

## **Rapport de stage 2A**

# **Développement d'une application mobile multiplateformes**

Année 2012-2013



Rédigé par  
Matthieu NICOLAS

13-15 Boulevard Joffre  
Bâtiment B  
1er étage  
54000 Nancy

## **Remerciements :**

Je tiens à remercier l'équipe de thém-is, et particulièrement M. Mathieu COLIN, pour leur accueil chaleureux ainsi que l'aide et les conseils qu'ils m'ont prodigués tout au long de mon stage.

## Sommaire :

<b>Introduction :</b>	4
<b>L'entreprise :</b>	5
<b>1. Présentation de l'entreprise</b>	5
<b>2. Activité de l'entreprise</b>	5
<b>3. Organisation</b>	6
<b>Le projet :</b>	7
<b>1. Présentation du projet</b>	7
1.1 Problématique	7
1.2 Cahier des charges	7
1.3 Présentation de l'environnement	8
<b>2. L'approche HTML5</b>	12
2.1 Pourquoi l'HTML5 ?	12
2.2 Présentation des outils	12
2.3 L'application HTML5	12
2.4 Les limites de l'HTML5	16
2.5 Bilan	17
<b>3. L'approche hybride</b>	18
3.1 Pourquoi une application hybride ?	18
3.2 Présentation des outils	18
3.3 L'application hybride	21
3.4 Bilan	22
<b>4. Bilan des approches</b>	23
4.1 Comparatif des approches possibles	23
4.2 Améliorations à apporter	23
<b>Conclusion :</b>	25
<b>Annexes :</b>	26
<b>1. Bibliographie</b>	26
<b>2. Glossaire</b>	26
<b>3. Comparatif des solutions de stockage locales</b>	28
<b>4. Fiche récapitulative des bases de l'Objective-C</b>	28

## **Introduction :**

Dans le cadre de notre 2<sup>nd</sup>e année de formation à TELECOM Nancy, nous devons effectuer un stage de fin d'année pour mettre en pratique nos connaissances et méthodes de travail acquises au cours de notre formation et pour acquérir une meilleure vision du monde du travail. Mon stage se déroula du 10 juin au 16 août, au sein de l'entreprise thém-is.

L'offre de stage de thém-is concernait le « Développement d'une application pour iPhone ou Android ». Ce sujet m'intéressa particulièrement, les environnements mobiles étant de plus en plus présents et importants dans notre quotidien. Aucun module de notre formation n'y étant consacré, il s'agissait donc de l'occasion parfaite pour m'initier à ce sujet.

## L'entreprise :

### 1. Présentation de l'entreprise

J'ai effectué mon stage au sein de l'entreprise thé-m-is. Cette entreprise, située au centre-ville de Nancy, est un cabinet de conseil et d'ingénierie informatique créé le 3 mars 2003 et dirigée par 3 associés :

- Lionel LIEGEOIS, Directeur Commercial
- Alain CHOFARDET, Directeur Technique
- Fabien BRUNELLI, Directeur de l'Innovation

Cette SARL compte actuellement une douzaine de collaborateurs. Il s'agit donc d'une entreprise à taille humaine dont la philosophie s'articule autour de 5 valeurs clés :

- L'expertise
- L'innovation
- La réactivité
- L'esprit d'équipe
- L'engagement

Elle a enregistré au cours de l'année 2012 un chiffre d'affaire de 2 millions d'euros.

### 2. Activité de l'entreprise

*La déesse Thémis siège dans l'Olympe, près de Zeus. Il lui confie la charge des missions les plus difficiles et les plus importantes. Thémis est réputée pour son écoute, sa sagesse, sa clairvoyance qui lui permettent de prodiguer des conseils efficaces aux hommes et aux dieux.*

Thém-is est un cabinet de conseil technologique indépendant, spécialisé en ingénierie des systèmes d'information et outils collaboratifs pour les entreprises. L'entreprise a pour rôle de conseiller et d'accompagner les entreprises dans l'élaboration et l'intégration d'applications répondant à leurs besoins.

L'activité de thé-m-is s'articule principalement autour de l'intégration de systèmes, l'environnement IBM Domino en particulier. Elle aide ses clients à déployer ces solutions en analysant leurs besoins et en configurant l'environnement en conséquence. Elle peut aussi assurer diverses formations aux utilisateurs afin de faciliter leur transition.

Afin de répondre aux besoins de clients n'étant pas pris en charge par ces solutions, thém-is développe des applications s'implémentant au sein des systèmes utilisés. Avec l'apparition des systèmes mobiles, thém-is a étendu son activité dans le but de permettre à ses clients d'intégrer ces appareils au sein de leur environnement existant.

Finalement, thém-is gère aussi la maintenance des systèmes et applications qu'elle a déployés, assurant un service client lorsque des problèmes sont rencontrés par les utilisateurs.

### 3. Organisation

L'entreprise est organisée de la manière suivante :

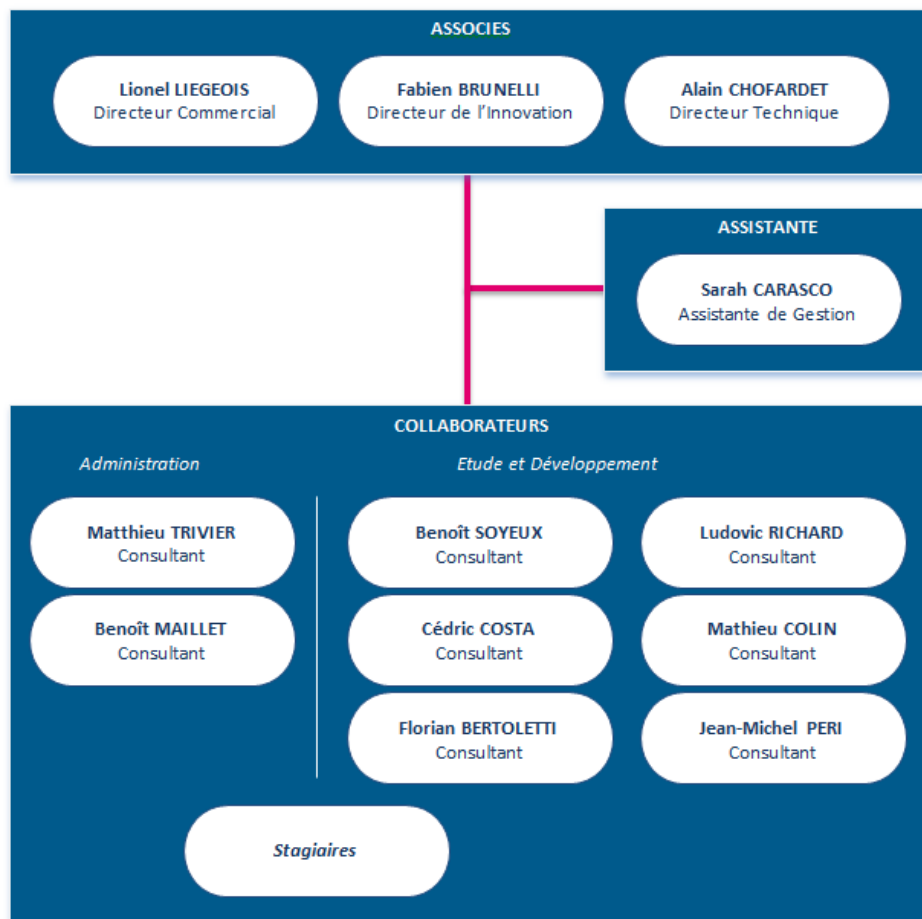


Figure 1 - Organigramme de thém-is

J'ai été intégré pendant mon stage au service d'étude et développement, travaillant de façon autonome et indépendante sur le projet qui m'a été confié.

