



UNIVERSITÉ MONTPELLIER II
MASTER INFORMATIQUE SPÉCIALITÉ AIGLE

17 mai 2013
Rapport de TER

Framework pour applications mobiles

Phares CHAKOUR
Alexandre MATHIEU
Bastien VANDERCHMITT

ENCADRANT : CHRISTOPHE DONY

Remerciement

Nous souhaitons remercier M. Christophe DONY notre tuteur qui a bien voulu nous encadrer pour ce travail d'études et de recherches.

Table des matières

1	Introduction	7
1.1	Présentation générale	7
1.2	Groupe de travail	7
1.3	Sujet et Problématique	7
2	État de l’art	9
2.1	Critères de classification	9
2.2	Tableau comparatif	12
2.3	Frameworks existants	12
2.3.1	Titanium	13
2.3.2	PhoneGap	14
2.3.3	jQueryMobile	15
2.3.4	Programmation du compteur	16
3	Proposition	18
3.1	Analyse des critères	18
3.2	Cahier des charges	19
3.2.1	Présentation générale du problème	19
3.2.2	Expression fonctionnelle du besoin	20
3.2.3	Cadre de réponse	23
4	Conception	24
4.1	Analyse	24
4.1.1	Première approche	24
4.1.2	Les Plugins	27
4.1.3	Bundles	29
4.1.4	Utilisateurs	31
4.2	Architecture	32
4.2.1	Général	32
4.2.2	Structure des plugins	33
4.2.3	Bundle	35

5	Réalisation	43
5.1	Organisation	43
5.2	Développement	43
5.2.1	Android	43
5.2.2	Firefox OS	44
5.2.3	Windows Phone	45
6	Résultats et Discussion	46
6.1	Gestionnaire de version	46
6.2	Développement d'une application mobile : <i>storage</i>	46
6.3	Objectifs	46
6.4	Viabilité du framework	48
6.5	Windows Phone	48
7	Conclusion	49
A	Liste des technologies utilisées dans le projet	51
B	Code source du compteur	52
C	Diagramme de Gantt	53
D	Code source de l'application <i>storage</i>	55
D.1	bundle.js	56
D.2	connection.js	56
D.3	input.js	56
D.4	route.js	56
D.5	IndexController.js	56
D.6	Note.js	57
D.7	index.tpl	57
D.8	add.tpl	57

Table des figures

2.1	Schématisation d'une application hybride	11
2.2	Principe général d'organistaion de Titanium	13
2.3	Implémentation du programme compteur avec différent framework	16
4.1	Génération d'une application Android qui inclue les fichiers HTM- L/Javascript fournis par l'utilisateur	26
4.2	Diagramme de séquence représentant la construction de la vue native durant l'exécution d'une application	27
4.3	Communication entre un plugin et le framework durant l'exécution d'une application mobile.	29
4.4	The Component-Oriented Programming Duality [LFH10]	30
4.5	Exemple de composite	31
4.6	Cas d'utilisations des outils de génération	32
4.7	Classe <i>Plugin</i> que le développeur de plugin doit étendre, fournie dans l'API de notre framework	34
4.8	Schématisation du MVC	35
4.9	Principe général du client/serveur	36
4.10	Mécanisme du moteur de template	37
4.11	La vue appelle le contrôleur	38
4.12	Fonctionnement du routage	39
4.13	Diagramme de séquence : Le chargement du Bundle initial	41
4.14	Diagramme de séquence : Le chargement d'une vue	42
5.1	Emulateur Firefox OS utilisé pour le développement	44

Résumé

Une application mobile est un programme informatique adapté aux téléphones portables et répondant à un besoin particulier. Une plateforme mobile est composé d'une multitude d'applications s'assemblant entre elles pour former un système complet et adapté à l'utilisateur. Pour une grande partie d'entre elles (Android, Firefox OS, Windows Phone, etc.), un kit de développement est fourni spécialement pour les développeurs de tous horizons, afin de combler les supermarchés de l'application mobile (Google Play, Firefox MarketPlace, App Store). A travers ce document, nous proposons une solution originale qui permet de réaliser des applications mobiles, offrant un gain de temps pour le développement et une bonne maintenabilité du code source. Cette solution se traduit par un framework qui permet à partir d'un même code source, de générer des exécutables issue de différentes plateformes mobile.

Abstract

A mobile application is a computer program adapted to mobile phones and responding to a particular need. A Mobile platform is a multitude of applications which fit together to form a complete system adapted to the user. For a large part of them (Android Firefox OS, WindowsPhone, etc..), a development kit is specially provided for developers of all backgrounds in order to fill mobile application stores (Google Play, Firefox MarketPlace, App Store). Through this paper, we propose a solution which aim to reduce the time of development and ensure good maintainability of the source code. This solution is summarized in a framework.

Chapitre 1

Introduction

1.1 Présentation générale

Dans le cadre du programme de Master 1 Informatique de la Faculté des Sciences de Montpellier II, il nous est demandé de réaliser en groupe un TER (Travail d'Études et de Recherche).

Cet exercice consiste à choisir un sujet, proposé par l'équipe enseignante ou bien à l'initiative du groupe d'étudiants lui-même, à caractère théorique et/ou pratique. Le but de ce TER est d'étudier le sujet afin d'en tirer une expérience valorisante pour la poursuite d'études ou l'insertion professionnelle. Le sujet doit également avoir des vertus pédagogiques en rapport avec les enseignements en informatique pourvus à la Faculté mais peut néanmoins aborder des thèmes nouveaux.

1.2 Groupe de travail

Le groupe de travail est composé de Mathieu Alexandre, Vanderchmitt Bastien et Chakour Phares.

Ce projet est encadré par Christophe Dony, professeur à l'université Montpellier II et chercheur au Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier (LIRMM).

1.3 Sujet et Problématique

Aujourd'hui, le développement des smartphones est en pleine expansion. Divers systèmes d'exploitation font leur apparition avec le désir d'occuper la plus grande place sur le marché.

Cette diversité qui ne cesse de s'accroître dans le monde du mobile pose problème pour le développement d'applications. Actuellement il est impossible de développer

une même application avec un même code source, pour différentes plateformes mobiles, en se basant sur les outils qu'offre ces dernières. Le développeur doit réaliser l'application autant de fois qu'il y a de plateforme afin que celle-ci soit portable¹.

Il est donc intéressant de proposer une solution qui permet au développeur de mettre en place un unique code pour une application sans pour autant délaissé la portabilité. Cette solution peut se traduire par un framework qui propose des outils pour générer des exécutables sur différentes plateformes à partir d'un même code source.

Nous analyserons en premier lieu les systèmes existants répondants à cette problématique afin de constater ce qui les caractérise, ainsi que leurs avantages et inconvénients. Nous déduirons de cette première étude un schéma de réalisation qui aboutira à la création de notre propre solution. Enfin, nous ferons le point sur les différentes phases abordées dans ce projet.

1. Un programme informatique est dit portable lorsque qu'il fonctionne correctement dans différents environnements d'exécution.

Chapitre 2

État de l’art

Tout d’abord nous introduisons la notion de framework :

En programmation informatique, un framework est un kit de composants logiciels structurels, qui sert à créer les fondations ainsi que les grandes lignes de tout ou d’une partie d’un logiciel.

(Wikipédia)

Avant de se lancer dans la réflexion du projet, il est important de faire des recherches pour connaître les différents travaux existants, répondant totalement ou partiellement à notre problématique. Cette recherche nous permet de définir les types de framework que nous pouvons actuellement trouver pour développer une application qui s’exécute sur plusieurs plateformes mobile. Cette vue d’ensemble sur les frameworks existants nous amène à définir des critères permettant de classer ces derniers.

2.1 Critères de classification

L’ensemble des frameworks, résultant de notre recherche, peuvent être classés selon plusieurs critères : la licence, le langage de programmation, le type d’application produite par le framework et les plateformes supportées. Ces critères de classification sont ceux qui déterminent, en grande partie, le choix d’un framework pour le développeur. Parmi cette liste, nous remarquons un critère particulier, ”Le type d’application produite par le framework”. En effet, à ce jour, dans le monde du mobile, ce critère prend une place très importante. Il est donc nécessaire de bien comprendre ce qui se cache derrière celui-ci. Ainsi, nous avons l’obligation de nous attarder sur ce sujet, en déterminant les types d’applications mobile qui existent actuellement. Il découle de nos recherches que les développeurs disposent de trois façons pour créer une application : native, web et hybride.

Les applications natives sont développées avec le langage de programmation le plus adapté à l'OS¹. C'est à dire, le langage préconisé par les plateformes mobile, qui fournissent une API (Application Programming Interface) permettant d'utiliser pleinement les ressources du téléphone (caméra, GPS...) ainsi qu'une vitesse d'exécution très rapide. Malheureusement, avec cette méthode de développement, l'application peut être exécutée uniquement sur la plateforme qui a fourni l'API. Par conséquent, le développeur a la charge de développer son application sur les différents systèmes d'exploitation mobile. Cela implique des coûts de développement qui sont difficile à supporter par les petites entreprises ou les développeurs freelance².

D'un point de vue de leur distribution, il est intéressant de noter que les applications natives ont la possibilité d'être publiées dans les "app stores". Ils ont pour but de distribuer l'application via internet et sont accessibles à partir du mobile via une application dédiée à cet effet. Même si les développeurs doivent se plier à des règles précises et subir le dictat des "app stores", cela reste un point fort dans la distribution à grande échelle de l'application, un point fort que les applications web ne possèdent pas.

Les applications web sont, en simplifiant un peu, globalement développées en HTML5. Elles sont donc parfaitement compatibles avec n'importe quelles plateformes mobile car elles s'exécutent via un navigateur web. De ce fait, elles nécessitent impérativement une connexion web et ne peuvent pas exploiter pleinement la partie matérielle des smartphones (caméra, GPS...). En terme de vitesse d'exécution, les applications web sont moins performantes mais restent tout de même rapides. Nous constatons donc que les applications web permettent de palier au problème de la portabilité mais sont incapables d'utiliser les ressources des téléphones. Ces applications web sont donc utiles pour créer des programmes qui ne nécessitent pas d'utiliser la partie matérielle du mobile. Pourtant, il existe un type d'application offrant la portabilité ainsi que la possibilité d'accéder aux ressources du smartphone : les applications hybrides.

Une application hybride est une application pour mobile qui combine des éléments HTML5 sous forme de web application mobile et des éléments d'une application native permettant d'utiliser les fonctionnalités natives des smartphones. Une application hybride peut également être distribuée via les plateformes de téléversement d'applications (App Store, Android Market, etc.). Le principe de l'application hybride permet de réduire les coûts et délais de développement nécessaires pour proposer plusieurs applications natives pour les différents systèmes d'exploitation mobile. Pour mieux comprendre nous pouvons analyser la figure 2.1.

1. Operating Sytem qui correspond à la plateforme mobile (Android, iOS, Windows-Phone...).

2. Développeur exerçant sa profession de manière indépendante.

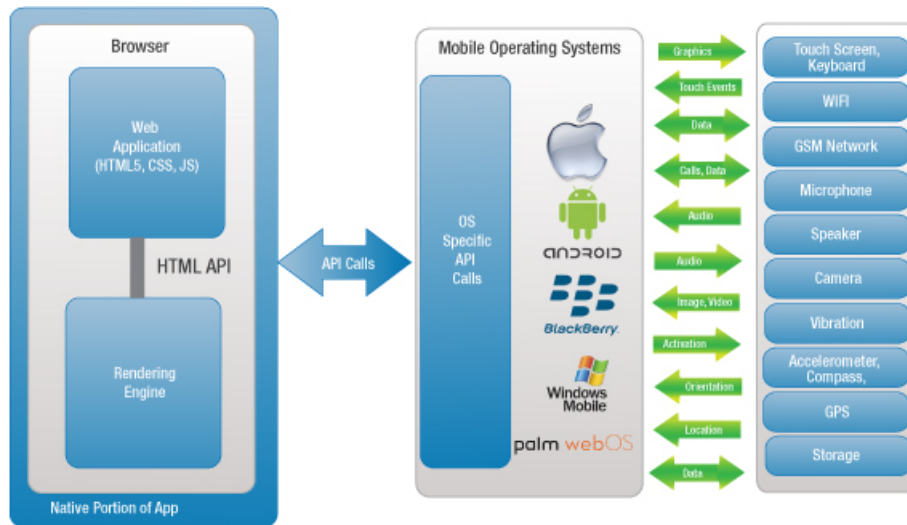


FIGURE 2.1: Schématisation d'une application hybride

<http://www.xrgsystems.com/solution.php?page=hybrid-applications>

Nous pouvons diviser ce schéma en trois parties : l'application, la plateforme et les ressources. Observons dans un premier temps la partie application. Nous constatons que l'application est en fait une application web encapsulée dans une application native. Nous obtenons de cette dernière la possibilité de s'installer sur le mobile, d'être distribué via les magasins d'applications et d'avoir une vitesse d'exécution très rapide. Pour accéder aux ressources du téléphone, l'application a à disposition une API (API Call dans le schéma) qui fournit à l'application un moyen de faire des appels vers la plateforme mobile, la deuxième partie du schéma. Ces appels sont nécessaires pour accéder aux ressources du téléphone car cet accès doit se faire via la plateforme mobile qui est capable de faire abstraction du matériel mobile. Ainsi si l'application veut connaître les coordonnées GPS, elle doit faire un appel à la plateforme mobile qui se charge de faire un accès vers la ressource GPS, troisième partie du schéma, et retourner les informations à l'application.

