RICM 4 – Communication & Multimedia

10/04/13

Projet E-cave

Par Jonathan Lemoine
et Nicolas Husson

Introduction :

Ce projet est proposé par Sophie Chareyron et Fréderic Rousseau, tutoré par Didier Donsez et en collaboration avec un groupe de 3i. L'objectif de ce projet est de prototyper un système de gestion et de surveillance (température, hygrométrie) d'une cave à vin. Les bouteilles sont équipées de tags NFC/RFID. Nous avons développé une application mobile qui permet de lire et écrire des tags NFC, de gérer l'inventaire de la cave ainsi que de monitorer la température et l'hygrométrie.

Matériel à disposition :

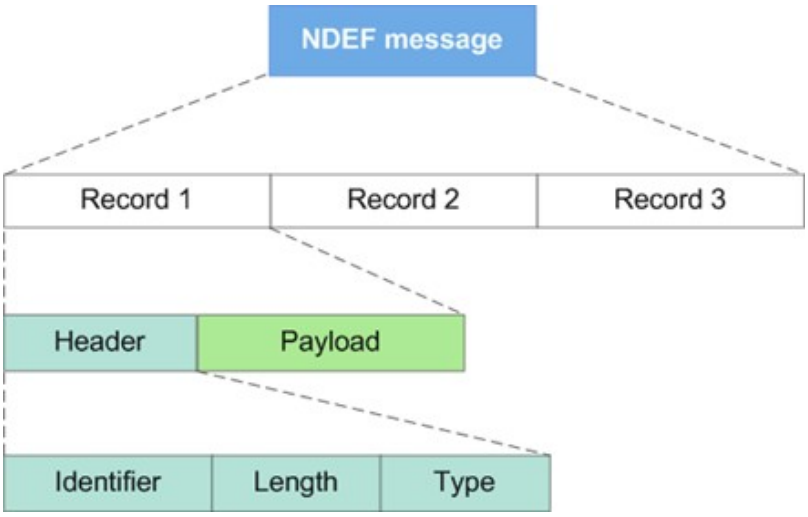
-Touchatag
-Clé USB NFC SCM3711
-Tablette androïd
-Portable androïd NFC
-Tags

Technologie utilisée :

-NFC
-Phonegap
-MongoDB
-Nodejs

I) Phase de recherche

Durant cette phase nous avonc étudié différentes technologies afin de choisir celles qui répondront au besoin de notre projet.
Tout d'abord il a fallu se familiariser avec la technologie NFC. Pour cela nous avons utilisé les applications natives afin d'écrire des tags nfc. Ensuite nous avons essayé d'écrire et de lire des tags avec le touchatag et la clé usb. Pour la clé il a fallu installer libnfc. L'utilisation de la clé usb nous a permis de lire au format héxadécimal les données écrites sur le tag ce qui nous amené à nous questionner sur le format des données du tag (Ndef).



Format Ndef

Ce format contient un tableau de records. Chaque record contient un header et un payload.
Le header contient un identifiant, la longueur du payload et un type (ex : URI).
Le payload est ce qui va contenir les données.

Après avoir avec succès écrit et lu sur les tags avec libnfc nous avons envisagé de faire notre projet en java. Nous nous sommes tourné vers les technologies JNI/JNA mais après réflexion et notre but étant de faire une application mobile nous nous sommes concentré sur les technologies phonegap et androïd et avons délaissé le touchatag et la clé usb. Il a ensuite fallu choisir entre phonegap et androïd. Androïd nous permettait une utilisation simple, de nombreux exemples disponibles cependant la restriction à l'OS androïd ne nous convenait pas et suite à la conférence sur phonegap, notre choix s'est porté sur ce dernier. L'avantage majeur de phonegap est sa portabilité (multi-plateforme) et l'abstraction du code natif a un avenir prometteur auprès des entreprises.

```java
package com.example.test;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;

public class MainActivity extends Activity {

        @Override
        protected void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.activity_main);
        }

        @Override
        public boolean onCreateOptionsMenu(Menu menu) {
                // Inflate the menu; this adds items to the action bar if it is present.
                getMenuInflater().inflate(R.menu.main, menu);
                return true;
        }

}
```

```java
package eCave;

import android.os.Bundle;
import android.view.Menu;
import org.apache.cordova.*;

public class MainActivity extends DroidGap {

        @Override
        public void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                super.loadUrl("file:///android_asset/www/mainPage.html");
        }

        @Override
        public boolean onCreateOptionsMenu(Menu menu) {
                // Inflate the menu; this adds items to the action bar if it is present.
                getMenuInflater().inflate(R.menu.main, menu);
                return true;
        }

}
```