## Le projet :

### 1. Présentation du projet

#### 1.1 Problématique

Actuellement deux systèmes d'exploitation destinés aux mobiles dominent le parc informatique des clients de thé-m-is : Android et iOS. Chacun de ces OS met à disposition de nombreux outils de développement nous permettant de concevoir des applications. Celles-ci sont alors dédiées et spécifiques à l'environnement pour lequel elles ont été conçues et sont dites natives.

De ce fait, une application développée en Objective-C pour iOS par exemple ne fonctionnera uniquement sous iPhone et iPad. Il peut donc s'avérer difficile d'expliquer aux clients que renouveler leur parc informatique mobile et changer de système d'exploitation impliquera aussi des coûts pour le portage de leur application vers le nouvel OS.

Le développement d'applications multiplateformes est donc primordial, permettant ainsi de réduire les coûts de développement globaux mais aussi d'offrir une plus grande flexibilité aux clients.

Ma tâche fut donc de rechercher et de tester différentes approches permettant le développement d'applications mobiles multiplateformes.

#### 1.2 Cahier des charges

Le projet qui m'a été confié était donc la réalisation d'une application de test. Celle-ci devait répondre à différents besoins :

##### *a) Gestion de la base de données*

L'utilisateur dispose d'une base de données décrite par le schéma relationnel suivant :

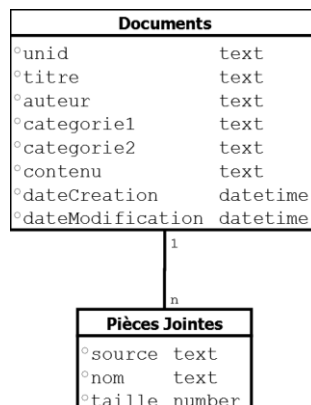


Figure 2 - Schéma relationnel de GestionDocs

L'application doit donc permettre d'interagir avec la base de données des façons suivantes :

- Consulter les données des différents documents
- Créer de nouveaux documents
- Éditer les documents existants
- Supprimer un document

#### *b) Mode non-connecté*

L'application étant destinée à une utilisation mobile, il faut aussi prendre en compte les contraintes d'un tel environnement, la principale étant l'absence possible de connexion Internet. Un mode non-connecté est donc nécessaire.

Ce mode doit permettre à l'utilisateur de consulter les données des documents qu'il a préalablement choisi de stocker localement et toujours fournir les fonctions de création, d'édition et de suppression.

#### *c) Synchronisation des données*

La présence d'un mode non-connecté implique que, lorsqu'on récupère une connexion Internet, un processus de synchronisation des données se déclenche. Celui-ci doit transmettre les modifications effectuées localement au serveur, mais aussi mettre à jour les données des documents stockés en local et potentiellement créer des versions de conflits de documents si ceux-ci ont été modifiés par un autre utilisateur depuis notre dernière connexion.

#### *d) Gestion des pièces jointes*

L'application doit aussi permettre à l'utilisateur de consulter les pièces jointes attachées aux documents. Il faut donc pouvoir les télécharger sur l'appareil afin de pouvoir les afficher même en mode non-connecté.

### **1.3 Présentation de l'environnement**

En tant que partenaire IBM, l'entreprise thém-is est spécialiste de l'environnement « IBM Notes et Domino ». Il s'agit d'une plate-forme offrant des fonctionnalités de messagerie et d'agendas ainsi que permettant de déployer des applications WEB.



Figure 3 - Logo d'IBM Notes



### *a) Base de données documentaire*

La plate-forme « IBM Domino » utilise une base de données documentaire afin de stocker les données. Une base de données de ce type possède un fonctionnement différent que des bases de données traditionnelles de type relationnel. Celle-ci est appelée NSF (Notes Storage Facility) et s'apparente à une archive contenant les données stockées mais incluant aussi d'autres fichiers utilisés dans le cadre du développement d'application.

### *b) Les Documents*

Chaque entrée dans une base de données documentaire est appelée « Document ». Un document se voit attribuer un identifiant unique, appelé Unid. Le document dépend ensuite du masque utilisé pour sa création.

### *c) Les Masques*

Il s'agit de masques permettant de définir une entrée dans la base de données documentaire. Ils prennent l'apparence d'un formulaire, composé de champs de différents types.

**Document**

Titre:

Auteur:

Auteur:

Catégorie 1:

Catégorie 2:

Date création:

Date dernière modification:

Contenu:

Pièces Jointes:

**Figure 4 - Masque correspondant à une entrée de Documents**

Il est intéressant de noter que dans le cas où l'on modifie un masque après sa création, les modifications ne sont pas répercutées sur les Documents déjà existants. Il est donc possible d'avoir des Documents correspondant à des masques différents mais provenant d'un seul et même masque.

#### d) Les Vues

Les vues sont utilisées afin d'afficher les entrées issues d'un formulaire. Elles sont paramétrables, ce qui permet de ne faire apparaître uniquement les champs souhaités. Elles sont notamment utilisées pour récupérer l'ensemble des documents issus d'un masque.

Date Création	Auteur	Titre	Catégorie 1	Catégorie 2	Date Modification	Contenu
22/07/2013	Matthieu Nicolas	Nouveau doc	Informatique	Matériel	22/07/2013 16:20:56	Bonne date ?
16/07/2013	Matthieu Nicolas	Essai modifié	Cuisine	Livre de poche	23/07/2013 10:54:02	Des fichiers, tout plein de fichiers...
12/07/2013	Matthieu Nicolas	Doc édité 3	Cuisine	Livre de poche	22/07/2013 13:17:49	La cuisine pour les nuls...
12/07/2013	Matthieu Nicolas	Doc 2 édité	Informatique	Matériel	16/07/2013 16:10:52	Il est vivant... Ou pas...