Grâce à ce principe d'application hybride, nous avons une combinaison avec le natif et le web qui rend l'application portable tout en gardant les accès aux ressources. Nous pouvons résumer l'ensemble de nos propos à l'aide du tableau suivant :

	Accès aux fonctionnalités de l'appareil	Vitesse	Coût de développement	App Store
Native	Complet	Très rapide	Cher	Disponible
Hybride	Complet	Vitesse Native	Raisonnable	Disponible
Web	Partiel	Rapide	Raisonnable	Indisponible

Nous avons donc vu en détails le critère "type d'application produite par le framework" car la compréhension de ce dernier est très importante pour la suite de notre projet. Grâce aux recherches que nous avons effectué, nous disposons à présent d'une vision d'ensemble sur les divers frameworks actuellement disponibles. Il est intéressant de regrouper ces derniers ainsi que les critères de classification afin de construire un tableau comparatif.

2.2 Tableau comparatif

Le nombre de frameworks disponibles est assez important. De ce fait nous avons listé ceux qui semblent apparaître le plus souvent.

Framework	Plateformes supportées	Licence	Langage	Type d'application		
				Native	Hybride	Web
PhoneGap	Android, iOS, Symbian, PalmOS	MIT (Open Source)	HTML, CSS, Javascript		✓	
Titanium	Android, iOS	Apache public v2.0 (Open Source)	HTML, CSS, Javascript	✓		
jQueryMobile	Android, iOS, Blackberry, Windows Phone, Symbian	MIT (Open Source)	HTML, CSS, Javascript			✓
RhoMobile	Android, iOS, Blackberry, Windows Phone, Symbian	MIT (Open Source)	HTML, Ruby	✓		
MoSync	Android, iOS	GPL v2	C/C++, Javascript, Python	✓	✓	
Xamarin	Android, iOS, Windows-Phone	LGPL	C#	✓		

Après avoir déterminé les critères, afin de classer la multitude de framework multi-plateformes que nous avons trouvé, nous allons étudier plus en détail trois d'entre eux.

2.3 Frameworks existants

Le choix des framework à étudier doit être un échantillon représentatif. C'est pourquoi, nous allons choisir trois frameworks, l'un produisant des applications natives, un deuxième pour les hybrides et un dernier pour les applications web.

Respectivement, nous avons donc sélectionné Titanium, PhoneGap et JQuery-Mobile.

2.3.1 Titanium

Titanium est un framework open source permettant de concevoir des applications mobiles natives pour les systèmes Android et iOS. Les applications sont écrites en HTML/Javascript et CSS, autorisant les développeurs web à utiliser au mieux leurs compétences. Mais d'autres langages sont aussi disponible comme le PHP ou encore le Python.

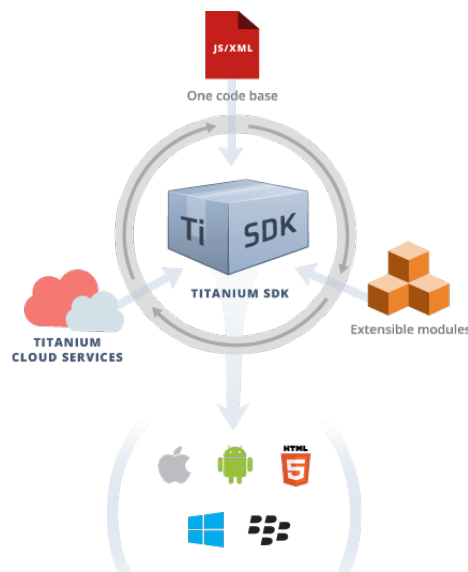


FIGURE 2.2: Principe général d'organistaion de Titanium

<http://www.appcelerator.com/>

Titanium fourni à l'utilisateur un SDK ³, notamment une version d'éclipse personnalisée, permettant de réaliser des applications mobile avec un grande facilitée. Ainsi, le développeur fourni au SDK le code source de son application pour que les outils fournis par Titanium puissent en générer des exécutables.

Les services proposés à l'utilisateur pour le développement d'une application mobile sont d'une grande qualité, et permettent de construire une application mobile en peu de temps. La possibilité d'utiliser plusieurs langages de programmation afin de concevoir une application mobile est un plus. En effet un développeur plus à l'aise avec une autre technologie que le HTML/Javascript trouvera son compte.

3. Kit de développement

Un point important de Titatium, concerne l'étape de génération d'exécutables à partir du code source fourni par l'utilisateur. En effet, ce service est effectué par un serveur particulier, demandant ainsi une connexion internet et un compte développeur. Nous imaginons donc tout les problèmes qui peuvent survenir en délocalisant le générateur d'exécutables, notamment si le serveur est en panne, ou bien encore surchargé de requêtes.

2.3.2 PhoneGap

Phone Gap est un framework open-source de développement mobile conçu au cours de l'année 2005, permettant de créer des applications pour smartphones avec les technologies JavaScript, HTML5 et CSS3. Les applications conçues avec ce système sont définies comme hybrides. Nous pouvons citer le support de onze plateformes mobiles, ce qui fait de ce logiciel un des leaders de la portabilité dans la catégorie framework d'applications hybrides. Ceci permet ainsi à l'utilisateur de toucher un large panel de consommateurs.

La conception d'application mobile nous permet d'aboutir à la génération d'un exécutable propre à chaque plateformes mobiles (principe de l'application hybride). Ainsi contrairement à une application de type web (site internet), l'utilisateur a la possibilité de pouvoir partager son application dans les *stores*⁴. Ces derniers sont un précieux atout pour tout développeur, car cela permet d'augmenter la visibilité d'une application mobile pour les utilisateurs de smartphones. Un des points faibles de Phone Gap intervient lors de la phase de développement de l'application mobile. En effet, nous pouvons distinguer trois étapes majeures dans la création d'une application avec ce framework :

1. Le développeur, avec l'environnement de développement de son choix, en respectant les normes de la plateforme mobile, commence par créer une application native.
2. Le développeur incorpore ensuite à cette application native le framework Phone Gap (ensemble de répertoires et de librairies).
3. Le développeur incorpore au framework Phone Gap l'ensemble de ses sources (fichiers HTML/Javascript et CSS).

Lorsque l'utilisateur veut étendre son application à une nouvelle plateforme mobile, il doit donc télécharger et installer l'environnement de développement adéquat, puis créer un début d'application native avant de pouvoir réellement en générer un exécutable. Ceci impose donc de posséder un minimum de connaissances dans le développement d'applications de la plateforme mobile en question.

4. Magasins d'application, parmi les plus célèbres, nous pouvons citer le Google Play Store (Android), ou bien encore le App Store (iOS).

2.3.3 jQueryMobile

jQueryMobile est une extension du célèbre framework javascript jQuery. Cette extension place le framework sur le marché de l'application mobile. Il répond à la fameuse problématique à laquelle doivent faire face tous les développeurs web : "Comment faire une version adaptée pour les smartphones de mon site web ?". Et c'est à ce moment là que jQueryMobile devient l'outil imparable pour créer un site web adapté à la plupart des mobiles. Il est vrai qu'il est possible de créer une version adaptée aux smartphones en utilisant simplement le CSS⁵. Il est donc légitime de se demander l'intérêt d'utiliser un tel framework.

Un développeur web peut donc créer uniquement avec le CSS une version adapté pour mobile. Cela s'appelle le responsive design. Avec jqueryMobile, le responsive design est automatique. Lorsque que nous programmons avec ce framework, le design s'adapte selon la taille des écrans sans s'en préoccuper. Mais l'aspect visuel qui s'agence selon les périphériques n'est pas le seul avantages présenté par jqueryMobile. Un des avantages le plus attractif est l'ajout d'événements par rapport au traditionnel framework jQuery. En effet, les événements standards que nous pouvons retrouver dans ce dernier ne peuvent pas forcément s'appliquer sur les périphériques mobiles. Par exemple, le "click" couramment utilisé pour les site web est remplacé par "tap" qui correspond à "un événement tactile rapide, complet". Ainsi nous pouvons bénéficier d'événements beaucoup plus appropriés à une interface mobile.

Nous constatons qu'il existe une disparité entre un site dédié pour un mobile et un autre pour un ordinateur. Même si le framework tend à simplifier le développement d'un site web version mobile, il ne dispense pas de reprogrammer l'ensemble du site afin de le rendre le plus ergonomique à chacun des périphériques. Donc il est déconseillé de l'utiliser pour une simple optimisation d'affichage, pour cela il est plus astucieux d'utiliser la technique de responsive design sans avoir recours au framework qui lui nécessite une réécriture du code source du site.

Le poids de la librairie est également un paramètre non négligeable : 24KB pour le script minifié⁶, 7KB pour la feuille de style de base minifiée elle aussi. Sans oublier qu'il faut inclure le framework de base jQuery. Autant de ressources à télécharger pour un utilisateur en situation de mobilité dont la bande passante peut être assez limitée.

Enfin, notons que malgré les progrès de Javascript et HTML5, que tous ce qui est faisable sur une application native ne l'est pas forcément par le biais d'une application web réalisée avec un tel framework.

5. Feuille de style permettant de définir la présentation graphique d'une page HTML

6. Script javascript dont le code est compressé, c'est sans espace ni commentaire afin de réduire le poids du fichier

2.3.4 Programmation du compteur

Pour comparer ces trois frameworks, nous devons utiliser une même base, c'est à dire programmer une application identique pour les trois frameworks étudiés. Nous allons donc programmer quelque chose d'assez simple, il s'agit d'un simple compteur disposant de trois boutons, la remise à zéro, l'incréméntation et la décrémentation. Le corps du programme est écrit en HTML et Javascript, et disponible dans l'annexe B.

Pour chaque framework, nous affichons une image correspondant à l'application durant son exécution, afin de pouvoir différencier les différentes formes de rendu (natif ou web). Cet exercice nous est également utile afin de pouvoir comparer la vitesse d'exécution du programme, c'est le temps que va mettre l'application pour mettre à jour le compteur à partir d'un appui sur un des boutons.

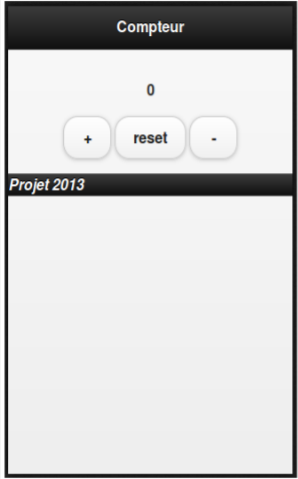
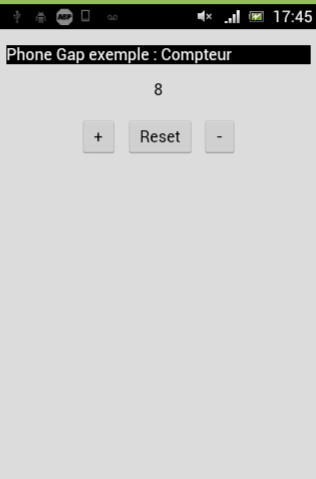

Framework	Jquery Mobile	Phone Gap	Titanium
Capture d'écran de l'application			
Type d'application	Application Web	Application Hybride	Application Native

FIGURE 2.3: Implémentation du programme compteur avec différent framework

D'après la figure 2.3, nous remarquons que l'application réalisé avec Phone Gap possède un style graphique très proche de celui des applications natives.

Mais la réalité est que l'application native et l'application web sont moins rapides que celle qui est native pour mettre à jour le compteur.

Chapitre 3

Proposition

3.1 Analyse des critères

Nous retenons des systèmes précédent, plusieurs points :

- Simplification du développement d’application mobile : Titanium nous fournit tout les outils pour concevoir une application et pour en générer un exécutable.
- Accès aux ressources matérielles du téléphone : Dans une application web, conçue par exemple avec JQuery Mobile, l’accès au matériel du téléphone (Caméra, GPS, ect.) est impossible. Seules les applications hybrides et natives permettent un tel accès.
- Un système de *plugins* : Un exemple est le framework Phone Gap qui met en place un système de plugins complet et puissant, permettant ainsi d’étendre les fonctionnalités du framework par des développeurs indépendant du projet.

Nous constatons que l’ensemble des framework étudiés proposent à l’utilisateur d’établir le code source de son application avec les technologies HTML/-Javascript et CSS. Nous avons décidé de proposer aux futurs utilisateurs de notre framework ces même technologies car elles permettent rapidement et simplement de mettre en oeuvre un projet.

