Tutoriel

I) Application

a. Notre produit :

Vous êtes sériephile et vous avez tendance à oublier la diffusion de votre série préférée du moment? Vous croulez sous les séries à quel épisode suis-je? Vous ne devez surtout pas oublier le commencement de cette nouvelle série? Si l'une de ses difficultés se présente WYSIWYA (= What You See Is Where You Are) est la solution!!!

Cette application propose des notifications en temps et en lieu. En effet, vous enregistrer votre série, en déterminant le lieu où vous souhaitiez regarder la série ainsi que l'horaire. Puis l'application vous le rappellent. Ainsi, en fonction de votre position GPS et de la date de programmation une notification vous est adressé. En interne, un compteur incrémente l'épisode auquel vous en êtes et vous l'affiche sous forme d'une barre de progression pour savoir l'avancement d'une série. Il y a possibilité de personnaliser l'image représentant la série en affichant l'affiche officielle de la série, une image de sa galerie ou encore d'une photo prise via la caméra du dispositif. Ainsi, vous pourrez capturer les affiches des nouvelles séries pour penser à les visionner dès leur sortie.

b. Notre objectif :

Notre objectif est un comparatif des fonctionnalités de périphériques mobiles au travers d'un langage natif (Android) et cross-platform (Ionic). Les fonctionnalités suivantes on été étudier : géolocalisation, caméra, notification, Splashscreen.
Dans cette application nous ne nous préoccupons en aucun cas du stockage des données en utilisant une base de données ou du stockage local.

II)Android

a. Présentation :

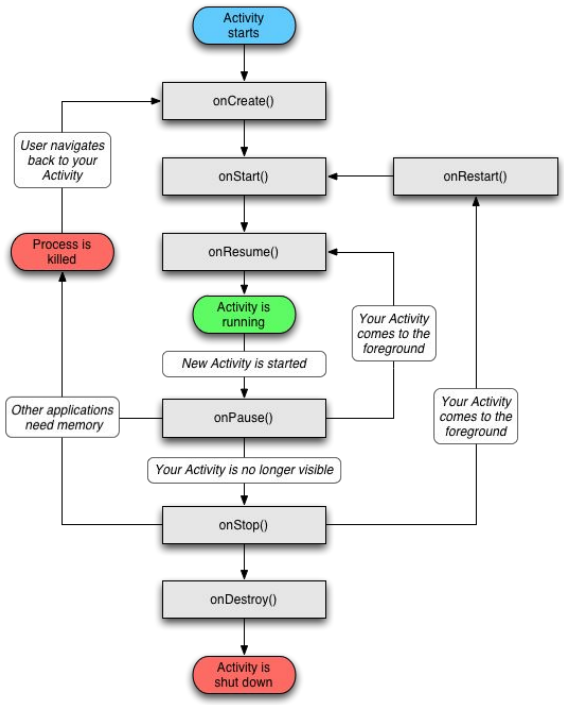
Android, est un système d'exploitation mobile open source basé sur le noyau Linux et développé actuellement par Google. Le système a d'abord été conçu pour les smartphones et tablettes tactiles, puis s'est diversifié dans les objets connectés et ordinateurs comme les télévisions (Android TV), les voitures (Android Auto), les ordinateurs (Android-x86) et les smartwatch (Android Wear). Il s'appuie sur le langage de Programmation Orienté Objet Java.

b. Installation :

Il nous faut installer la JDK (=Java Development Kit) 6 ou supérieur, en effet, Android repose sur Java en amont. Ensuite, on installe la SDK (=Software development kit) d'Android et IDE Android Studio qui est l'IDE de prédilection dans le monde Android.

c. Paradigme processus application Android :

Avant de programmer notre projet, il est essentiel de comprendre le cycle de vie très particulier d'une application Android :



onCreate

Elle sert à initialiser notre activité ainsi que toutes les données nécessaires à cette dernière.

onStart

Cette méthode est pour signifier le début du passage au premier plan de notre application. En cas de problème, l'activité sera transférée à OnStop.