Figure 5 - Exemple d'une vue listant les entrées du formulaire Document

#### e) Les XPages

XPages est une technologie de développement WEB permettant de développer des applications interagissant avec les composants Domino présentés précédemment. Cette technologie prend la forme d'une surcouche du langage HTML et est parfaitement compatible avec les nouveautés HTML5, ainsi que les langages JavaScript et CSS.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <xp:view xmlns:xp="http://www.ibm.com/xsp/core" xmlns:xc="http://www.ibm.com/xsp/custom">
3   <meta name="viewport" content="width=device-width, initial-scale=1" />
4   <xc:Ressources></xc:Ressources>
5
6   <div data-role="page">
7
8     <xc:Header></xc:Header>
9
10    <div id="divContent" data-role="content">
11
12      <xp:button value="Label" id="button1"></xp:button>
13      <xp:checkBox text="Label" id="checkBox1"></xp:checkBox>
14      <xp:link escape="true" text="Link" id="link1"></xp:link>
15
16    </div><!-- /content -->
17
18    <xc:MenuFooter></xc:MenuFooter>
19
20  </div><!-- /page -->
21
22 </xp:view>

```

Figure 6 - Exemple de XPage

### f) Les Agents

Les agents, codés en Java ou en LotusScript, sont utilisés afin de déclencher des traitements côté serveur depuis une XPage. Ils peuvent donc être utilisés pour la gestion des documents par exemple, afin de récupérer un document en particulier ou d'en créer un...

```
public class CreateDocument extends AgentBase {  
    public void NotesMain()  
    {  
        try  
        {  
            Session session = getSession();  
            session.setConvertMime(true);  
            Database database = session.getCurrentDatabase();  
            AgentContext agentContext = session.getAgentContext();  
            Document documentContext = agentContext.getDocumentContext();  
  
            // Query string.  
            if(documentContext.hasItem("Query_String_Decoded"))  
            {  
                String queryString = documentContext.getItemValueString("Query_String_Decoded");  
                System.out.println(queryString);  
                String[] tab = queryString.split("&");  
                HashMap parameters = new HashMap();  
  
                for(int i = 0; i < tab.length; i++)  
                {  
                    String s = tab[i];  
                    int index = s.indexOf("=");  
                    if(index > -1)  
                    {  
                        String name = s.substring(0,index);  
                        String value = s.substring(index + 1);  
                        parameters.put(name,value);  
                    }  
                }  
  
                //Le type Document ici n'a rien à voir avec le nom du Form  
                //On récupère le document modifié  
                Document document = database.createDocument();  
            }  
        }  
    }  
}
```

Figure 7 - Exemple d'Agent

## 2. L'approche HTML5

### 2.1 Pourquoi l'HTML5 ?

L'HTML, avec le JavaScript et le CSS, nous permet de concevoir des applications web qui sont fonctionnelles pour tous les appareils possédant un navigateur internet. De plus, avec sa 5<sup>ème</sup> version, l'HTML s'est dotée de nombreuses fonctionnalités permettant de répondre aux besoins du cahier des charges. En effet, de nouvelles API ont fait leur apparition permettant d'utiliser une base de données au sein du navigateur ou même de créer un espace de stockage de fichiers. Les XPages étant une surcouche de l'HTML, il est donc possible de profiter de ces nouvelles fonctionnalités de l'HTML5 tout en utilisant les composants des XPages afin d'interagir aisément avec l'environnement Domino.

### 2.2 Présentation des outils

#### *a) IBM Designer*

IBM Designer est une surcouche d'Eclipse faisant partie de l'environnement IBM Domino. Ce logiciel permet de concevoir des applications pour cet environnement, en mettant à disposition de nombreux composants graphiques pour faciliter le développement des Masques, XPages...

#### *b) jQuery Mobile*

jQuery Mobile est une déclinaison pour mobile du framework JavaScript et CSS jQuery. Celui-ci met à disposition de nombreux composants graphiques, et facilite grandement l'écriture de scripts en instaurant une syntaxe rapide et simplifiée du JavaScript.

### 2.3 L'application HTML5

#### *a) Gestion de documents*

En interrogeant l'URL d'une vue, nous pouvons récupérer au format JSON l'ensemble des informations affichées par celle-ci.