Nous allons nous concentrer sur la génération d’application native. D’après [Bal12], les applications natives offrent une uniformité entre elles. Cette uniformité est caractérisée par principalement deux choses pour l’utilisateur. La première est que l’utilisateur peut communiquer de la même façon (en utilisant les mêmes gestes et interactions) avec les applications natives. La deuxième est que les applications héritent en grande partie de la convivialité inhérente aux cadre de base et semblent immédiatement familières pour les utilisateurs.

Pour générer un exécutable a partir du code de l’utilisateur, Titanium utilise la transformation de code. Or la transformation de code est fastidieuse à mettre en place dans notre cas, et par manque de temps nous avons choisi d’interpréter

le code source fourni par l'utilisateur lors de l'exécution de l'application sur un téléphone.

Nous remarquons que dans les différents frameworks, aucun ne propose une solution pour organiser le code source d'une application. Il est donc intéressant de créer un outil indépendant du framework afin de structurer le code. L'indépendance de cet outil laisse le libre choix au développeur de l'utiliser.

Seul Titanium propose un outil de génération d'exécutable. Cependant ce dernier est délocalisé sur un serveur. Plusieurs inconvénients interviennent, nous pouvons citer la confidentialité du code qui n'est pas assurée et la trop grande dépendance aux problèmes techniques qui peuvent survenir sur le serveur. C'est pourquoi nous allons créer nos propres outils de génération d'exécutables que l'utilisateur pourra utiliser en toute liberté.

De tout ces critères, nous espérons tirer de ce framework les avantages suivants :

- La rentabilité : Un seul code source pour plusieurs plateformes mobiles supportées.
- La réutilisabilité : Le framework s'adapte facilement au support d'une nouvelle plateforme mobile.
- L'extensibilité : Le framework peut accueillir de nouvelles fonctionnalités indépendamment de son développement.

Dans la suite du projet, nous avons décidé de nommer notre Framework "rizla".

3.2 Cahier des charges

3.2.1 Présentation générale du problème

Enoncé du besoin

L'objectif principal est de proposer une solution à la problématique sous forme de framework qui, à partir d'un même code source, doit être capable de générer une application vers différentes plateformes. Ces plateformes sont les suivantes : Android, Windows Phone, et Firefox OS.

Le framework doit donc fournir un outil de génération d'application mobile vers l'une des plateformes supportées. Mais il doit également proposer des outils pour faciliter le développement. L'utilisateur du framework doit, à l'aide de ces outils, pouvoir créer un nouveau projet, créer des plugins pour le framework et également créer des bundles (nous détaillerons ces éléments plus tard).

Environnement du produit recherché

Équipements

Afin de mener à bien ce projet, nous disposons de plusieurs périphériques qui nous serviront tout au long du développement. Ces derniers seront utilisés pour

des tests et pour la présentation finale du produit. Voici une liste du matériel que nous utiliserons tout au long du projet :

Android :

Nous disposons de deux smartphones équipés de la version d'Android 4.0 (Ice Cream Sandwich), ainsi qu'un troisième équipé de la version 2.3.3 (Gingerbread). Le SDK Android (Software development kit) nous fournit aussi un émulateur prenant en compte toutes les versions de la plateforme déjà sortie sur le marché. Nous développerons cette partie sous le logiciel Eclipse en Java.

FirefoxOS :

Cette plateforme en est à ces débuts sur le marché des smartphones, il n'y a donc actuellement que très peu de smartphones la supportant. En prenant en compte de ce paramètre, nous développerons notre produit sur un émulateur fourni par le SDK.

Windows Phone :

L'IDE (Integrated Development Environment) ce qui signifie « Environnement de développement intégré » que nous utiliserons est fourni directement dans le SDK de Microsoft et s'appelle Visual Studio Express for Windows Phone. Nous utiliserons l'émulateur de smartphone fournis par le SDK étant donné que nous ne disposons pas de terminaux Windows Phone.

Contraintes

Une première contrainte matérielle intervient au niveau de la plateforme Windows Phone. En effet, le SDK fournis par cette dernière ne fonctionne pas sur d'autres systèmes d'exploitation que Windows 7 (ou supérieur). Nous utiliserons donc un matériel adapté afin d'assurer une portabilité du logiciel vers la plateforme Windows Phone.

De plus, la dernière version du sdk, Windows Phone 8, ne peut-être installée que sur un système 64-bits. Nous avons donc choisis de développer notre framework à partir de la version 7.1, qui elle, est également disponible pour les systèmes 32-bits.

Concernant les plateformes mobiles FirefoxOS et Windows Phone nous ne pourrions assurer un support se limitant aux fonctionnalités fournies par l'émulateur. En effet, ne possédant pas de smartphones physiques équipés de ces plateformes afin d'effectuer des tests en profondeur, nous serons limités par les capacités offertes par l'émulateur en question.

Nous devons également nous plier à la contrainte qu'est le temps imparti pour réaliser ce projet. En effet, rapport et travail final sont à rendre le 17 Mai dernier délai, ce qui nous laisse quatre mois complets pour mener à terme ce TER.

3.2.2 Expression fonctionnelle du besoin

Cadre d'utilisation

Le cadre d'utilisation du logiciel se situe dans le secteur du développement informatique, plus précisément dans le développement d'applications pour plate-

formes mobiles. Le framework s'adresse à plusieurs types d'utilisateurs : les premiers utilisateurs concernés par notre logiciel seront les développeurs d'applications mobiles. Ensuite, viennent les développeurs qui voudront améliorer le framework par l'ajout de fonctionnalités via le système de plugins. Nous retrouvons aussi les développeur de *bundles* qui voudront partager leurs bout d'application. Enfin, il ne faut pas oublier les utilisateurs de l'application finale dont le bon fonctionnement est directement relié au framework et au développeur qui aura mis au point cette application.

Fonction d'usage

Le logiciel doit aider à la conception et à la création d'applications mobiles, et permettre la génération de ces dernières, à partir d'un même code source, en exécutables directement lisibles par les plateformes mobiles Android, FirefoxOS et Windows Phone.

Fonctions de service et de contrainte

Créer une application mobile multi-plateforme :

Solution technique : Outils permettant la création d'un projet d'application mobile.

Commentaires : Un projet est un ensemble de répertoires et de fichiers, permettant d'imposer une structure similaire quel que soit l'application mobile développée à l'aide du logiciel.

Générer des exécutables vers des plateformes mobile :

Solution technique : Outils fournis par les SDK des plateformes supportées par le logiciel, et permettant de générer des application mobiles.

Contrainte : Les exécutables devront être disponible pour les plateformes mobile Android, FirefoxOS et Windows Phone.

Commentaires : Les outils de génération d'application proviennent de l'extérieur, ils ne sont en aucun cas propre au logiciel.

Étendre le produit :

Solution technique : Ajouter une fonctionnalité *Plugin*, qui est un module externe permettant l'ajout d'une nouvelle fonctionnalité au logiciel.

Debugger une application mobile multi-plateforme :

Solution technique : Générer un exécutable de l'application multi-plateforme vers une plateforme mobile cible, l'installer sur un smartphone ou émulateur connecté à la machine de développement, puis l'exécuter sur le smartphone.

Commentaires : Ce besoin exprime un moyen de tester les applications développées de manière rapide et efficace.

Critères d'appréciation

La rapidité de l'exécution de l'application générée est un des critères qui rends notre projet effectif et opérationnel. Par exemple, l'application finale ne devra pas "freezer" ou avoir un temps de chargement trop lourd à supporter pour l'utilisateur.

Le rendu visuel devra être similaire aux applications natives de façon à ne pas déstabiliser l'utilisateur.

La prise en main du framework par un nouvel utilisateur ne devra pas être trop fastidieuse.

3.2.3 Cadre de réponse

Options et variantes proposées non retenues au cahier des charges

Etant donné la multitude de plateformes disponibles pour terminaux mobiles, nous avons limités notre développement à trois d'entre elles : Android, Firefox OS, Windows Phone. Nos critères de choix se basent sur le fait qu'Android est l'OS (Operating System) le plus populaire et le plus répandu. En effet, Android dispose de 58% des parts de marché au premier semestre de 2013 (source : [http ://www.journaldunet.com/ebusiness/internet-mobile/part-de-marche-os-mobile-q1-2013-0413.shtml](http://www.journaldunet.com/ebusiness/internet-mobile/part-de-marche-os-mobile-q1-2013-0413.shtml)). Nous avons délibérément choisis de ne pas développer notre framework pour supporter des applications sur iOS, le système d'exploitation d'Apple, car nous préférons nous orienter vers les deux plateformes émergentes que sont Firefox OS et Windows Phone. De cette manière, nous pourrions découvrir deux nouvelles façon de concevoir des applications mobiles. De plus, Firefox OS utilise le moteur de rendu Gecko pour faire fonctionner des applications web développées en format HTML5 ce qui est un avantage pour notre projet (voir Etat de l'art).

Perspective d'évolution technologique

Dans le cadre de l'évolution du logiciel, nous pouvons aborder le thème des plateformes mobiles supportées. Une évolution possible serait d'ajouter en plus des plateformes mobiles déjà supportés, d'autres plateformes telles que IOs (IPhone) ou BlackBerry OS afin de permettre au développeur d'applications d'élargir au maximum son rayon d'action.

Chapitre 4

Conception

4.1 Analyse

4.1.1 Première approche

La première approche consiste à analyser le déroulement des différentes étapes effectuées par le framework depuis la conception d'une application mobile jusqu'à son exécution sur un smartphone. Voici un schéma de conception d'application à l'aide du framework :

1. L'utilisateur fournit un code source au framework (sous la forme de fichier HTML/Javascript)
2. Le framework, à partir du code source, fournit à l'utilisateur un exécutable qui correspond à l'application mobile (les différentes plateformes mobiles ont chacune leur propre exécutable)
3. L'utilisateur installe et exécute l'application sur un appareil mobile

Ces trois étapes (Conception, génération et exécution) constituent le fondement même du framework. Voyons dans le détail le fonctionnement de ces dernières.

La conception

Un des points importants dans ce framework est la centralisation des données dans une seule architecture. En effet, comme nous avons pu le remarquer avec le framework Phone Gap, la génération d'exécutable nécessite de créer manuellement autant de projets natifs que de plateformes mobiles visées. Nous allons, dans ce projet, automatiser ces tâches afin de simplifier la génération d'exécutable, et de permettre à l'utilisateur de pouvoir concevoir une application mobile sans connaissance préalable des différentes méthodes de développement et API proposées par les plateformes mobiles.

Le framework se présente à l'utilisateur sous la forme d'un ensemble de répertoires, de bibliothèques et d'exécutables. L'utilisateur va donc pouvoir gérer un ensemble de

ressources soigneusement regroupées, afin de concevoir son application mobile. Ces ressources sont :

- Un espace dédié au code source de l'application.
- Un espace contenant toutes les bibliothèques utilisées dans la conception de l'application mobile.
- Un espace où l'utilisateur peut ranger tout les fichiers images, vidéos et CSS(apparence de l'application).
- Un espace contenant tout les plugins liés au projet.

L'auteur doit donc concevoir son application mobile au sein du framework, afin de pouvoir accéder entre autre aux phases de debuggage du code source. En effet un point important, durant la phase de développement, est de permettre au développeur de pouvoir tester son application directement sur un navigateur, ou bien directement sur un smartphone grâce à la génération d'exécutable que va proposer le framework.

La génération

La génération d'exécutable (application mobile spécifique à une plateforme) se traduit par une transformation du code source fourni par l'utilisateur en un autre code source lisible directement depuis la plateforme mobile souhaitée.

La génération d'exécutable à partir du code source fourni par l'utilisateur est utilisé par les systèmes Android et Windows Phone. En effet, sur Android par exemple, l'utilisateur écrit un code source en Java, puis à l'aide d'outils fournis par la plateforme, génère une application mobile. Les fichiers Java sont ainsi compilés en fichier *dex*, qui est un langage intermédiaire spécifique à la machine virtuelle Android (Dalvik).

La transformation de code HTML/Javascript vers un autre langage se veut délicate et fastidieuse à mettre en place dans notre situation, d'autant que si nous considérons qu'en règle générale chaque plateforme possède un langage qui lui est propre¹. Nous devons au maximum éviter les dysfonctionnements logiciels en partie causés par l'évolution fréquente des différentes plateformes mobiles. C'est pourquoi nous avons choisi de ne pas transformer le code HTML/Javascript fourni par l'utilisateur.

La solution que nous avons retenue est d'interpréter le code source lors de l'exécution de l'application mobile. Cet interpréteur est un navigateur web, car ce dernier prend en charge les fichiers de type HTML/Javascript. De plus, lors de nos études avec les différentes plateformes mobiles, nous avons remarqué que toutes ces dernières incorporent un navigateur web, ce qui nous permet ainsi d'élargir notre gamme de plateformes prises en charge par le framework.