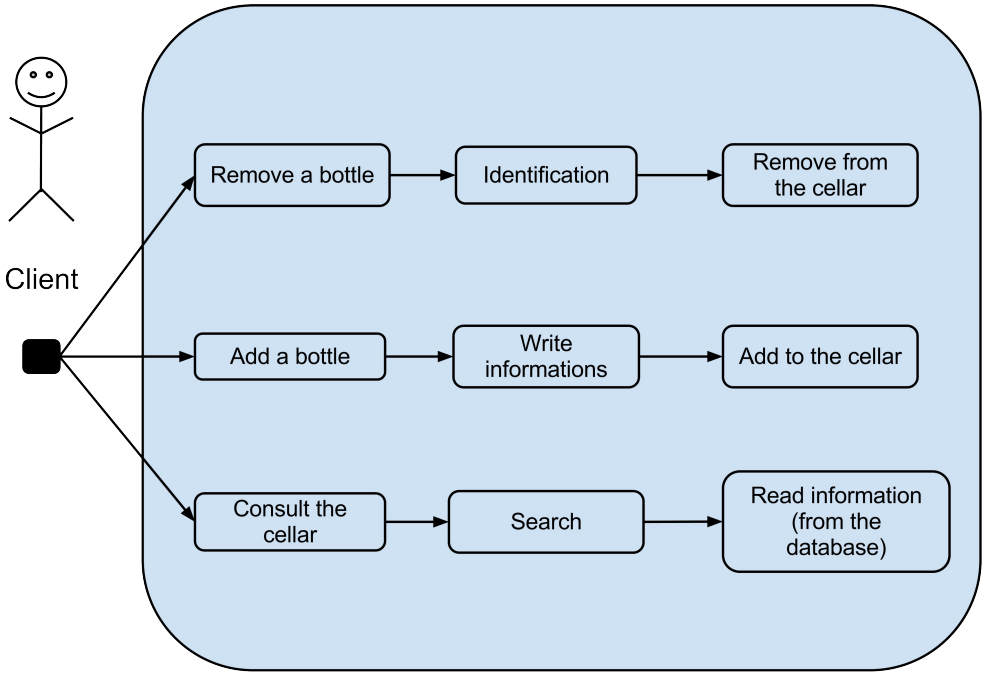
La différence réside dans l'utilisation du package org.apache.cordova. Phonegap

utilise du html pour décrire les différentes pages de l'application et du javascript pour y effectuer des opérations.

Après cette phase de recherche nous avons commencé à modéliser la structure de notre application.

II) Phase de modélisation

Nous avons tout d'abord défini les cas d'utilisations de notre application.