```
{
  "@timestamp": "20130731T091842,69Z",
  "@toplevelentries": "5",
  "viewentry": [
    {
      "@position": "1",
      "@unid": "AE53BB05A8595CF1C1257BB0004ED8BE",
      "@noteid": "A5A",
      "@siblings": "5",
      "entrydata": [
        {
          "@columnnumber": "0",
          "@name": "fl_DateCreation",
          "datetime": {
            "0": "20130722T162056,00+02",
            "@dst": "true"
          }
        },
        {
          "@columnnumber": "1",
```

Figure 8 - JSON récupéré d'une vue

Grâce à ces données, l'application crée la liste des documents :

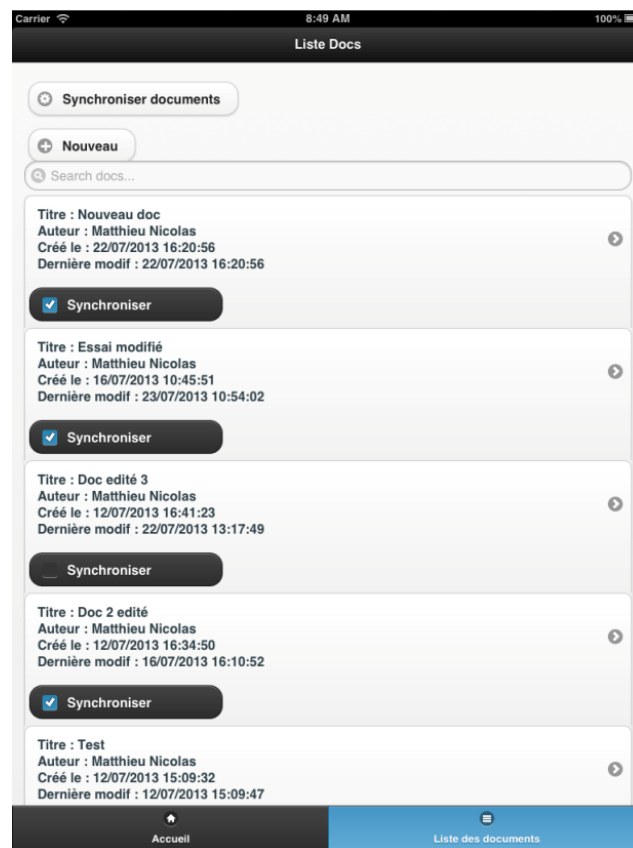


Figure 9 - Capture d'écran de la liste des documents

Cette liste permet au client, en cliquant sur un document, d'interroger un agent. L'unid du document souhaité étant envoyé à l'agent, celui-ci nous renvoie les données complètes de ce document au format JSON et nous permet de les afficher dans la page suivante :

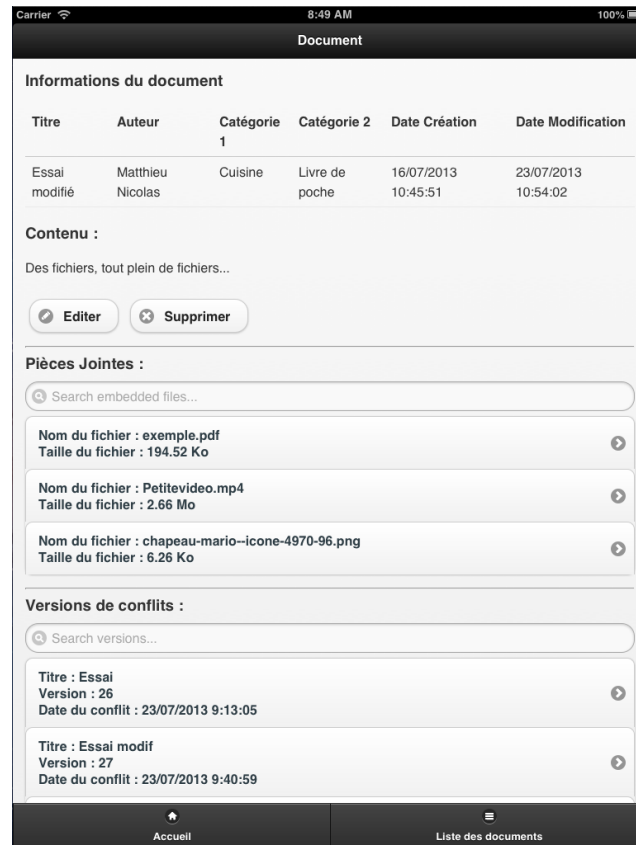


Figure 10 - Capture d'écran de la fiche d'information d'un document

L'utilisateur a alors accès aux fonctions d'édition et de suppression de ce document.

Depuis la liste des documents, nous pouvons aussi accéder à la création de document, et en cochant la checkbox d'un document, stocker localement ses données.

### *b) Stocker des données localement*

Une des nouveautés de l'HTML5 fut l'apparition du LocalStorage et du SessionStorage. Je fus chargé de rechercher si ceux-ci pouvaient être utilisés en tant que solutions viables de stockage de données au sein du navigateur. Ces derniers permettent de stocker, l'espace d'une session dans le cas du SessionStorage ou de façon persistante pour le LocalStorage, 5Mo de données par domaine sous la forme de chaîne de caractères. Il est donc très vite apparu que ces deux espaces de stockage n'étaient pas adaptés à l'utilisation que nous souhaitions en faire et qu'ils correspondaient plutôt à une évolution des Cookies. Cependant il est intéressant de noter que, contrairement aux cookies qui sont transmis à chaque requête au serveur, les données stockées dans ces espaces de stockages ne sont connues et utilisées uniquement par le navigateur.

Nous avons alors étudié les autres solutions de stockage possible au sein du navigateur : SQLite, qui est l'implémentation d'une base de données relationnelle utilisant SQL, et IndexedDB, qui est l'implémentation d'une base de données orientée objet. Cependant, l'application étant destinée à des appareils mobiles sous iOS ou Android, la compatibilité des navigateurs était à prendre en compte :



Figure 11 - Compatibilité des navigateurs

Nous avons donc décidé d'utiliser une base de données SQLite pour stocker les données localement.

### c) Détecter l'absence de connexion Internet

Afin de savoir dans quel mode, connecté ou non-connecté, l'application doit fonctionner, nous devons détecter si l'appareil possède actuellement une connexion internet. Pour cela, nous utilisons la variable JavaScript « navigator.onLine » qui indique si l'on est connecté à internet.

Il nous suffit alors d'adapter nos fonctions, selon la valeur de cette variable, afin de récupérer les données nécessaires depuis le serveur via une requête AJAX ou par le biais d'une requête SQL.

### d) La synchronisation

Comme expliqué précédemment dans le cahier des charges, la présence d'un mode de fonctionnement non-connecté implique la mise en place d'un processus de synchronisation des données. Celui-ci s'effectue en plusieurs étapes :

Tout d'abord, il transmet les nouveaux documents créés localement au serveur afin de les rendre accessibles à tous les utilisateurs.

Il procède ensuite à l'examen des documents mis à jour ou supprimés localement. Pour ces deux cas, des conflits peuvent apparaître : en effet, un document présent sur le serveur peut avoir été modifié par un autre utilisateur depuis notre dernière connexion. Pour détecter ce cas particulier, nous comparons la date de la dernière modification du document au moment où nous embarquons ses données à la date de la dernière modification actuelle du document.

Nous pouvons alors, si elles sont différentes, créer une version de conflit du document en local afin de conserver les changements que nous avons effectués puis récupérer la nouvelle version du document depuis le serveur.

Finalement nous vérifions que les documents embarqués, mais n'ayant subi aucune modification, sont bien à jour et les actualisons dans le cas contraire.

## 2.4 Les limites de l'HTML5

### *a) Le stockage de fichiers*

Une autre nouveauté majeure apportée par la nouvelle version du langage HTML est l'apparition des API File et FileSystem. Celles-ci étendent les possibilités du langage JavaScript en nous permettant de manipuler et lire des fichiers, mais aussi de créer un espace de stockage au sein du navigateur. Cependant, l'API FileSystem n'est actuellement implémentée que par le navigateur Chrome et, malgré l'apparition de plusieurs bibliothèques telles que Filer.js étendant cette implémentation à d'autres navigateurs, ces fonctionnalités restent inaccessibles au sein des navigateurs mobiles dont notamment iOS Safari.