La question qui se pose est de trouver une solution pour fournir les fichiers sources HTML/Javascript de l'utilisateur aux outils de générations d'applications mobiles des différentes plateformes? Pour répondre à cette question, nous

1. Android et Java, Windows Phone et C#, Firefox OS et HTML/Javascript, iOS et Objective C

mettons en place des squelettes d'applications natives. Un squelette d'application native est le code source d'une application mobile spécifique à une plateforme, écrit dans le langage que cette dernière impose. C'est dans ce squelette que nous mettons en place le navigateur web et toutes les configurations nécessaires au bon déroulement de l'interprétation des fichiers sources de l'utilisateur. En associant le squelette et les fichiers sources de l'utilisateur, nous composons le code source d'une application complète, dans le respect des règles des plateformes mobiles. Une fois cette étape franchie, il ne nous reste plus qu'à appeler les outils de génération d'exécutables des plateformes pour obtenir notre application.

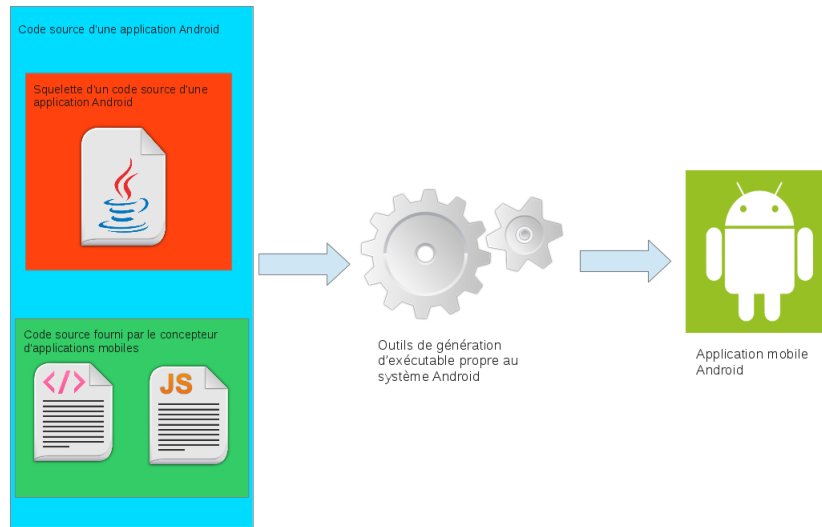


FIGURE 4.1: Génération d'une application Android qui inclue les fichiers HTML/Javascript fournis par l'utilisateur

L'exécution

L'exécution de l'application mobile sur un smartphone (ou émulateur) est l'étape qui va correspondre à l'interprétation des fichiers sources fournis par l'utilisateur. Le navigateur web va donc se mettre en place et charger un premier fichier HTML/Javascript choisi par le concepteur de l'application. Afin de respecter les critères que nous avons définis dans le cahier des charges, nous rendons le navigateur web invisible au yeux de l'utilisateur de l'application mobile. De cette façon nous rendons impossible l'affichage du contenu de la page HTML depuis le navigateur web, pour laisser place à la construction de notre propre

vue (rendu natif).

Un point important de cette étape est le dialogue entre le Javascript contenu dans la page HTML, et l'API native² de la plateforme mobile en question. Sur Android, par exemple, pour construire et afficher un bouton sur la vue d'une application mobile durant son exécution (dynamiquement), il faut appeler une méthode fournie par l'API Android et écrite en Java. Le navigateur web que nous avons mis en place précédemment, nous offre un moyen de communiquer entre cette API et le Javascript. En effet, nous pouvons à partir d'une fonction écrite en Javascript et lu par le navigateur, appeler une fonction écrite en Java du côté de la plateforme mobile. Le navigateur web nous permet aussi de communiquer dans le sens inverse, c'est à dire du Java vers le Javascript. Grâce à ce dialogue, nous allons pouvoir construire nos vues, et accéder aux différentes ressources matérielles du téléphone (voir figure2.1).

La première étape de cette interprétation va correspondre à l'analyse du document HTML lu par le navigateur. En effet pour construire la vue native, nous avons besoin de connaître le contenu du fichier HTML. Nous avons donc besoin de définir un programme Javascript, un moteur de rendu, qui va s'occuper d'analyser syntaxiquement³ le HTML et d'appeler l'API native de la plateforme native concernée en conséquence. Dans le cas contraire la page correspondant au fichier serait non affichable.

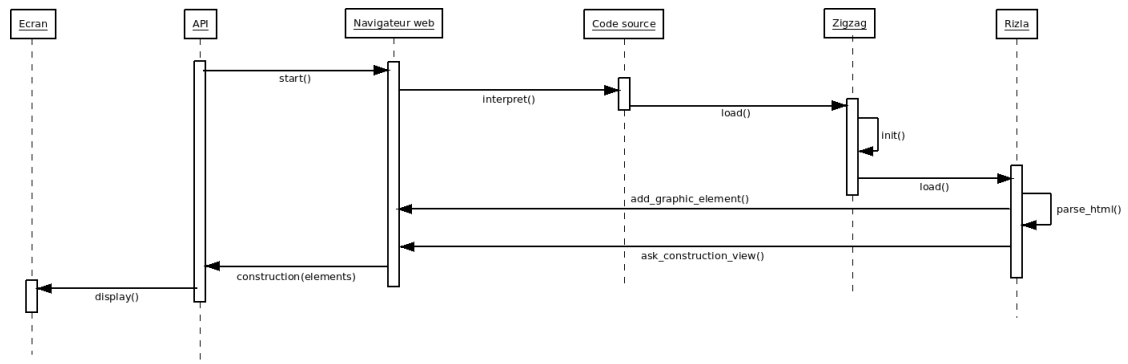


FIGURE 4.2: Diagramme de séquence représentant la construction de la vue native durant l'exécution d'une application

4.1.2 Les Plugins

Les plugins sont des modules externes qui complètent le framework pour lui apporter de nouvelles fonctionnalités. Les plugins sont mis au point par des

2. Interface de programmation permettant d'accéder à toutes les ressources du smart-phone : accès au matériel, construction des vues.

3. de parser en anglais

développeurs n'ayant pas nécessairement de relation avec les auteurs du framework, mais ayant des connaissances dans les plateformes mobiles sur lesquelles les plugins sont basés. Par exemple nous pourrions avoir un plugin qui permettrait d'accéder à l'appareil photo afin de prendre des clichés. Nous avons donc décidé d'inclure un système de plugins, afin de permettre l'extensibilité du framework. La question ici va être de définir les modules externes, à quoi correspondent-ils, et surtout de quelle manière les connecter au framework afin que les utilisateurs puissent s'en servir simplement et efficacement.

Etudes de cas

Nous avons vu précédemment le framework Phone Gap qui permet de concevoir des applications mobiles hybrides, permettant ainsi un accès au matériel du téléphone via une interface Javascript. Cette dernière est articulée autour d'un système de plugins complet, donnant la possibilité à des développeurs externes⁴ d'ajouter de nouvelles fonctionnalités au logiciel.

De part la similitude des intentions, le système de plugins de ce framework est intéressant pour nous, car il aborde le même problème. En effet, un plugin est censé apporter une nouvelle fonctionnalité au framework, mais que faire lorsque celle-ci fonctionne de manière différente sur chaque plateforme mobile ? Pour répondre à ce problème, Phone Gap utilise deux notions : le *front-end* et le *back-end*, notions que l'on retrouve dans d'autres systèmes notamment dans la programmation web⁵.

Le *front-end* correspond à ce que l'utilisateur aura la capacité de voir. Par exemple pour un plugin qui utilise l'appareil photo d'un téléphone, on aura deux méthodes : *takePicture()* et *savePicture()*. Ces fonctions décrivent l'interface du plugin et sont définies dans un unique fichier Javascript. Ce dernier étant le point d'entrée du plugin dans sa liaison avec le framework.

Le *back end* correspond à l'implémentation native des fonctions définies dans le fichier d'interface Javascript. Ces implémentations respectent les architectures spécifiques aux plateformes mobiles souhaitées, assurant ainsi leur support. Lorsque l'utilisateur du plugin appelle une fonction définie dans le fichier d'interface, son implémentation native est alors invoquée par le framework. Ce dernier se charge automatiquement de vérifier sur quelle plateforme mobile le plugin est exécuté, afin d'appeler la bonne fonction.

Notre système de plugins

Nous avons décidé de nous inspirer du système de plugins qu'offre le framework Phone Gap. Nous allons donc mettre en place une architecture qui contiendra un unique fichier Javascript contenant les fonctionnalités d'interfaces, ainsi que

4. Développeurs non liés au framework en lui-même

5. Le *front-end* va correspondre à la partie client du site internet, alors que le *back-end* s'identifiera à la partie serveur. Les mots *front-facing* et *back-facing* sont aussi utilisés.

plusieurs répertoire aux noms des différentes plateformes mobiles, qui contiendront les implémentations native des fonctions.

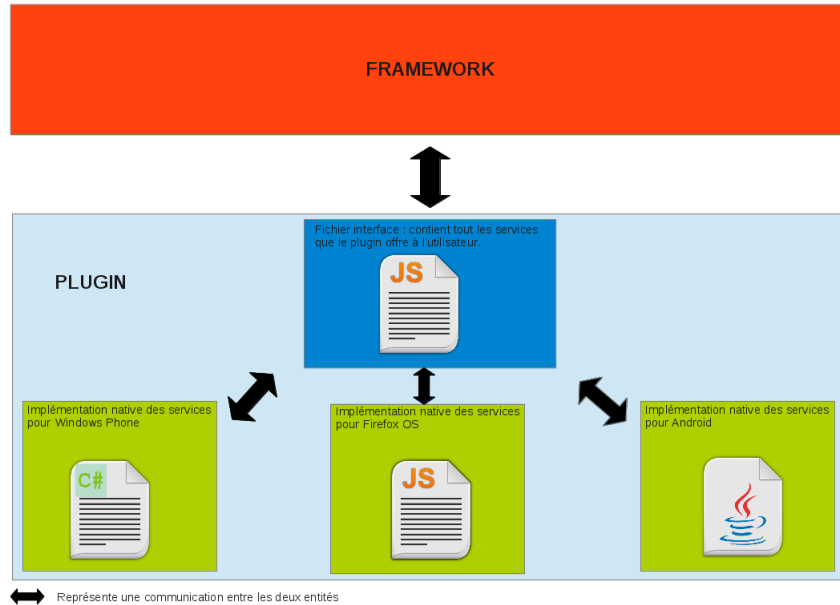


FIGURE 4.3: Communication entre un plugin et le framework durant l'exécution d'une application mobile.

4.1.3 Bundles

Pour le moment, nous avons étudié une architecture afin d'étendre le framework au besoin des développeurs. Il est intéressant à présent de mettre en place une architecture qui permet d'organiser le développement d'une application. Le but principal de cette architecture est de fournir au développeur un moyen de réutiliser facilement des parties d'applications. Une partie d'application peut être définie comme un ensemble de fonctionnalités. Suite à diverses recherches, nous avons découvert la programmation par composant (POC) qui semble correspondre à nos attentes. Nous allons brièvement étudier ce paradigme de programmation. Cela nous permettra de nous en inspirer dans le but de donner la possibilité de découper le développement d'application en plusieurs composants interconnectés.

La programmation par composant est une approche modulaire qui permet de découper une application en plusieurs morceaux appelés composants. Un composant est donc un bout d'application qui regroupe un ensemble de fonctionnalités.

Celui-ci possède un nombre quelconque d'entrée et de sortie qui servent à connecter d'autres composants. Nous différencions deux types de développeur, celui qui programme les composants et l'autre qui les connecte entre eux afin de créer une application.

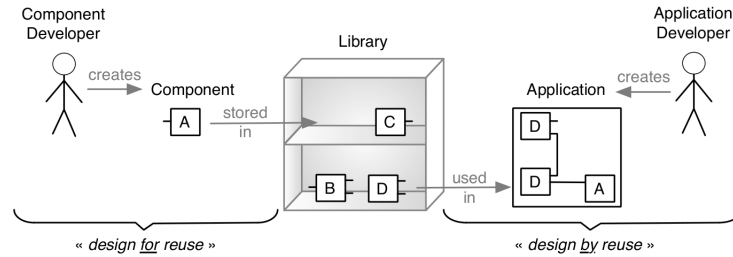


FIGURE 4.4: The Component-Oriented Programming Duality [LFH10]

Comme le montre la figure 4 [LFH10], le développeur de composant met à disposition différents composants pour que les développeurs d'application puisse les utiliser et les connecter. Cela implique que pour développer un composant, il faut être capable de planifier le maximum d'entrée/sortie puisque l'interconnexion dépend de ces dernières. Le développeur d'application ne doit pas se préoccuper du code contenu dans le composant⁶. De ce fait, il est nécessaire de fournir avec un composant, une documentation afin de décrire les fonctionnalités implémentées mais aussi un contrat qui définit les entrées/sorties disponibles.

Les programmeurs de composants peuvent aller encore plus loin dans le développement en faisant des agrégations de composants. Cette encapsulation de composants est appelée composite. La figure 5 est un exemple simple de composite contenant deux composants.