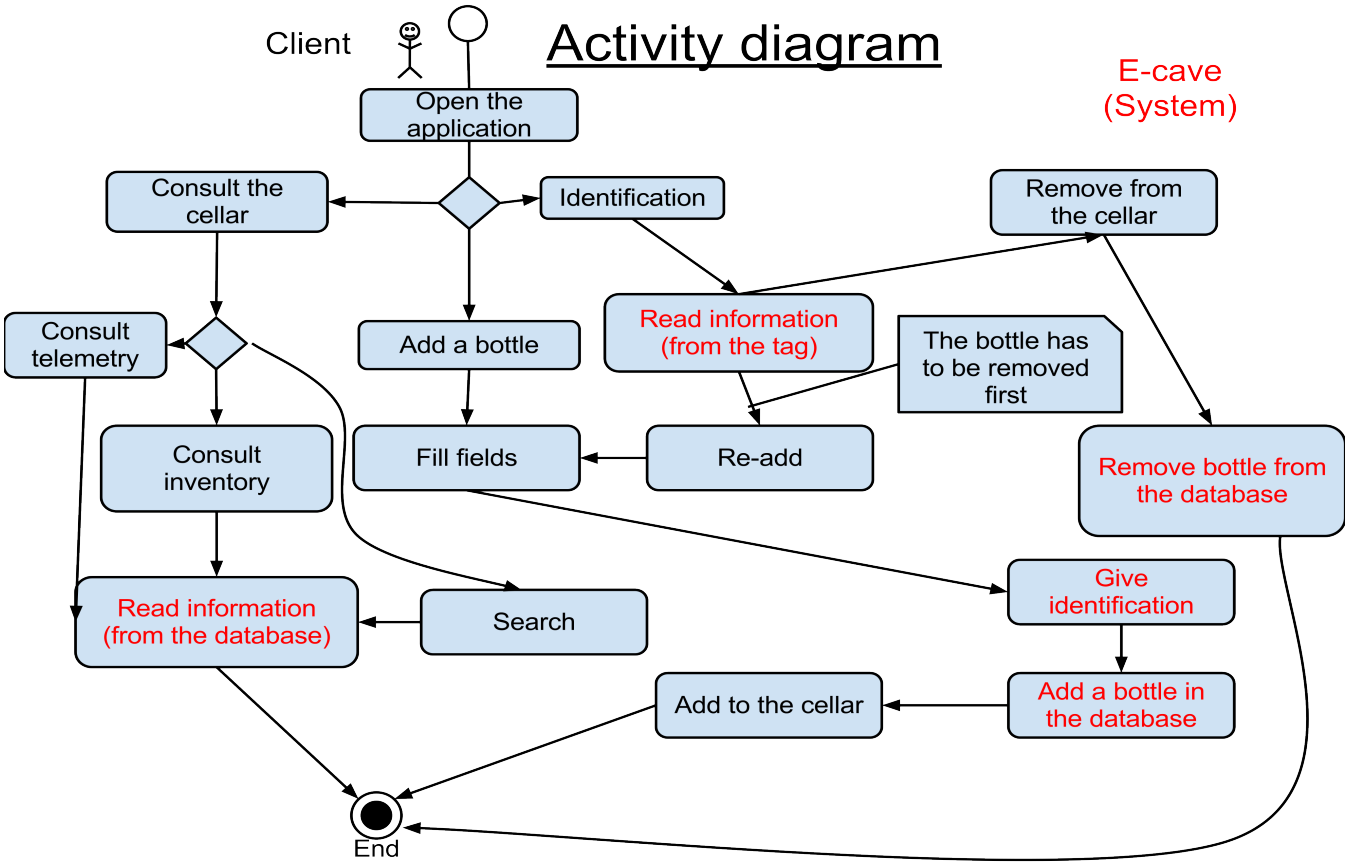


Usecase Diagram

Les trois cas sont :

- l'ajout d'une bouteille
  - ajout d'informations concernant la bouteille
  - ajout de la bouteille dans le cellier
- la suppression
  - identification de la bouteille
  - suppression de la bouteille du cellier
- la consultation de l'inventaire
  - recherche d'une bouteille
  - consultation des informations

Cela nous a amené au diagramme d'activité suivant :



Pour décrire ce diagramme nous allons reprendre les cas d'utilisations mentionnés précédemment.

Ajout d'une bouteille
L'utilisateur peut ajouter une bouteille de deux manières différentes. La première consiste à remplir les champs décrivant la bouteille. La seconde est de pré-remplir les champs à l'aide d'un tag d'une bouteille déja présente. Il place ensuite son tag sur le dispositif nfc ce qui a pour conséquence d'écrire les données sur le tag et de mettre à jour la base de donnée.

Supression d'une bouteille
Dans ce cas, l'utilisateur doit au préalable identifier son tag. Le système lui retourne les informations contenues sur le tag. Il peut alors choisir de supprimer sa bouteille (le système décrémentera le stock de cette bouteille dans la base de donnée) ou bien de préremplir les champs afin d'écrire un nouveau tag.

Consultation
L'utilisateur va dans le menu consulter. Les différentes bouteilles de sa cave sont affichées. Il peut aussi effectuer une recherche (par champs) ainsi que consulter la

télémétrie.