Afin de stocker les fichiers sur l'appareil, nous avons alors dû développer une application native. Celle-ci, paramétrée de façon à pouvoir être lancée par le biais d'une URL, est alors appelée depuis le site afin de procéder au téléchargement des pièces jointes lors du stockage d'un document en local.

Cependant, cette méthode possède plusieurs points négatifs : tout d'abord, elle nécessite le développement d'une application native pour chaque environnement souhaité. L'utilisateur doit ensuite procéder à l'installation de cette application s'il souhaite profiter de toutes les fonctionnalités du site. De plus, l'appel de l'application native a pour effet de déclencher une transition : le navigateur donne la main à l'application qui passe en 1<sup>er</sup> plan et inversement lorsque l'application a terminé son traitement. Cette transition s'avère particulièrement lourde et gênante au cours de l'utilisation de l'application. Finalement, le navigateur ne dispose d'aucun moyen de déterminer quand est-ce que l'application a terminé son traitement et lui a rendu la main.

### *b) L'affichage de fichiers*

Le stockage des pièces jointes étant assuré par une application native, nous ne pouvions gérer leur affichage que de façon native. Nous avons donc ajouté à notre application les méthodes permettant d'afficher les pièces jointes grâce aux fonctions natives de l'appareil.



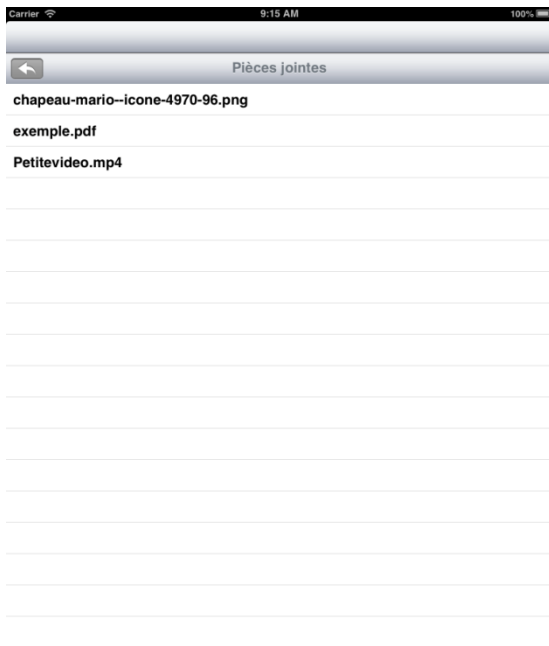


Figure 12 - Capture d'écran de la liste des pièces jointes d'un document

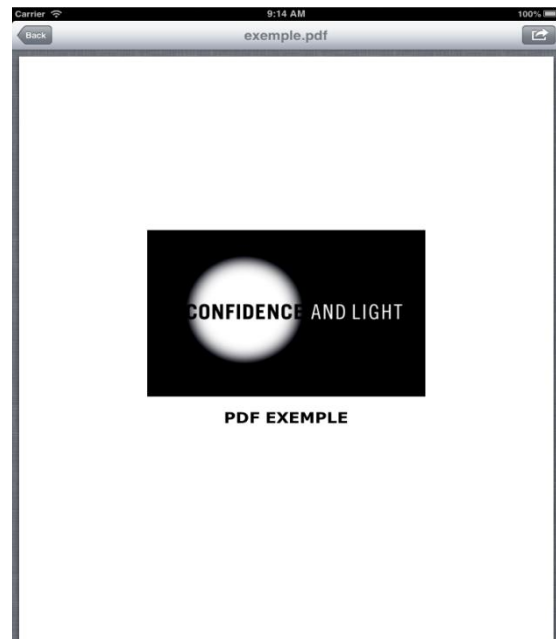


Figure 13 - Capture d'écran d'une pièce jointe affichée depuis l'application

Néanmoins cette solution partage les défauts décrits ci-dessus de l'application de stockage de fichier.

### c) Le cache

Afin de fonctionner en mode non-connecté, le site web nécessite d'être préalablement mis en cache sur l'appareil. Cependant, la mise en cache du site nécessite que l'utilisateur parcourt l'ensemble des pages en mode connecté. Il s'agit donc d'une opération longue et contraignante.

De plus, la mise en cache apparaît comme une source probable de bugs lors d'une mise à jour du site. En effet, nous avons pu constater au cours du développement que le navigateur préférait afficher fréquemment la version du site provenant du cache plutôt que la nouvelle version hébergée sur le serveur.

## 2.5 Bilan

En raison des limites évoquées précédemment, nous avons conclu que l'HTML5 est une solution viable mais qui peut s'avérer lourde d'utilisation pour le client si l'application doit permettre de stocker et d'afficher des fichiers.

Il s'agit toutefois d'une solution dont l'évolution est à suivre, les navigateurs des systèmes d'exploitation mobile mettant régulièrement à jour leur support des nouvelles API.

### 3. L'approche hybride

#### 3.1 Pourquoi une application hybride ?

##### *a) Qu'est-ce qu'une application hybride ?*

Une application hybride est une application principalement développée en HTML5, JavaScript et CSS. Elle est ensuite compilée à l'aide d'un outil tel que Worklight ou Phonegap permettant de choisir pour quel(s) environnement(s) nous souhaitons la déployer. Ceci nous fournit une application au fonctionnement identique à une application native, mais se contentant d'exécuter du web.

##### *b) Fonctions natives*

Une application hybride, étant installée comme une application standard, a accès aux fonctions natives de l'appareil. De ce fait, nous pouvons appeler le système de fichiers natif pour notre application mais nous pouvons aussi avoir accès au GPS ou à la caméra de l'appareil par exemple.

##### *c) Mise en cache des pages*

L'application hybride prenant la forme d'une archive contenant l'ensemble des fichiers HTML, JavaScript et CSS, plus aucune mise en cache des pages web n'est requise : les fichiers nécessaires sont directement présents sur l'appareil.

#### 3.2 Présentation des outils

Afin de développer une application hybride, nous avons utilisé l'outil IBM Worklight. Il s'agit d'un plug-in d'Eclipse nous permettant de concevoir des applications web en HTML, JavaScript et CSS et de les déployer sous forme d'applications hybrides pour différents systèmes d'exploitation mobiles comme iOS et Android, mais aussi Windows Phone 8 ou BlackBerry... . Cet outil met à disposition de nombreuses méthodes JavaScript pour faciliter le développement multiplateformes, provenant notamment de la librairie Cordova.

### a) Cordova



Figure 14 - Logo de Cordova

Cette librairie, développée par Apache et intégrée dans Worklight, nous permet de développer des applications hybrides utilisant directement les fonctions natives de l'appareil via du code JavaScript. Ceci nous permet donc de diminuer la portion de code spécifique à l'environnement de déploiement en instaurant des méthodes communes à l'ensemble des systèmes d'exploitation.

### b) Worklight Server

Worklight Server est requis afin de profiter de plusieurs fonctionnalités de Worklight.

La 1<sup>ère</sup> d'entre elles est l'utilisation d'Adapteurs. Les adaptateurs servent de passerelle pour interroger en JavaScript un serveur distant et récupérer des données via HTTP, SQL DB, JMS ou IBM Cast Iron. Les données sont ensuite retournées au format JSON à l'application.