La notion de composant est encore plus complexe mais la simple définition générale nous est nécessaire pour la suite du projet.

Les avantages dans l'utilisation de la programmation orientée composants sont multiples. Ce type de programmation est particulièrement pratique pour le travail en équipe car elle divise le travail en plusieurs composants qui sont à la fin très facilement assemblés. De plus, le partage en composant permet la ré-utilisabilité de ces derniers au sein d'une autre application. Nous pouvons également noter un avantage non-négligeable qui est la facilité pour la maintenance du code. Malgré de nombreux avantages, la POC présente une grande difficulté pour les programmeurs qui est de factoriser un logiciel en composants. De plus, la planification de toutes les entrées/sorties possible pour un composant est impossible. Ainsi, la POC demande une analyse plus poussée que pour un autre style de programmation puisqu'elle requiert, avant de débiter le projet, d'en connaître l'ensemble des briques logicielles qui le compose.

6. On note que la plupart du temps un composant ne contient que du code compilé

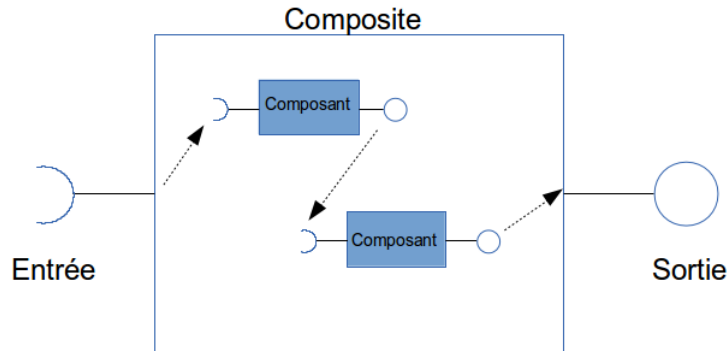


FIGURE 4.5: Exemple de composite

Nous allons donc nous inspirer de la notion de composant pour proposer aux utilisateurs de notre framework une architecture qui facilite la ré-utilisabilité du code. Nous allons donc créer une version de composant simplifiée appelé Bundle.

4.1.4 Utilisateurs

Nous allons, dans cette section, analyser les différents utilisateurs possible du framework. En effet, le produit final se destine à être utilisé par des développeur qui voudront concevoir des applications mobiles. Nous pouvons donc considérer plusieurs catégories de développeur dans notre cas. Nous avons décider de les classer par hiérarchie, en analogie aux connaissances requises du système et de ses technologies.

- Les développeurs de plugins : Ils ont la maîtrise des plateformes natives dans laquelle ils créent un plugin, ainsi que de l'architecture du framework. Leur rôle se définit par l'ajout de nouvelles fonctionnalités au framework.
- Les développeurs de bundles : Ils ont la maîtrise de la notion de bundle ainsi que de l'architecture du framework. Leur rôle se définit par la création de bout d'application, que d'autres développeurs pourront utiliser à leur tour.
- Les développeurs d'applications mobiles : Ceux ci on un minimum de connaissances en Javascript afin de pouvoir connecter les bundles, ainsi que dans l'utilisation du framework afin de générer des exécutables d'applications mobiles.

La notion commune à tout les développeurs est la maîtrise des technologies HTML/Javascript et CSS.

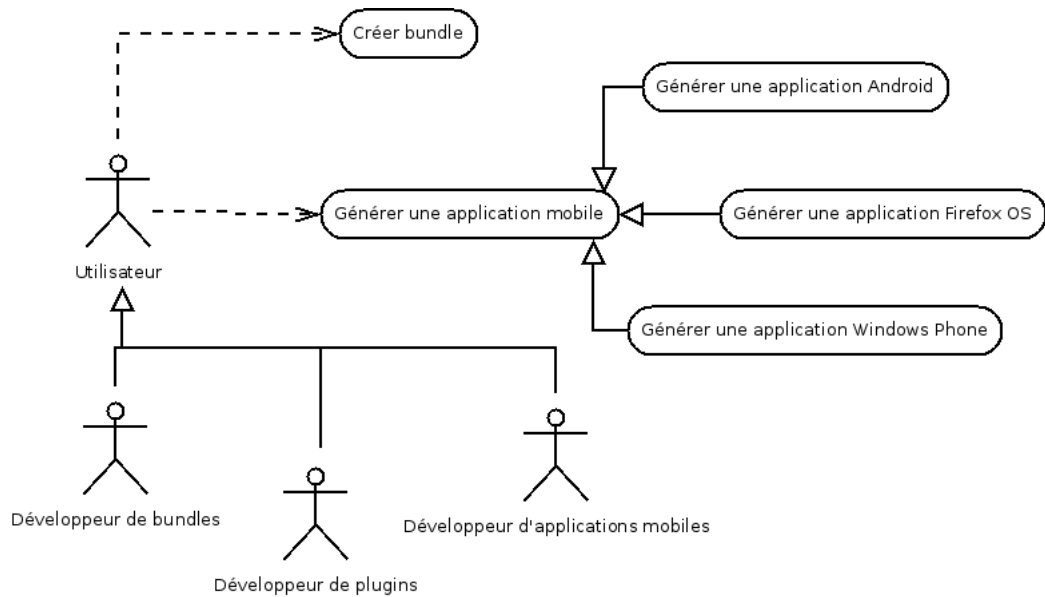


FIGURE 4.6: Cas d'utilisations des outils de génération

4.2 Architecture

4.2.1 Général

Il nous faut maintenant définir comment sera organisé l'espace de développement du framework. Voici les différents répertoires contenus dans le framework ainsi que leurs rôle dans la conception d'applications :

src : Représente toutes les sources de l'application mobile, définies par des fichiers HTML/Javascript, le tout étant organisé suivant le modèle imposé par les *bundles*.

lib : Représente les bibliothèques utilisées par l'application mobile lors de son exécution.

bin : Contient tous les outils qui permettent de générer des exécutables d'applications mobiles.

plugins : Représente tous les plugins utilisés par le développeur.

config : Contient les fichiers de configuration du framework.

ressources : Représente tous les fichiers considérés comme des ressources, les images, les vidéos, les sons et les fichiers CSS.

build : Une fois que l'utilisateur génère un exécutable pour une application mobile, elle se retrouve dans ce répertoire.

doc : La documentation du framework.

L'espace de travail est donc organisé de sorte à ce que l'utilisateur puisse centraliser les fichiers qui le nécessitent (par exemple les bibliothèques, ou des images utilisées plusieurs fois dans une même application).

Les outils de génération d'exécutables

Les outils de génération d'exécutables permettent à partir du code source de l'application mobile fourni par l'utilisateur, de générer plusieurs exécutables visant différentes plateformes mobiles. Ces outils sont en liaisons avec les fichiers de configuration définis par l'utilisateur. Ces derniers contiendront des champs de valeurs qui auront leur sens une fois la génération d'un exécutable en cours. Par exemple, une application mobile Android, Windows Phone ou Firefox OS possède une icône par défaut qui la représente dans le menu de ces plateformes. Grâce aux fichiers de configuration, l'utilisateur pourra définir une icône de son choix pour l'application en ajoutant une simple ligne dans un des fichiers de configuration. Voici un exemple de champ possible :

```
icon = ../img/mon_icone.png
```

4.2.2 Structure des plugins

Nous avons vu précédemment que les *plugins* sont organisés autour d'un fichier interface en Javascript, et reliés à des implémentations natives des services que propose ce dernier. Voyons en détail le fonctionnement global d'un *plugin*

Description des services

Le fichier interface est écrit en Javascript, car il sera appelé par le framework durant l'exécution de l'application mobile. En effet, ce dernier va proposer un ensemble de fonctions qui seront utilisables par le développeur de l'application.

Afin d'étendre au maximum le framework, et de permettre une grande liberté d'action de la part des développeurs de *plugins*, nous décidons d'imposer un minimum de règle à respecter. Premièrement, le développeur de plugins devra étendre un objet issu du framework et définissant certaines méthodes. Une fonction d'initialisation est mise en place, elle sera appelée avant le rendu graphique de l'application. Cela peut être utile dans le cas d'un plugin nécessitant de se mettre en place rapidement dès le lancement de l'application.

Un des points importants à ne pas oublier est la reconnaissance automatique de la plateforme mobile d'exécution dans le corps des fonctions. Ainsi, suivant la plateforme dans laquelle on se trouve le plugin s'adaptera en conséquence. De plus, un champ est réservé à la feuille de style CSS, permettant ainsi au développeur de *plugin* de définir le design associé à son intention.

Voici un exemple d'un plugin simple en Javascript :

```
Rizla.plugins.Hello = {  
  name : 'Hello',  
  version : '0.1',  
  css : 'css/hello.css',  
}
```

```

onInit : function(){
    /* fonction d'initialisation */
    console.log("Initialisation");
},

/* fonction simple, qui pourra être appelée par l'utilisateur du plugin */
hello : function(){
    Rizla.callPlugin("Hello.hello");
},
};

```

L'utilisateur du *plugin* en question pourra donc appeler la fonction *hello()* du plugin dans le code source de son application de cette manière :

```
Rizla.plugins.Hello.hello();
```

La méthode *callPlugin(name)* permet de sélectionner automatiquement l'implémentation native de la méthode issue du plugins, les deux étant un seul et même paramètre sous forme de chaîne de caractère séparé par un point. Cette syntaxe nous est utile pour identifier un package Java ou un objet Javascript. Suivant la plateforme d'exécution de l'application mobile, le plugin se charge d'appeler l'implémentation correspondante native.

Android

Lorsque qu'un développeur veut développer un plugin pour la plateforme Android, il doit étendre une classe *Plugin* prévue à cet effet. La classe possède

<i>Plugin</i>
-mWebView: WebView
#mContext: Context
+execJS(js:String): void
+run(json: JSONObject): Object

FIGURE 4.7: Classe *Plugin* que le développeur de plugin doit étendre, fournie dans l'API de notre framework

une méthode *run* qui sera appelée indirectement par l'utilisateur. C'est le framework qui se charge d'invoquer cette méthode, car il sait si il se trouve sur la plateforme Android.

Firefox OS

La déclaration des plugins Firefox OS est très simple. Il suffit de définir un objet javascript qui englobe un ensemble de méthodes. Voici un exemple qui introduit une méthode simple :

```

Hello = {
  hello : function(){
    console.log("Hello Firefox OS!");
  },
}

```

4.2.3 Bundle

Comme énoncé précédemment, nous allons nous inspirer des composants pour créer nos *Bundles*. Nous rappelons que l'idée est de proposer les *Bundles*, aux utilisateurs, dans le but de construire une application uniquement en les assemblant. Nous allons voir comment fonctionnent les *Bundles*.

Structure

Dans notre système, les *Bundles* sont des dossiers qui contiennent un ensemble de fichiers. Nous imposerons donc une architecture afin d'ordonner cette ensemble. Utilisons une architecture répandu dans le monde du web, le Modèle Vue Contrôleur (MVC). Le choix du MVC est principalement dû à sa notoriété, nous la retrouvons dans la plupart des frameworks web tel que Symfony, Zend Framework, Silex, EZ-Component, Django... Ainsi, nous nous assurons que les utilisateurs de ces frameworks (développeur web) seront capable de facilement prendre en main notre framework. De plus, les divers frameworks existant qui implémentent le MVC sont, pour nous, une source d'inspiration pour notre projet. Mais qu'est ce que le MVC ? Les créateurs de ce dernier [GEK88] avait pour but de séparer de manière optimal, trois unité fonctionnelle. Cela permet une meilleure compréhension et modification de chacune d'entre elles. Ces trois unités sont nommées modèle, vue et contrôleur. Respectivement, ces unités sont chargé de centralisé les opérations effectuées sur le domaine de l'application, les opérations chargées de l'affichage et les opérations nécessaires à l'interaction utilisateurs entre le modèle et la vue. Nous pouvons donc schématiser la situation de la manière suivante :

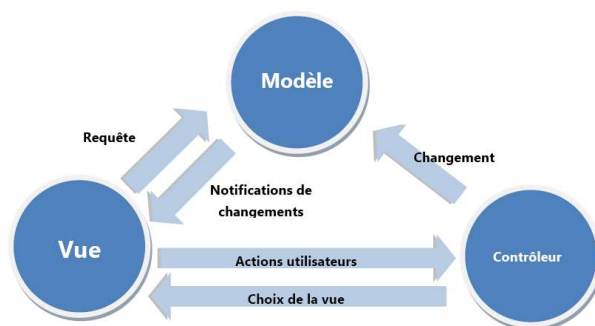


FIGURE 4.8: Schématisation du MVC

Nous devons donc adapter ce paradigme de programmation dans le cadre de notre projet. Notre situation présente quelques inconvénients. En effet, notre configuration est un peu différente de celle du web. Le web fonctionne grâce à la notion de client/serveur. Un client fait une requête sur un serveur qui lui renvoie un fichier.