III) Implémentation

a) Fonctionnalités :

Afin d'implémenter notre application, nous avons tout d'abord définit les deux fonctions principales qui nous permettent de lire puis d'écrire sur les tags NFC. Afin de permettre la lecture des tags, il a fallut mettre en place des listeners nous permettant de détecter un tag NFC. Par la suite, on récupère les informations contenues dans le payload du tag avec la fonction suivante :

```
 function displayBottle(bottle) {
    var text = nfc.bytesToString(records[0].payload); // on récupère le champ payload du tag
    text = text.substring(3, text.length); // on supprime les 3 premiers caractères ajoutés par défaut par le dispositif NFC (eng)
    var vin = jQuery.parseJSON(text); // on convertit text en format JSON
    document.bottle.output.value = vin;
};
```

De même pour écrire sur un tag NFC :

```
        var textVin = JSON.stringify ({"typeDeVin": typeDeVin ,"annee":annee,"domaine":domaine,"dateInput": dateInput, "dateOutput": "", "stocked": stocked}); // on transforme le texte en format JSON
        var ndefRecord = ndef.textRecord(textVin); // on le place dans un record
        nfc.write([ndefRecord], function() { // on appelle la fonction write de la bibliothèque nfc
            navigator.notification.vibrate(100);
            alert("Success, the tag has been written");
            $.mobile.changePage("mainPage.html");
        }, function(reason) {
            navigator.notification.alert(reason, function() { // S'il y a un problème lors de l'écriture
            }, "There was a problem");
        });
```

Une fois ces fonctions effectuées nous avons pu complexifier notre application afin qu'elle corresponde au diagramme d'activité :

- pré remplissage des champs
- consultation de la base de donnée
- suppression d'une bouteille
- température / hygrométrie

Le nombre de bouteilles présentes dans la cave est stocké sur la base de donnée, par

l'attribut stocked. Une bouteille peut être présente dans la base de donnée (stocked = 0) et ne pas l'être dans la cave. Cela nous permet d'ajouter/retirer des lots de bouteilles (caisses, cartons …). De plus, si l'on veut ajouter une bouteille qui n'est plus présente dans la cave, le pré remplissage des données peut se faire via la base de donnée. Si l'on ajoute une bouteille déjà présente dans la base de donnée, le champ stocked est modifié en conséquence (prévention des erreurs de l'utilisateur : stock négatif …).

b) Serveur et base de donnée :

Pour créer la base de donnée et le serveur nous avons choisis d'utiliser la plateforme heroku pour notre serveur (nodejs) et pour notre base de donnée (mongoDB), sur les conseils de Didier Donsez. L'utilisation du langage Nodejs pour le serveur nous permet d'uniformiser le langage utilisé du côté serveur et client. La base de donnée MongoDB est une base de donnée qui doit contenir l'ensemble de nos bouteilles, les relevés de températures ainsi que d'hygrométrie sous format JSON (BSON). Le serveur que l'on utilise est mongoLab, une version de MongoDB. Sur cette dernière nous effectuons des requêtes de type http.
Ex :
Accès à la base de donnée :
https://api.mongolab.com/api/1/databases/my-db/collections/my-coll?apiKey=myAPIKey

Recherche les entités où typeDeVin vaut « rouge » :
https://api.mongolab.com/api/1/databases/my-db/collections/my-coll?q={"typeDeVin":"rouge"}&apiKey=myAPIKey

Conclusion :

Tout d'abord, la collaboration avec les 3I4 n'a pas été un succès. Ayant eu des problèmes d'utilisation de capteurs (digitaux, analogiques) ils ont été dans l'incapacité de contribuer à notre projet. Pour simuler des relevés de températures et d'hygrométries nous avons rajouté en dur dans la base de donnée des valeurs. De plus, nous n'avons pas implémenté la notification consistant à prévenir l'utilisateur d'une anomalie par manque de temps. Ce projet fut laborieux de par l'utilisation de langage de programmation et de nouvelles technologies qui nous étaient inconnus. Cependant aprés s'être formé en autodidacte nous avons pu mener à bout ce projet.