```
function requestAuth()
{
    WL.Logger.debug("On passe bien par là");
    var input =
    {
        method : 'post',
        returnedContentType : 'json',
        path : 'names.nsf?Login',
        parameters:
        {
            Username: 'matthieu nicolas',
            Password: 'matthieu2013',
            Redirect: '/'
        }
    };
    return WL.Server.invokeHttp(input);
}

function getDocuments()
{
    var input =
    {
        method : 'get',
        returnedContentType : 'JSON',
        path : 'MCL/GestionDocs.nsf/va_$Documents?readviewentries&outputformat=JSON'
    };
    return WL.Server.invokeHttp(input);
}
```

Figure 15 - Exemple d'Adapteur

Worklight Server nous permet aussi de mettre à jour aisément une application hybride : en effet, il suffit de déployer la nouvelle version de l'application sur le serveur. Chaque client se connectant au serveur dans le cadre de l'utilisation de son application sera alors notifié qu'une nouvelle version est disponible et pourra la télécharger directement, sans devoir passer par le biais du Store correspondant à son environnement.

Une version locale de Worklight Server est installée et mise à disposition dans Eclipse lors de l'installation du plug-in Worklight. Ce serveur local nous permet de tester notre application grâce au simulateur Worklight.

Cependant, Worklight Server nécessite une licence d'utilisation. Celle-ci, coûteuse, nuit grandement à la compétitivité financière de cette approche. Je fus donc chargé de vérifier la nécessité de Worklight Server.

### c) *Simulateur Worklight*

Comme expliqué précédemment, Worklight met à disposition un simulateur afin de tester son application. Celui-ci, s'exécutant dans un navigateur web, nous permet de choisir l'environnement pour lequel nous souhaitons vérifier le bon fonctionnement de notre application.

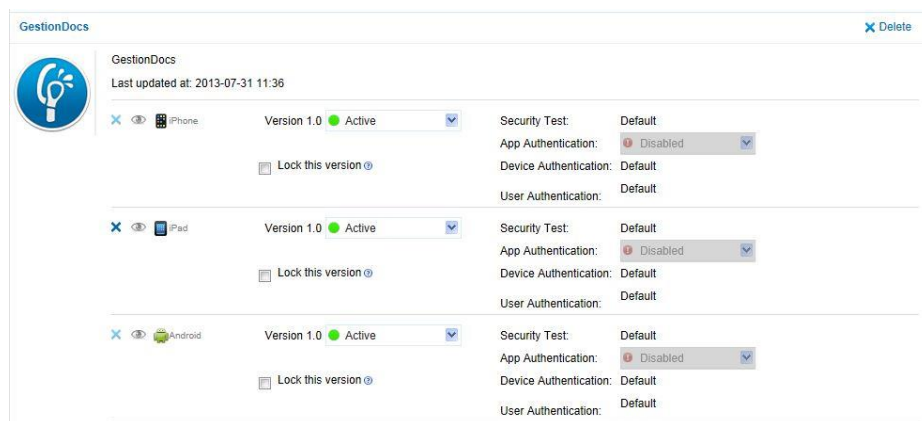


Figure 16 - Console Worklight

Il intègre Cordova, nous permettant ainsi de simuler des interactions de l'utilisateur avec les fonctionnalités natives telles que la prise d'une photo avec l'appareil, mais aussi de consulter l'arborescence du système de fichiers.

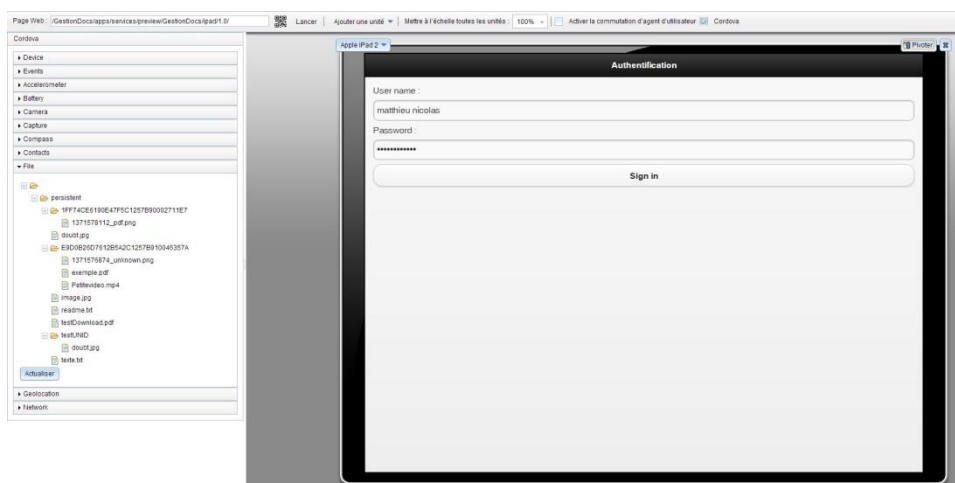


Figure 17 - Simulateur Worklight

### 3.3 L'application hybride

#### a) Chargement de données à partir d'un domaine distant

Afin d'insérer les données dans les pages web de l'application, qui d'origine ne contiennent que la structure de la page, nous utilisons des requêtes AJAX pour interroger le serveur. Cependant, l'application s'exécutant en local, les requêtes sont effectuées sur un domaine distant. Ceci est une opération interdite en raison de la « Cross-Domain Policy ».

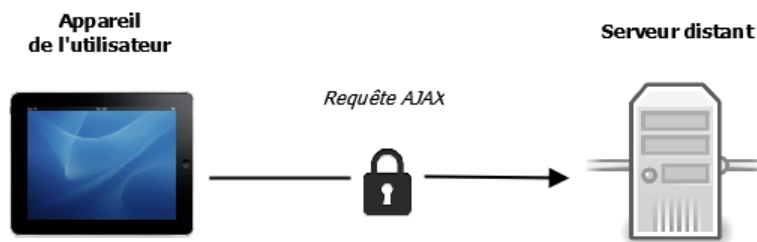


Figure 18 - Requête AJAX bloquée en raison de la Cross-Domain Policy

Pour contourner cette contrainte, la solution proposée par Worklight est l'utilisation d'un adaptateur. En utilisant un HTTP Adaptateur, nous pouvons interroger le serveur par le biais de l'adaptateur qui nous retourne les données au format JSON.

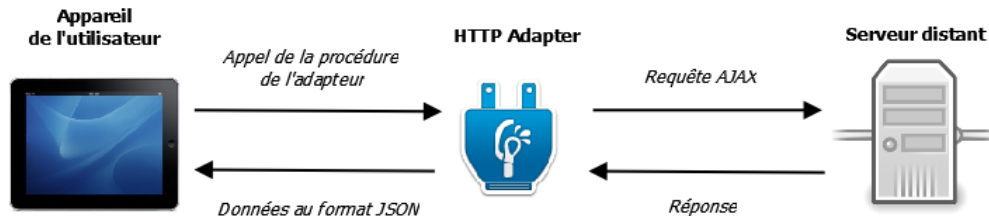


Figure 19 - Utilisation d'un adaptateur pour récupérer les données depuis un domaine distant

Cependant, télécharger une pièce jointe via l'API de Cordova s'avère problématique si nous utilisons un adaptateur. En effet, seul l'adaptateur communique avec le serveur. De ce fait, lui seul est authentifié et autorisé à accéder aux fichiers. Il faut donc dans ce cas procéder à l'encodage en base 64 du fichier sur le serveur, ce qui entraîne une augmentation conséquente de la taille du fichier. Une fois encodé, il faut transmettre les données du fichier à l'adaptateur, qui va ensuite les faire parvenir à l'application. L'application doit finalement décoder les données pour recomposer le fichier initial et le stocker.

Toutefois, en déployant et testant l'application sur l'environnement cible, il nous est apparu que la sécurité empêchant le chargement de données depuis un système distant n'est pas activée dans le navigateur utilisé au sein d'une application hybride. Il s'est donc avéré que le simulateur, s'exécutant dans un navigateur Web, souffre des limitations inhérentes à cet environnement et qu'il est préférable de tester son application directement sous l'environnement cible.

### *b) Utilisation de pages natives*

Worklight implémente plusieurs méthodes permettant l'utilisation de pages natives au sein de l'application hybride. Celles-ci nous permettent contrôler l'appel d'une page native et le retour, les deux pouvant transmettre des données entre les parties hybride et native. Ces appels et retours se font sans transition, il s'agit concrètement d'un simple changement de vue ou d'activité selon l'environnement, et de façon transparente pour l'utilisateur.

## **3.4 Bilan**

L'approche hybride possède donc de nombreux avantages : elle nous permet de créer des applications à base d'HTML, JavaScript et CSS capables, en addition aux fonctionnalités inhérentes à ces langages, d'appeler des morceaux de code natif afin de pallier aux limites de ces langages de programmation.

De plus, l'application étant installée sur l'appareil, nous ne rencontrons plus aucun souci de mise en cache.

Cependant, cette solution ne sera jamais aussi optimisée qu'une application native complète. Elle souffre aussi de problèmes liés à l'outil Worklight directement. En effet, bien que puissant, celui-ci présente plusieurs défauts : tout d'abord, il nécessite de recompiler et de redéployer l'application à chaque modification pour chaque système d'exploitation mobile choisi. Cette opération, bien que rendu très simple, s'avère lente. Son simulateur, quant à lui, souffre principalement du fait qu'il ne représente pas fidèlement les fonctionnalités de l'environnement simulé. C'est pourquoi il peut être préférable d'exporter rapidement son projet sous l'environnement de développement du système d'exploitation principalement visé.

Il s'agit toutefois d'un logiciel récent, particulièrement puissant et qui une fois pris en main permet le développement rapide d'applications multiplateformes. Cette solution est donc viable et permet de répondre aux besoins spécifiés dans le cahier des charges.

## 4. Bilan des approches

### 4.1 Comparatif des approches possibles

Critère / Approche	Natif	HTML5	Hybride
Langage(s) utilisés	Java pour Android, Objective-C pour iOS	HTML, JavaScript et CSS	HTML, JavaScript, CSS et possibilité d'ajouter du code natif
Multiplateformes ?	Non	Oui	Oui
Taille de la base de données locale	Illimité	50Mo	Illimité
Accès aux fonctionnalités natives de l'appareil	Oui	Non	Oui
Force	Optimisée pour un environnement	Solution simple multiplateformes	Allie les atouts de l'HTML avec les fonctionnalités natives
Faiblesse	Limitée à un environnement	Limitée par les fonctionnalités supportées par les navigateurs	Environnement de développement contraignant

### 4.2 Améliorations à apporter

#### a) Sécurité

Actuellement, l'application invite l'utilisateur à s'authentifier si une connexion internet est établie. Cependant, en mode hors-connecté, aucune sécurité empêche d'accéder au contenu stocké en local et aux différentes fonctionnalités de l'application, faute d'accès au serveur Domino qui gère les droits d'accès.

De plus, sous iOS, les différents fichiers téléchargés sont stockés dans un répertoire propre à l'application et sont inaccessibles à l'utilisateur autrement qu'en utilisant l'application. A l'inverse, sous Android, les fichiers sont stockés dans un répertoire qui est accessible à l'aide d'un explorateur de fichiers.

De ce fait, des améliorations au niveau de la sécurité et de la confidentialité des données peuvent être envisagées telle que paramétrer un mot de passe pour le mode hors-ligne.

*b) Généralisation des méthodes principales*

Plusieurs méthodes de cette application de test sont réutilisables dans des applications similaires, telle que celle permettant de former, depuis les données transmises par le serveur ou conservées localement, une liste d'éléments. Une modification de leur prototype afin de les rendre plus flexibles permettrait de faciliter et d'accélérer le développement de ces nouvelles applications.



## **Conclusion :**

Ce stage fut l'occasion de mettre en application les méthodes de travail et les connaissances acquises au cours de ma formation mais aussi d'accroître mes compétences. En effet, j'ai pu découvrir d'autres environnements de développement et outils tels qu'IBM Domino et Worklight. Ce stage m'a de plus permis de développer mes connaissances en Web en découvrant le framework jQuery, mais aussi en développement Android et iOS.

Travailler au sein de thém-is fut aussi une expérience enrichissante, me permettant de découvrir le monde du travail ainsi que le fonctionnement d'une entreprise à taille humaine.

Je suis donc pleinement satisfait de cette expérience, ce stage ayant répondu à mes attentes aussi bien sur le plan technique que humain, et j'en garderai un excellent souvenir.

## Annexes :

### 1. Bibliographie

- **them-is.fr** : site Internet de Thém-is.
- **ibm.com** : portail IBM permettant notamment d'accéder aux documentations sur leurs outils tel que Worklight et aux forums de discussion consacrés aux problèmes rencontrés.
- **jquerymobile.com** : site de jQuery Mobile présentant comment utiliser le framework et illustrant ses résultats.
- **stackoverflow.com** : forum pour développeurs couvrant de nombreux sujets.
- **siteduzero.com** : plateforme regroupant de nombreux tutoriels sur le développement informatique.
- **raywenderlich.com** : site présentant des tutoriels sur le développement d'application iOS.

### 2. Glossaire

#### 2.1 OS

Acronyme anglais désignant « *Operating System* », qui se traduit en français par « *Système d'exploitation* ». Il s'agit de l'ensemble des programmes gérant les ressources du système informatique et formant l'environnement dans lequel s'exécutent les autres programmes. Les systèmes d'exploitation les plus connus sont Windows pour les ordinateurs, et Android et iOS pour les smartphones et tablettes.

#### 2.2 API

Acronyme anglais pour « *Application Programming Interface* » se traduisant par « *Interface de programmation* ». Une API est un ensemble de classes et méthodes permettant à un utilisateur d'intégrer les fonctionnalités d'un programme tiers au sein de son propre programme ou d'accéder à ses données.

Dans le cadre du développement WEB, le W3C, l'organisme de normalisation des technologies WEB, définit par le biais d'API les standards auxquels doivent se conformer les implémentations des différentes fonctionnalités. Par exemple, SQLite est l'implémentation correspondant à l'API Web SQL Database proposée par le W3C.

## 2.3 Framework

Un framework est un ensemble de composants ayant pour but de définir l'architecture d'un logiciel et de mettre à disposition de l'utilisation diverses fonctionnalités supplémentaires.

## 2.4 Requêtes AJAX

Une requête AJAX (acronyme de « *Asynchronous JavaScript and XML* ») permet d'interroger, en tâche de fond, une page WEB en JavaScript. Ceci permet d'échanger des données entre le serveur WEB et le navigateur de façon dynamique, sans nécessiter le chargement d'une nouvelle page.

## 2.5 JSON

« *JavaScript Object Notation* » est un format de données textuelles permettant de représenter des objets de façon générique et structurée.