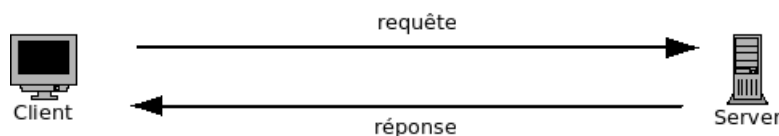


FIGURE 4.9: Principe général du client/serveur

Pour nous la situation est différente car nous ne disposons pas de serveur. Il est vrai qu'il existe aujourd'hui des serveurs javascript comme *Node.js*. Une solution serait donc de lancer en parallèle de notre application un serveur javascript mais les ressources limitées du téléphone ne nous le permettent pas. C'est pourquoi, nous devons simuler le comportement d'un serveur. Cette simulation pourrait paraître sans intérêt puisque le développeur web peut très bien programmer uniquement côté client. Mais il serait vite limité comme par exemple l'envoi de formulaire qui lui serait impossible. Ainsi ses habitudes serait bouleversé. Le but du framework étant de faciliter le développement d'application mobile notamment pour les développeurs web, la programmation doit lui paraître la plus familière possible. Donc ne bousculons pas ses habitudes et voyons comment un framework web habituellement gère le MVC pour essayer de retranscrire au plus près.

La vue

Un des éléments qui ne change pas dans notre cas, c'est la vue. En effet, la présentation des données dans le web se fait via les fichiers HTML. Généralement les frameworks web utilisent des moteurs de templates. Son rôle est principalement d'augmenter la lisibilité du code. Les moteurs de templates permettent de formater un contexte de données pour les présenter à l'utilisateur.

L'utilisation d'un moteur de template donne un gros avantage. Il fournit une syntaxe particulière permettant comme un langage de programmation de, bouclé (for ou while) et de faire des conditions (if), sur des variables du contexte. Ainsi la vue est codé uniquement avec cette syntaxe sans utiliser un métissage de langage de programmation.

Dans notre projet, nous utilisons un moteur de template existant. Ecrit en javascript, il se nomme *Handlebars.js*. Nous modifions l'extension des vues en *.tpl*. La raison de ce choix est de permettre de bien différencier les templates des usuelles fichiers *html*. Ainsi les vues sont rapidement mis en place dans le système. Etudions, à présent le cas des contrôleurs.

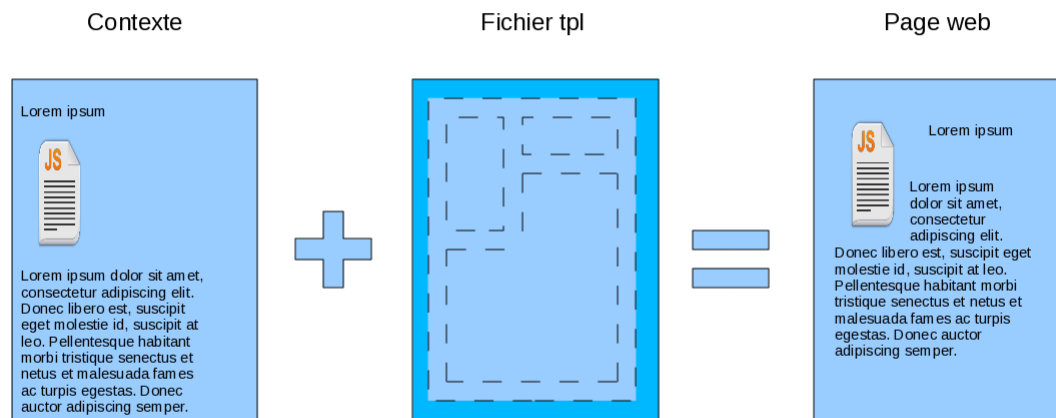


FIGURE 4.10: Mécanisme du moteur de template

Le contrôleur

Traditionnellement, lors d'une requête vers le serveur, les frameworks web font appel à un bootstrap⁷. C'est lui qui se charge d'instancier le contrôleur et de faire appel à l'action⁸ demandée. Et c'est l'action qui retourne au client la vue associé à la requête. L'usage d'un serveur présente donc un avantage. Il peut analyser la requête envoyé, construire un fichier (*html* en général) en conséquence, pour ensuite le retourner au client. Dans notre cas, nous n'avons pas la possibilité de construire un fichier *html* en fonction d'une requête. La seule chose possible est de charger directement un fichier *html* en le ciblant grâce à l'url. C'est donc à notre fichier *html* de jouer le rôle de bootstrap. C'est à dire que nos vues doivent charger un script javascript qui instancie le contrôleur et appelle la méthode. La question suivante se pose : Comment sans requête le script peut-il instancier les bon objets ? Utilisons tous simplement l'url qui est capable de transporter des informations sérialisées. L'url peut contenir

7.

8. Généralement, les méthodes d'un contrôleur qui peuvent être appelé via une requête sont appelé *action*

des paramètres, il nous suffit simplement lorsque l'on cible le fichier HTML de transmettre en paramètre le nom du contrôleur et de l'action. Ainsi le script n'a plus qu'à analyser les paramètres de l'url pour trouver le contrôleur et l'action. Pour ne pas se noyer dans les détails schématisons la situation :

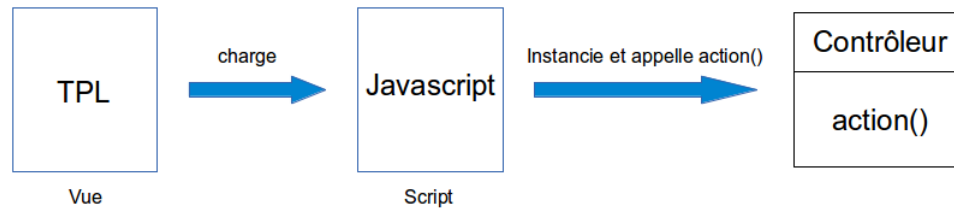


FIGURE 4.11: La vue appelle le contrôleur

Même si la structure reste similaire pour le programmeur, l'ordonnement est lui différent. Mais ayons un œil critique sur la solution proposée. Pour sélectionner un contrôleur, ainsi que son action, un système d'url paramétrée est proposé. Ne faudrait-il pas automatiser ce système ? Cela permettrait entre autre d'éviter les erreurs du programmeur lors du paramétrage. Nous allons donc mettre en place un routeur en javascript.

Le routage (figure 4.12) va permettre au développeur d'associer un chemin d'accès, c'est à dire un nom de contrôleur et d'action, à un identifiant (choisi par le développeur) appelé route. Ainsi, nous construisons une table de routage. Dès lors, le développeur peut demander au routeur de charger une route. Cela engendrera le mécanisme suivant :

Les unités de vue et de contrôleurs sont maintenant correctement gérées par l'application. Voyons à présent comment traiter le cas des modèles dans notre projet.

Le modèle

Sous sa forme la plus brute, la couche modèle peut être vue comme les données manipulées dans l'application. Par données nous entendons tout ce qui est persistant, c'est-à-dire tout ce que nous pourrions lire à partir d'une source, et modifier pour le relire plus tard si besoin est. La persistance d'une donnée peut être assurée par l'utilisation d'une base de données. Les trois plateformes étudiées dans ce projet possèdent un système de gestion de base de données (SGBD) : SQLite pour Android et Firefox OS, Microsoft SQL Server pour Windows Phone.

La logique voudrait donc que nous établissions un système qui puisse abstraire la communication avec les différents SGBD. L'utilisateur pourrait ainsi stocker des données simplement et sans se soucier du SGBD présent sur la plateforme mobile. Par soucis de temps, nous avons décidé de ne pas étudier cette partie en profondeur. C'est pourquoi nous introduisons une nouvelle notion : le

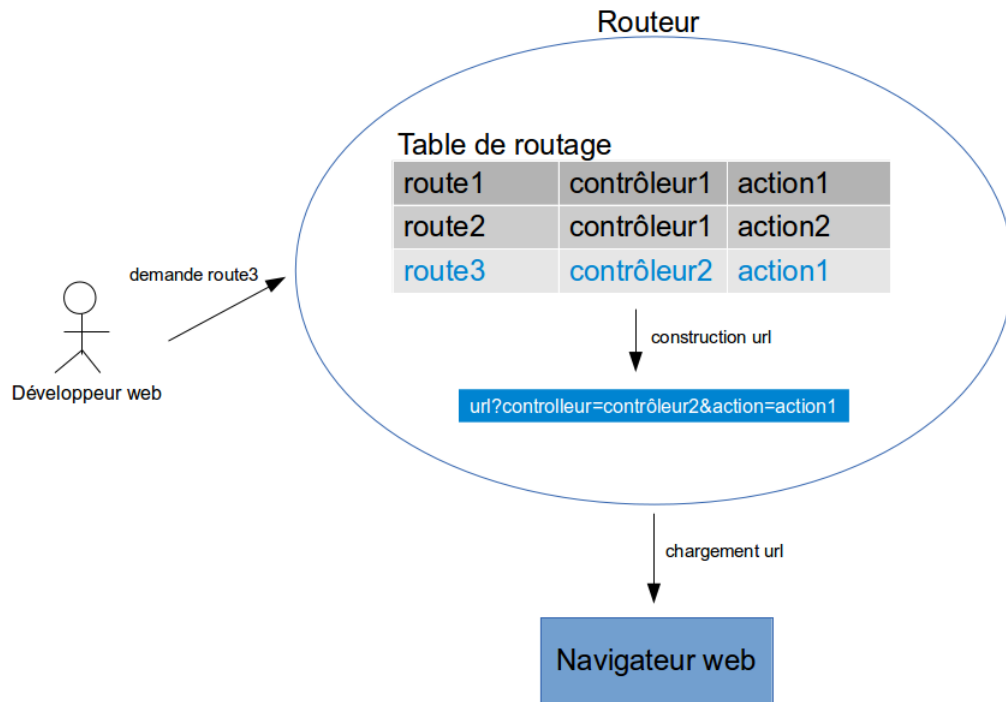


FIGURE 4.12: Fonctionnement du routage

*Web Storage*⁹.

Le *Web Storage* permet le stockage persistant des données dans le navigateur web. Nous avons vu précédemment que les fichiers sources fournis par l'utilisateur étaient interprétés par un navigateur web. Ces derniers permettant tous le support du *Web Storage*¹⁰, nous avons décidé de mettre en place cette notion. Les avantages que l'on peut espérer en tirer sont la facilité de la mise en place d'un tel système (interface Javascript simple d'utilisation et indépendante du navigateur web). L'inconvénient majeur est que l'espace de stockage mis à disposition est limité à quelques méga octets, et donc n'est pas utilisable avec la manipulation d'importante banque de données.

Afin de rendre intuitif la manipulation des données par l'utilisateur, nous décidons d'utiliser une librairie Javascript : *js-models*. Cette dernière permet l'abstraction des données par la représentation en objet de celle-ci. De plus, cette librairie a la particularité intéressante d'implémenter directement le *Web Storage*.

9. appelé aussi le *DOM Storage*

10. Liste des navigateurs qui assurent le support du *Web Storage* : [http ://dev-test.nemikor.com/web-storage/support-test/](http://dev-test.nemikor.com/web-storage/support-test/)

Généralement les modèles gèrent des données envoyées via un formulaire d'une page html. Nous arrivons cependant aux limites du langage HTML. Il faut pouvoir analyser les informations transmises par le formulaire et cela ne peut pas se faire en langage HTML. Mais nous n'avons pas de serveur donc nous sommes obligés de transmettre le formulaire vers une page HTML. Nous devons donc intercepter l'envoi de tous les formulaires afin de sérialiser les informations. La sérialisation permet ainsi de passer les données du formulaire, via l'url, à la nouvelle page html. Le script javascript qui se charge d'analyser l'url détectera que nous avons à faire à une transmission de formulaire en prenant en compte la méthode employée, c'est à dire GET ou POST. Il fournira ensuite en paramètre de l'action les données du formulaire. Pour indication, dans l'usage habituelle, GET et POST permettent de transmettre de façon différentes les informations. La méthode GET emploie la même méthode que nous, c'est à dire sérialise les informations pour les transmettre via l'url. En revanche, pour la méthode POST, les données sont transmises de manière cachées et donc de manière plus sécurisée. Ainsi, notre solution utilisera une méthode GET quelque soit la méthode demandée par le développeur. Cela est donc un énorme point faible de cette solution qui supprime donc l'usage de la méthode POST et ceux sans que le programmeur ne s'en aperçoive.

L'ensemble de nos propos est concrétiser au travers d'un mini framework annexe et indépendant appelé *Zigzag.js*, qui est le javascript chargé d'analyser l'url, appelé les contrôleurs et action...

Maintenant que nous disposons d'une structure complète des *Bundles*, nous devons nous demander comment interconnecter ces *Bundles*.

