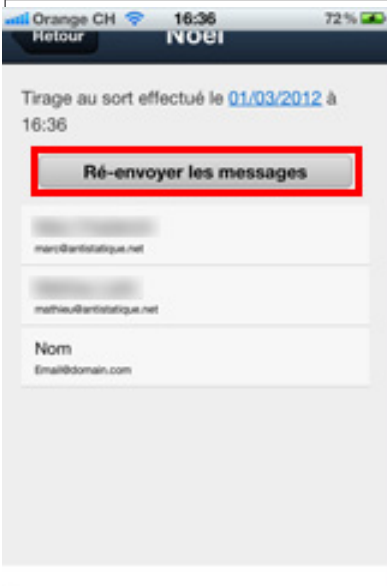
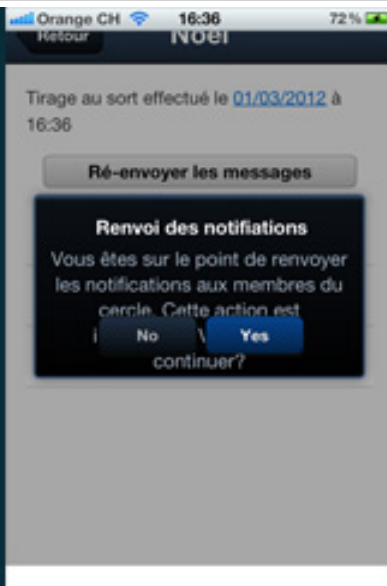
Test à effectuer	Résultat attendu	Description
		<p>Avant d'effectuer ce test, insérez au minimum 3 membres dans le cercle courant.</p> <p>Lorsque 3 membres minimum sont présents dans le cercle, le bouton Tirage au sort doit s'activer.</p> <p>Cliquez dessus.</p> <p>Vous devez être redirigé vers le formulaire d'édition des informations de l'organisateur.</p> <p>Ce formulaire comporte trois champs : « Nom », « Email », « Mobile ».</p>

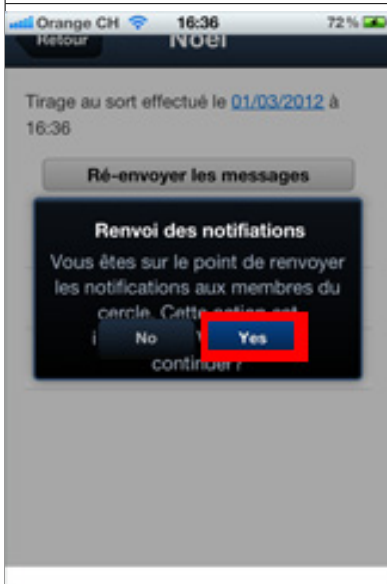

Test à effectuer	Résultat attendu	Description
		<p>Remplissez les champs en omettant d'en remplir un.</p> <p>Cliquez sur le bouton « Go ».</p> <p>Une alerte doit apparaître vous signifiant que les champs sont obligatoires.</p>


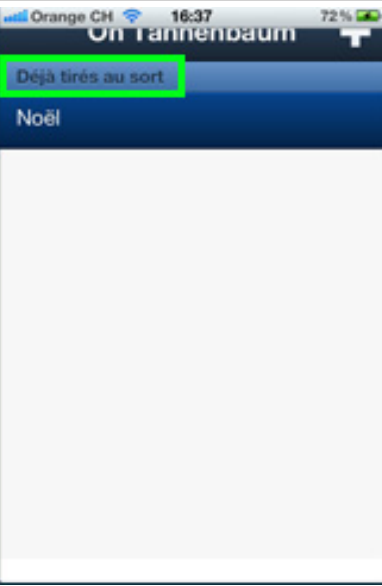
Test à effectuer	Résultat attendu	Description
		<p>Remplissez correctement tous les champs et appuyez sur le bouton « Go ».</p> <p>Une pop-up de confirmation doit apparaître vous demandant de confirmer le lancement du tirage au sort et vous mettant en garde contre le fait que des notifications seront envoyées aux membres du cercle.</p>


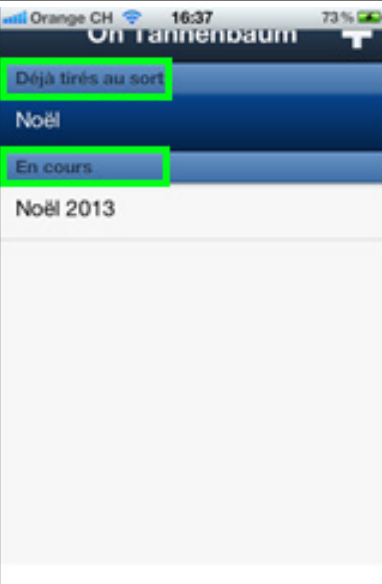
Test à effectuer	Résultat attendu	Description
		<p>Cliquez sur le bouton « Yes » présent dans la pop-up de confirmation.</p> <p>Vous devez être redirigé vers la page de résumé des informations du cercle.</p> <p>Une alerte doit apparaître au bout de quelques secondes pour vous informer du bon déroulement du processus d'envoi des notifications.</p>

Test à effectuer	Résultat attendu	Description
		<p>Cliquez sur le bouton «Super » présent dans la pop-up d'alerte.</p> <p>Elle doit alors disparaître.</p>