```
{
  "unid": "F7364B8CBAEB0064C1257BC20032C4E1",
  "titre": "Document # 1 - Vente",
  "piecesJointes": [
    {
      "source": "fd_hole.png",
      "nom": "fd_hole.png",
      "taille": 150,
      "dateModification": "01/08/2013 12:04:15"
    },
    {
      "source": "fd_hole.002.png",
      "nom": "fd_hole.002.png",
      "taille": 150,
      "dateModification": "01/08/2013 12:04:15"
    }
  ]
}
```

Figure 20 - Exemple d'objet JSON

## 2.6. Base64

Base64 est un codage des données utilisant 64 caractères disponibles sur la majorité des systèmes. Cet encodage est principalement utilisé pour transmettre des messages mais aussi pour envoyer sous forme textuelle le contenu d'un fichier. Il possède néanmoins le défaut d'augmenter la taille du contenu d'environ 50%.


### 3. Comparatif des solutions de stockage locales

	JSONStore	Encrypted Cache	Local Storage	Indexed DB	Cordova Storage	Cordova File
Android Support	Yes	Yes	Yes	Yes	Yes	Yes
iOS Support	Yes	Yes	Yes	Yes	Yes	Yes
Web	Dev. Only (3)	Yes	Yes	Yes	-	-
Data Encryption (1)	Yes	Yes	-	-	-	-
Maximum Storage	Available space	~ 5 MB	~ 5 MB	> 5 MB	Available space	Available space
Reliable Storage (2)	Yes	-	-	-	Yes	Yes
Adapter Integration (1)	Yes	-	-	-	-	-
Multi User Support (1)	Yes	-	-	-	-	-
Indexing	Yes	-	-	Yes	Yes	-
Type of Storage	JSON Documents	Key – value pairs	Key – value pairs	JSON Documents	Relational (SQL)	Strings

- > (1): These features are further described in the module **JSONStore – Common JSONStore usage**.
- > (2): **Reliable Storage** means that your data is not deleted unless the application is removed from the device or one of the methods that removes data is called.
- > (3): **Dev. Only** means that it is designed only for development. There are no security features and a ~5 MB storage space limit.

Figure 21 - Comparaison de solutions de stockage locales

### 4. Fiche récapitulative des bases de l'Objective-C



## Objective-C Cheat Sheet and Quick Reference

#### Assumptions

This cheat sheet assumes you have enabled Automatic Reference Counting (ARC) for your project. Enjoy!

#### Class Header (.h)