L'interconnexion

L'application ,développé par le développeur web, sera appelé application composite en analogie avec la figure 5 (application créée à partir de la connexion de plusieurs *Bundles*). L'application composite doit avoir connaissance de tous les *Bundles* qu'elle doit interconnecter. Dans ce but, nous créons un fichier *bundles.js* qui va permettre au programmeur web de déclarer tous les *Bundles* qu'il souhaite connecter, nous appelons cela l'enregistrement des composants. Pour enregistrer un *Bundle*, le développeur doit indiquer deux choses. Premièrement, il doit donner un nom arbitraire qui lui permettra d'identifier le *Bundle* dans toute l'application. Ensuite, il doit indiquer le chemin où se trouve le répertoire du composant. Ainsi, le chargement du fichier *bundles.js* par *Zigzag.js* permettra d'enregistrer l'ensemble des composants et de les identifier avec un nom propre à chacun. Pour interconnecter, nous allons créer un nouveau fichier *connections.js*, où le développeur va indiquer les connexions qu'il désire. Deux informations sont nécessaires. Il doit indiquer la sortie et l'entrée qu'il veut connecter.

Les entrées se définissent grâce à deux critères : l'identifiant du *Bundle* et l'identifiant d'entrée. C'est à la charge du programmeur du *Bundle* de définir cette entrée. Cette dernière doit être déclarer dans le fichier *inputs.js*. Ce fichier doit être créé dans chaque *Bundle* permettant ainsi de définir les entrées propre à chaque composants. Il permet au développeur d'associer un identifiant arbitraire

à une route.

Les sorties se définissent comme les entrées, c'est à dire un couple nom *Bundle* et identifiant de sortie. En revanche, la déclaration de l'identifiant de sortie diffère par rapport aux entrées. Brièvement, lorsque le développeur veut faire une sortie du *Bundle*, il doit fait appel à une fonction en passant en paramètre l'identifiant de la sortie et optionnellement un ensemble de données. Cette fonction ce charge de prévenir l'application composite que le *Bundle* courant désire effectué une sortie avec tel identification et tel données. Une fois averti, l'application composite utilise l'entrée associé à cette sortie et transmet les données à l'entrée concerné.

Pour déclarer le *Bundle* initial, c'est à dire le premier *Bundle* appelé par l'application. Il faut connecter l'entrée de ce *Bundle* avec une sortie NULL. Ainsi l'application détectera que c'est le *Bundle* initial. Mais pour faire cette détection au démarrage, il est nécessaire de lancer au premier coup un script qui se charge de cette vérification. Pour cela, un fichier *index.html* est placé dans l'application composite. Ainsi pour démarrer l'application il suffit de cibler via l'url ce fichier qui s'occupera de lancer le premier *Bundle*.

Diagramme de séquence

Résumons l'ensemble du déroulement de notre système au travers d'un scénario simple. Le démarrage d'une application composé uniquement d'un *Bundle*.

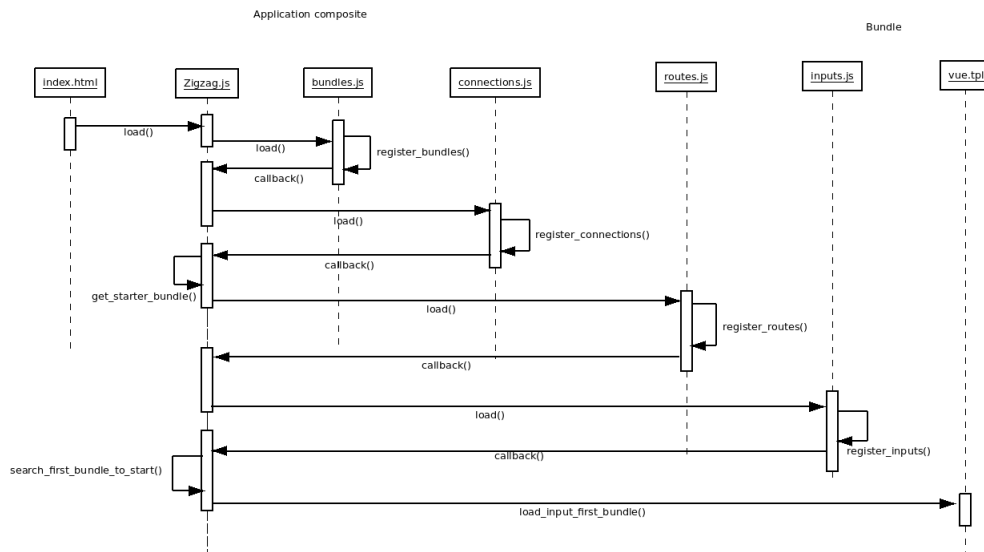


FIGURE 4.13: Diagramme de séquence : Le chargement du Bundle initial

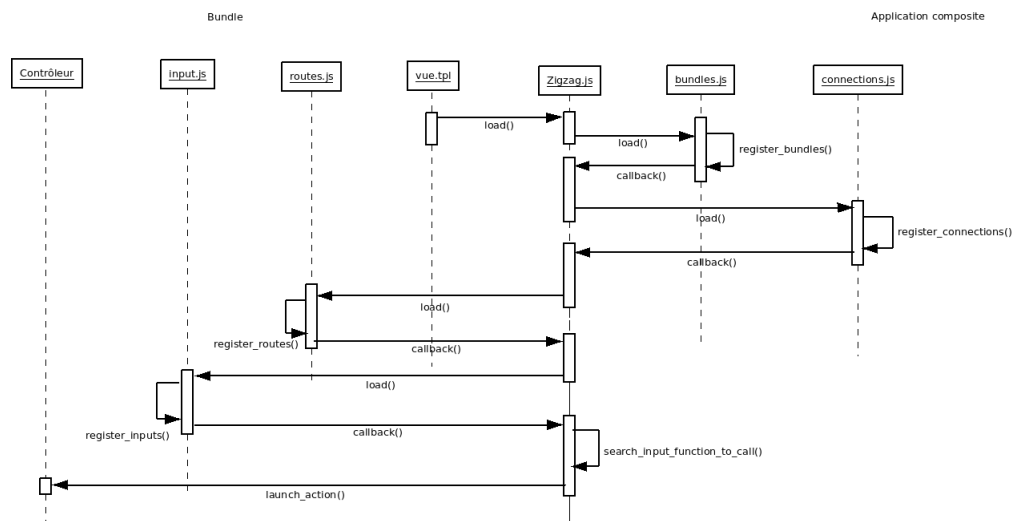


FIGURE 4.14: Diagramme de séquence : Le chargement d'une vue

Chapitre 5

Réalisation

5.1 Organisation

Dans un premier temps, nous avons réfléchis et discutés ensemble de la conception et l'analyse du projet. Nous nous sommes fixés un planning sous forme d'un diagramme de Gantt afin de structurer nos travaux et de répartir les tâches à effectuer. De cette manière, nous gardons un oeil sur l'échéance du projet pour ne pas se faire surprendre par le temps. Le diagramme de Gantt est disponible dans l'annexe C.

Les tâches sur les différentes plateformes mobiles ont été réparties de la sorte :

Chakour Pharès : Android

Mathieu Alexandre : Firefox OS

Vanderchmitt Bastien : Windows Phone

5.2 Développement

Dans cette partie, nous allons aborder les aspects techniques du projet ; c'est-à-dire les tenants et aboutissants de la programmation. Comme vu précédemment, nous nous sommes répartis les tâches à effectuer sur les différents OS entre les trois membres du groupe. Nous allons donc expliquer en détails en quoi consiste réellement ce travail en décomposant par système d'exploitation.

5.2.1 Android

Pour développer sous Android nous utilisons l'environnement de développement Eclipse ainsi que le plugin ADT¹. Nous avons décidé d'assurer le support des versions à partir de la 4.0 (Ice Cream Sandwich), car certaines fonctionnalités de l'API de ces dernières nous sont indispensables. En effet des problèmes sont

1. Android development tools

survenus lorsque nous avons mis en place les bundles et le *Web Storage* dans les versions inférieures à la 4.0.

5.2.2 Firefox OS

Firefox OS est un système d'exploitation pour smartphones récent. Il se base exclusivement sur des technologies web (HTML5, Javascript et CSS3).

Le système n'est pas encore disponible au grand public, c'est pourquoi nous utilisons un émulateur fourni par la Mozilla Fondation. Cet émulateur fonctionne directement depuis le navigateur web Firefox sous la forme d'un add-on²

Les applications pour Firefox OS sont écrites en HTML/Javascript. Cela a pour effet de nous simplifier énormément les différentes implémentations qui sont plus délicates sur Android et Windows Phone. En effet, nous n'avons pas à implémenter de moteur de rendu, puisque le HTML/Javascript possède un rendu natif sur cette plateforme.

L'étape de la génération d'exécutable est beaucoup plus légère dans ce cas, car les sources fournies par l'utilisateur utilisent déjà le langage natif de la plateforme. Ainsi, il nous suffit juste d'associer les différentes sources avec un fichier de configuration spécifié par la plateforme. Nous obtenons alors une application Firefox OS prête à être installée sur un appareil mobile.

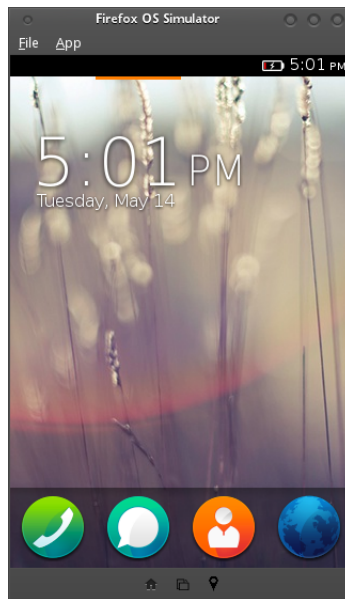


FIGURE 5.1: Emulateur Firefox OS utilisé pour le développement

2. Un add-on dans le cadre de Firefox est un petit programme se greffant au navigateur

5.2.3 Windows Phone

Le développement sur cette plate-forme se résume en deux parties, avec d'un côté un fichier XAML et de l'autre un fichier écrit en C# (abréviation pour C sharp). Le fichier XAML, qui est un dérivé du XML, permet au développeur d'écrire des interfaces ; c'est-à-dire de créer, positionner et paramétrer les éléments graphiques de l'application. Ipso facto, le fichier XAML contient des balises qui décrivent ce qui doit s'afficher sur l'écran du téléphone. Une zone de prévisualisation, appelée concepteur ou plus souvent designer en anglais, est disponible sur Visual Studio en parallèle de l'édition du XAML ce qui facilite grandement la programmation. En effet, de cette manière pas besoin de lancer un émulateur pour visualiser à quoi ressemble l'application. On peut voir en direct ce qu'on réalise. Chaque fichier XAML possède un code-behind (code derrière en anglais) qui est en fait le fichier écrit en C#. Ce fichier permet de gérer toute la logique associée au XAML. En effet, c'est dans ce fichier que sont implémentés les gestionnaires d'événements, et toute les actions que le programmeur souhaite réaliser qui ne peuvent pas être décrites par le fichier XAML. Une des caractéristiques les plus appréciées du développement en XAML est de pouvoir séparer la partie visible par l'utilisateur (environnement graphique) du code en lui-même.

Notre application est constituée d'un unique composant qu'est le « Web-Browser ». Celui-ci permet d'héberger des pages Web et autres documents compatibles avec un navigateur. Dans notre cas, le WebBrowser est invisible pour l'utilisateur grâce à la balise XAML `Visibility="Collapsed"`. Au lancement de l'application, le pseudo-navigateur charge le fichier html que l'utilisateur du framework a développé. Ensuite, de la même façon que sur Android, le fichier HTML appelle un script Javascript qui parse la page html et permet à l'application d'afficher en natif le code écrit en HTML5/CSS. Cette partie est rendue possible par la communication entre le code C# et Javascript via la méthode `window.external.notify`.