Test à effectuer	Résultat attendu	Description
		<p>Cliquez sur le bouton « Ré-envoyer les messages ».</p> <p>Une pop-up de demande de confirmation doit apparaître.</p>

Test à effectuer	Résultat attendu	Description
		<p>Cliquez sur le bouton «Yes».</p> <p>La pop-up doit disparaître et une alerte doit apparaître au bout de quelques secondes dans le but de vous notifier que les messages ont bien été renvoyés.</p>

Test à effectuer	Résultat attendu	Description
		<p>Cliquez sur le bouton « Retour ». Vous devez être redirigé vers la page de listing des cercles existants.</p> <p>Le cercle Noël doit se trouver dans la liste.</p> <p>La label « Déjà tirés au sort » doit être présent au dessus.</p>

Test à effectuer	Résultat attendu	Description
		<p>Ajoutez un cercle nommé « Noël 2013 ».</p> <p>Une fois créer, vous devez être rediriger vers la page de listing des cercles existants.</p> <p>Le cercle « Noël 2013 » doit être présent dans la liste sous le label « En cours » et le cercle « Noël 2013 » sous le label « Déjà tirés au sort »</p>

4.3 Sources de l'application

Les sources de l'application sont disponibles sous forme d'un fichier .zip dans le dossier « Sources » joint à ce document.

Elles sont aussi disponibles publiquement sur github à l'adresse

<https://github.com/antistatique/ohtannenbaum>

5 Mise en service

5.1 Liste des documents fournis

La liste des documents joints à ce rapport de projet sont les suivants :

- Annexes/planing_initial.pdf
- Annexes/journal_de_travail.pdf
- Annexes/wireframe_app_ohtannenbaum.pdf
- Annexes/cdc_ohtannenbaum.pdf
- Sources/sources.zip

6 Conclusions

Au début de ce projet, j'ai rencontré de nombreuses difficultés liées à ma méconnaissance de la technologie utilisée. J'ai eu besoin de beaucoup me documenter afin de pouvoir mettre en place les premières fonctionnalités, ce qui m'a fait prendre un retard relativement important sur mon planning initial.

Heureusement, je me suis senti de plus en plus à l'aise avec l'utilisation de Sencha-Touch au fur et à mesure que les jours passaient. J'ai donc réussi à rattraper la majorité de mon retard et, au final, le seul objectif qui n'est pas atteint est la notification par SMS.

Malgré cela, je suis relativement satisfait du résultat que j'ai obtenu. Je regrette cependant de ne pas avoir eu plus de temps à ma disposition afin de peaufiner l'interface et de ré-écrire certaines parties du code.

Le fait d'avoir eu l'opportunité de découvrir une technologie et d'acquérir de nouvelles compétences m'a beaucoup motivé. J'espère que j'aurai l'occasion de les entretenir et de les développer dans de futurs projets.

Je tiens enfin à profiter de cette conclusion pour remercier chaleureusement toute l'équipe d'Antistatique.net pour son engagement dans la préparation et le suivi de mon travail de diplôme ainsi que pour tout le temps qu'elle a consacré à me former durant mon année de stage.

Je remercie aussi Monsieur Gruaz et Monsieur Montemayor pour leur engagement en tant qu'experts et pour le temps qu'ils consacrent à mon TPI.

7 Annexes

7.1 Webographie

- Documentation officielle de l'API Sencha-touch : <http://docs.sencha.com/touch/1-1/>
- Documentation officielle de l'API PhoneGap : <http://docs.phonegap.com/en/1.4.1/index.html>
- Exemple de développement d'une application de gestion de contacts utilisant Sencha-Touch et PhoneGap : <http://www.sencha.com/learn/a-sencha-touch-mvc-application-with-phonegap/>
- Exemple d'implémentation de composants avec Sencha-Touch <http://dev.sencha.com/deploy/touch/examples/kitchensink/>
- Wiki PhoneGap officiel: <http://wiki.phonegap.com>
- Forum Sencha-Touch : <http://www.sencha.com/forum/>
- LearningCenter Sencha-Touch : <http://www.sencha.com/learn>
- Site de l'application weinre : <http://phonegap.github.com/weinre/>
- Tutorial sur Sencha-Touch : <http://miamicoder.com/2011/writing-a-sencha-touch-application-part-1/>
- Documentation sur les proxy : <http://edspencer.net/2011/02/proxies-extjs-4.html>
- Documentation sur le localStorage : <http://data-that.blogspot.com/2011/01/local-storage-proxy-with-sencha-touch.html>
- Documentation sur l'erreur « The connection To The Server Was Unsuccessful » :
<http://comments.gmane.org/gmane.comp.handhelds.phonegap/7943>
http://community.phonegap.com/nitobi/topics/app_dies_on_startup_connection_to_the_server_was_unsuccessful
- Exemple de création de toolbar : <http://miamicoder.com/2010/creating-sencha-touch-toolbar-buttons/>
- Documentation sur la création de boutons : <http://technopaper.blogspot.com/2008/02/all-about-extbutton.html>
- Divers articles consultés sur <http://stackoverflow.com>

7.2 Journal de travail

Voir fichier « journal_de_travail.pdf » dans le dossier « Annexes »