```
#import "AnyHeaderFile.h"

@interface ClassName : SuperClass {
    // declare instance variables
    // (optional)
}
// define properties
// define methods (including any
// custom initializers)
@end
```

#### Class Implementation (.m)

```
#import "YourClassName.h"

@implementation ClassName
// synthesize properties (optional in
// Xcode 4.4+)
// implement methods (including any
// custom initializers, and dealloc)
@end
```

#### Defining Methods

```
- (anytype) doIt: {
- (anytype) doItWith: (anytype)a;
- (anytype) doItWith: (anytype)a
andB: (anytype)b;
}
```

#### Implementing Methods

```
- (anytype) doItWith: (anytype)a
andB: (anytype)b {
    // Do something with a and b...
    return retVal;
}
```

#### Creating a Class Instance

```
ClassName * myClass =
    [[ClassName alloc] init];
```

#### Calling a Method

```
[myClass doIt];
[myClass doItWith:a];
[myClass doItWith:a andB:b];
```

#### Defining Properties

```
@property (attribute1, attribute2)
propertyName;
```

<b>strong</b>	Adds ref to keep object alive
<b>weak</b>	Object can disappear, become nil
<b>assign</b>	Normal assign, no reference
<b>copy</b>	Make copy on assign
<b>nonatomic</b>	Make not threadsafe, increase perf
<b>readonly</b>	Create getter/setter (default)
<b>readonly</b>	Create just getter

#### Synthesizing Properties

```
// Optional in Xcode 4.4+
@synthesize propertyName;
@synthesize propertyName =
_myInstanceVariable;
```

#### Using Properties

```
[myClass setPropertyName:a];
myClass.propertyName = a; // alternative

a = [myClass propertyName];
a = test.propertyName; // alternative
```

#### Declaring Variables

```
anytype myVariable;
```

<b>int</b>	1, 2, 500, 10000
<b>float</b>	1.5, 3.14, 578.234
<b>double</b>	
<b>BOOL</b>	YES, NO, TRUE, FALSE
<b>NSString</b>	NSString, NSArray, etc.
<b>id</b>	Can hold ref to any object

#### Custom\_INITIALIZER Example

```
- (id)initWithParam:(anytype)param {
    if ((self = [super init])) {
        self.propertyName = param;
    }
    return self;
}
```

#### NSString Quick Examples

```
NSString *personOne = @"Ray";
NSString *personTwo = @"Shawn";
NSString *combinedString =
[NSString stringWithFormat:
@"%d: Hello, %d!",
personOne, personTwo];
NSLog(@"%d", combinedString);
NSString *tipString = @"$24.99";
float tipFloat = [tipString floatValue];
```

#### NSArray Quick Examples

```
NSMutableArray *array =
[NSMutableArray arrayWithObjects:
personOne, personTwo, nil];
[array addObject:@"Waldo"];
NSLog(@"%d items!", array.count);
for (NSString *person in array) {
    NSLog(@"%d: %d", person);
}
// Xcode 4.4+ alt: array[2]
NSString *waldo =
[array objectAtIndex:2];
```

Source: raywenderlich.com. Visit for more iOS resources and tutorials!

Version 1.2. Copyright 2012 Ray Wenderlich. All rights reserved.

Figure 22 - Bases de l'Objective-C