Chapitre 6

Résultats et Discussion

6.1 Gestionnaire de version

Nous avons hébergé notre framework sur la plateforme Bitbucket qui propose un service basé sur mercurial pour le gestionnaire de version, les liens vers les différents dépôts sont les suivants :

Rizla : <http://bitbucket.org/poulp/rizla> (Framework)

Rizla Android : <http://bitbucket.org/poulp/rizla-android> (Portage sur Android)

Zigzag : <http://bitbucket.org/pchakour/zigzag.js> (Framework javascript)

Documentation : <http://rizla.readthedocs.org/en/latest/> (Documentation en ligne)

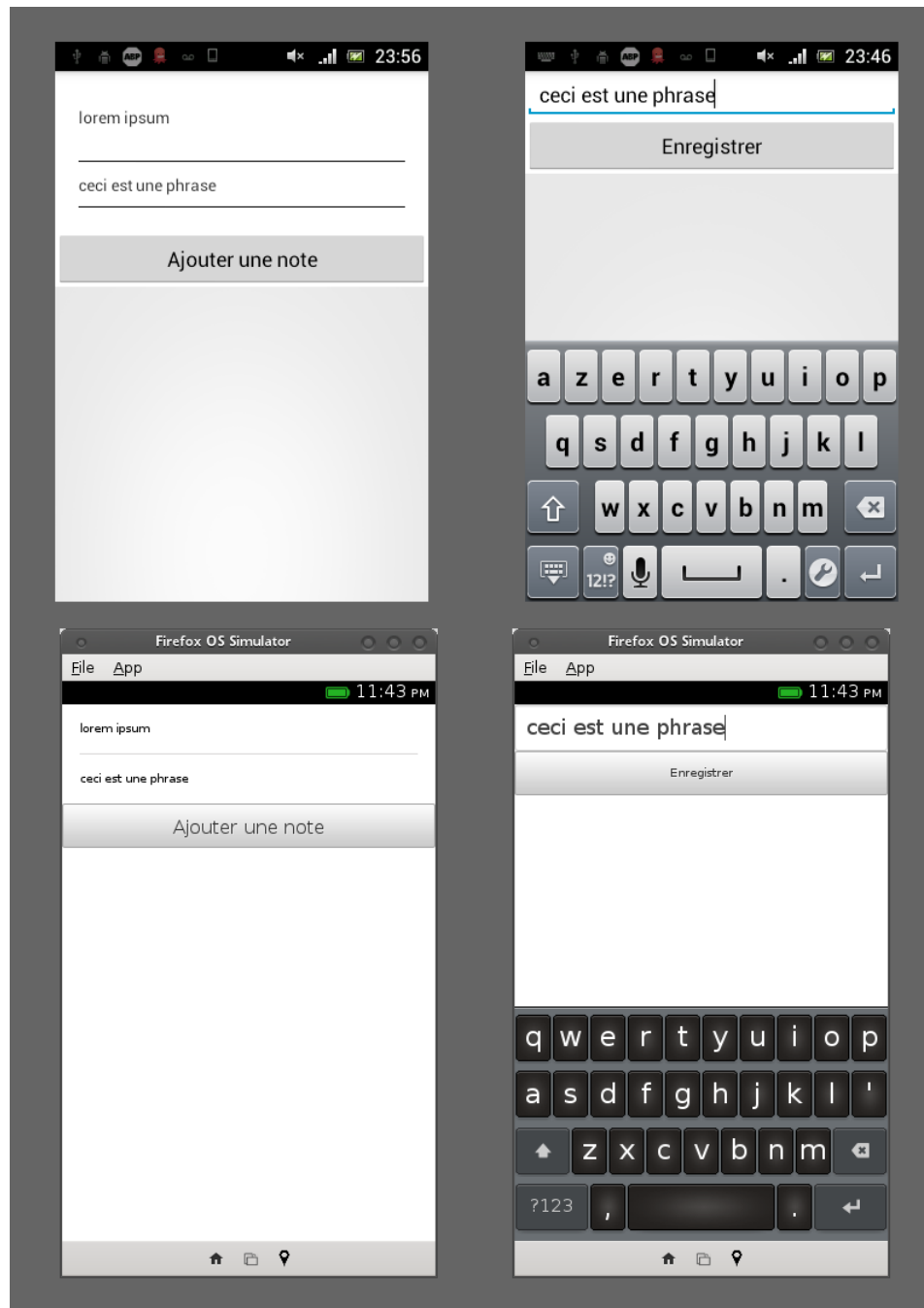
6.2 Développement d'une application mobile : *storage*

Nous allons dans cette partie écrire une petite application que nous installerons sur les plateformes Android et Firefox OS (Windows Phone n'étant pas suffisamment avancé). L'utilisateur rentre des mots via une entrée texte sur la première page qui s'affiche. Ces mots sont stocké dans le *Web Storage* et devront être affiché sur une autre page sous forme d'une liste.

Le code source de cette application est disponible dans l'annexe C. La figure qui suit représente les deux pages de l'application affichées sur les différentes plateformes mobile.

6.3 Objectifs

L'objectif de cette partie est de constater si le travail effectué respecte bien les termes du cahier des charges. C'est-à-dire prendre du recul et voir si les buts que



nous nous étions fixés ont été atteints. Nous commencerons donc par remarquer que le framework est opérationnel pour les plate-formes Android et Firefox Os. Malheureusement, en raison de problèmes survenus pendant le développement et du manque de temps, la partie sur Windows Phone n'est pas terminée mais nous reviendrons plus en avant sur ce point. Le cahier des charges n'est donc pas entièrement respecté mais il l'est pour les deux systèmes d'exploitation que sont Android et Firefox Os. En effet, pour chaque plate-forme il est possible de : créer et développer une application mobile, générer des exécutables, déboguer un projet, et ce à partir d'un même code source HTML5/CSS/Javascript.

6.4 Viabilité du framework

Nous pouvons tout d'abord nous demander pour la conception de quel type d'application le framework est-il une solution. En effet, si nous prenons le cas des jeux vidéo, ce sont des programmes qui demandent beaucoup de ressources de la part de l'appareil mobile. Le temps de chargement d'une application et son interprétation dans le cadre de notre framework ne nous permettent pas d'envisager le développement d'un tel projet. Nous pouvons dire que ce framework se destine plus à faire tourner des applications peu gourmandes en ressources.

6.5 Windows Phone

Revenons maintenant sur la partie du développement concernant la plate-forme Windows Phone. En effet, des problèmes se sont manifestés lorsque nous avons dû incorporer au projet les nombreux fichiers Javascript nécessaires à l'analyse syntaxique de la page HTML. Après de fastidieuses recherches, il s'avère que pour pouvoir faire appel à ces fichiers, il faut les placer dans une zone du terminal nommée Isolated Storage. En effet, pour des raisons de sécurité, l'exécution des applications Silverlight se fait dans un environnement très restreint (impossible d'accéder à la base de registre, au système de fichiers, etc.). Le sdk de Windows Phone nous offre donc un emplacement pour stocker des informations sous forme de textes ou de fichiers : c'est "l'Isolated Storage" ou stockage isolé. Or, pour pouvoir ajouter nos fichiers Javascript à cet Isolated Storage de façon générique, nous avons besoin de lancer notre application en FullTrust (confiance totale). Une application basique est réalisée en PartialTrust (confiance partielle). Nous avons alors tenté de donner les pleins pouvoirs à notre framework en passant par deux utilitaires en lignes de commandes : Strong Name Tool (sn.exe) pour signer l'assembly avec un nom fort, opération nécessaire à l'utilisation de Caspol.exe (Outil Code Access Security Policy Tool) qui permet aux administrateurs de modifier la stratégie de sécurité au niveau de l'ordinateur et de l'utilisateur. Malheureusement, étant donné le manque de documentation sur le sujet, et le manque de temps, nous ne sommes pas parvenus à un résultat concluant.

Chapitre 7

Conclusion

A travers ce projet nous mettons en oeuvre un framework destiné à la production d'applications pour plateformes mobiles. Le framework s'identifie autour de trois mots clés : rentabilité, réutilisabilité et extensibilité.

Afin de réduire le coût de développement d'applications mobiles multi-plateformes, nous proposons des outils permettant d'automatiser certaines tâches comme la génération d'exécutables. Ainsi à partir d'un même code source, le développeur a la possibilité de redistribuer son application sur autant de plateformes mobiles que le framework le permet. De plus, en introduisant la notion de *bundles*, des bouts d'applications peuvent être partagés entre les utilisateurs. Ceci permet alors de ne pas reproduire un bout de code qui existe déjà.

Notre projet traite trois frameworks : Android, Firefox OS et Windows Phone. A travers ces différents systèmes, nous constatons qu'il n'est pas simple d'implémenter la construction de vue native à partir de fichiers HTML/-Javascript. En effet, les technologies et les architectures diffèrent d'une plateforme à l'autre. Nous mettons donc en place une application dotée d'un navigateur pour interpréter les fichiers sources fournis par le navigateur.

L'extensibilité du framework se traduit par un système de plugin permettant à des développeurs externes de pouvoir ajouter de nouvelles fonctionnalités. Ainsi nous assurons le support en s'adaptant aux technologies sans cesse en évolution sans nuire au code principal du framework.

Le choix du HTML/Javascript comme langage de conception dans ce framework n'est pas lié seulement au fait qu'il soit simple et intuitif à utiliser lors de la conception d'une application. Nous avons constaté que bon nombre de frameworks pour applications mobiles ont aussi fait ce choix. De plus de nouveaux systèmes d'exploitations pour smartphones rendent natif l'utilisation du HTML/Javascript (c'est le cas de Firefox OS et Ubuntu Phone). Cette attirance pour les technologies web, dans le milieu du mobile, nous pousse à nous demander si ces dernières ne deviendraient pas le nouveau standard de développement d'applications dans les années à venir.

Bibliographie

- [Bal12] Aral Balkan. Mobile considerations in user experience design : "web or native?". *Smashing Magazine*, 2012.
- [GEK88] Stephen T. Pope Glenn E. Krasner. A cookbook for using the model-view-controller user interface paradigm in smalltalk-80. 1988.
- [LFH10] Christophe Dony Luc Fabresse, Noury Bouraqadi and Marianne Huchard. Component-oriented programming : From requirements to language support. 2010.

Annexe A

Liste des technologies utilisées dans le projet

jQuery

<http://jquery.com/>
Framework Javascript

handlebars.js

<http://handlebarsjs.com/>
Moteur de templates en javascript.

crossroad.js

<http://millermedeiros.github.io/crossroads.js/>
Librairie de routage en javascript.

js-models.js

<http://benpickles.github.io/js-model/>
Pour représenter la partie des modèles dans l'architecture MVC.

Python 2.7

<http://www.python.org/>
Pour les outils de générations.

Jinja2

<http://jinja.pocoo.org/docs/>
Moteur de templates en python.

Sphinx

<http://sphinx-doc.org/>
Pour générer la documentation.

Latex

Pour rédiger le rapport.

Annexe B

Code source du compteur

```
<html>
  <head>
    <title>Compteur</title>
  </head>
  <body style="background-color:#DCDCDC">
    <div class="app" style="text-align:center">
      <p id="number">0</p>
    </div>
    <div style="text-align:center">
      <button id="plus">+</button>
      <button id="reset">Reset</button>
      <button id="moins">-</button>
    </div>
    <script type="text/javascript" src="js/jquery-1.9.1.min.js"></script>
    <script type="text/javascript">
      app.initialize();
      $('#plus').click(function(){
        $('#number').text(parseInt($('#number').text()+1);
      });
      $('#moins').click(function(){
        $('#number').text(parseInt($('#number').text()-1);
      });
      $('#reset').click(function(){
        $('#number').text('0');
      });
    </script>
  </body>
</html>
```

Annexe C

Diagramme de Gantt

[illegible]

Annexe D

Code source de l'application *storage*

Tout d'abord nous présentons la structure de l'application. Les fichiers qui nous intéressent sont en gras et détaillés dans la suite de l'annexe.

```
application/  
-- lib/  
-- config/  
---- bundles.js  
---- connection.js  
---- config.cfg  
-- ressources/  
-- src/  
---- index.html  
---- bundles/  
----- Rizla/  
----- StorageBundle/  
----- input.js  
----- route.js  
----- models/  
----- Note.js  
----- controllers/  
----- IndexController.js  
----- views/  
----- Index/  
----- index.tpl  
----- add.tpl
```


D.1 bundle.js

```
Z.bundle.addBundle("storage", "Rizla/StorageBundle");
```

D.2 connection.js

```
Z.bundle.connection(null,["storage","listInput"]);
```

D.3 input.js

```
Z.bundle.addInput(  
  "listInput", /* nom de l'entrée  
  "index" /* nom de la route */  
);
```

D.4 route.js

```
Z.route.addRoute(  
  'index', /* nom de la route */  
  'Index', /* nom du controleur */  
  'index' /* nom de la vue */  
);  
Z.route.addRoute('add', 'Index', 'add');
```

D.5 IndexController.js

```
Z.controller.IndexController = {  
  indexAction : function(){  
    Z.importModel('../models/Note.js');  
  
    var notes = Note.all();  
  
    Z.view = {  
      notes : notes,  
    };  
  },  
  
  addAction : function(){  
    Z.importModel("../models/Note.js");  
  
    $("#save").click(function(){  
      var test = $("#note").val();  
      if (test != ""){
```

```

        var tmp = new Note({content:test});
        tmp.save()
        Z.route.load("index");
    }
    });
},
}
}

```

D.6 Note.js

```

var Note = Model("note",function(){
this.persistance(Model.localStorage);
});

```

```

Note.load();

```

D.7 index.tpl

```

<html>
<head>
<script type="text/javascript" src="../../../../lib/zigzag.js"></script>
</head>
<body>
<script type="text/x-handlebars">
<section data-type="list">
<ul>
{{#each notes}}
<li><p>{{this.attributes.content}}</p></li>
{{/each}}
</ul>
</section>
</script>
<a class="button" href="#add">Ajouter une note</a>
</body>
</html>

```

D.8 add.tpl

```

<html>
<head>
<script type="text/javascript" src="../../../../lib/zigzag.js"></script>
<head>
<body>
<input type="text" id="note">

```

```
        <button id="save">Enregistrer</button>
    </body>
</html>
```