onResume

Cette méthode est appelée au moment où votre application repasse en foreground. À la fin de l'appel l'application se trouve au premier plan et reçoit les interactions utilisateurs.

onPause

Si une autre activité passe au premier plan, la méthode onPause est appelée sur notre activité. Il nous faut sauvegarder l'état de l'activité et les différents traitements effectués par l'utilisateur.
À ce stade, votre activité n'a plus accès à l'écran, vous devez arrêter de faire toute action en rapport avec l'interaction utilisateur (désabonner les listeners).

onStop
Appelée quand notre activité n'est plus du tout visible quelque soit la raison. Dans cette méthode nous nous devons d'arrêter tous les traitements et services exécutés.
onDestroy

Appelée pour terminer le processus. Toutes les données non sauvegardées sont perdues.

d. Le projet :

d.1 Initialisation :

Nous initialisons le projet en utilisant l'une des UI proposé par Android : Navigation Drawer Activity proposant un ensemble de layout. Nous ajoutons une classe MainActivity héritant de ActivityGroup qui définit la structure de l'application générale contenant l'ensemble des éléments graphiques dans un ArrayList<View> ainsi que les évènements rattachés.

Le MainActivity lance des activités comme la liste des séries enregistrées, la géolocalisation et la notification quand l'application est lancée.

Concernant la mise en place des vues nous devons instancier un objet de type View et l'afficher dans une interface graphique. Par défaut, nous nous retrouverions devant un carré blanc qui mesure 100 pixels de côté. Pas très glamour, j'en conviens. C'est pourquoi, quand on crée une vue, on doit jouer sur au moins deux tableaux : les dimensions de la vue, et son dessin.
Nous construisons une interface graphique représentée sous le format XML. Dans le contexte du développement d'interfaces graphiques pour Android la racine du XML sera très souvent un layout. Ce layout peut avoir des enfants, qui seront des widgets ou d'autres layouts.
Tous les éléments sont identités et retrouvés dans le code par un identifiant unique.

De ce fait, pour lancer des activités il faut les déclarer dans l'AndroidManifest.xml comme :
```
<activity android:name="com.example.smallbirdking.seriess.Serie" />
```

Pour les services c'est aussi dans l'AndroidManifest.xml qu'ils sont déclarés nous ajoutons des licences pour permettre le portable d'utiliser des services : GPS, Album et caméra comme :
```
<uses-permission android:name="android.permission.INTERNET"></uses-permission>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
```

d.2 Ajout de série:

Nous réalisons une nouvelle activité nommée Serie qui va requêter auprès de l'API gérant les séries. Cette API retourne du format JSON. Nous obtenons le résultat de la requête Json par url à l'aide du service AsyncHttpClient. Nous utilisons une bibliothèque externe JsonFormat qui est un parseur permettant de convertir facilement une chaîne de caractères JSON en une classe objet.
Après quoi, nous établissons une classe Adapter qui implémente BaseAdapter récupérant les données qui nous intéressent issue de l'API dans des variables. Ensuite, nous ajoutons une classe List qui est une simple vue chargée de l'affichage sous forme de liste des séries.

Pour charger les images récupérées par url "https://image.tmdb.org/t/p/w300/image_name" fourni par l'API, nous utilisons un objet Picasso fourni par un package Android (com.squareup.picasso.Picasso;). Nous chargeons l'image à l'aide de la méthode Picasso.with(context).load(imageURL).into(imageView);

d.3 Chargement photo/image | caméra.

Plus précisément :

Notre classe Serie contient un SimpleAdapter permettant de réaliser diverse opérations.
Lorsque vous cliquez sur l'icone plus notre évènement : gridView1.setOnItemClickListener () est activé permettant l'appel à des albums locaux afin de parvenir aux images stockées dans notre “gallerie” accessible via pathImage de chaîne.

Enfin, nous chargeons l’image et rafraichissons la vue par onResume GridView setAdapter () et notifyDataSetChanged ().

Concernant la partie camera, il existe un service prédéfinit :
Intent cameraintent = new Intent(MediaStore.*ACTION_IMAGE_CAPTURE*);
startActivityForResult(cameraintent,*CAM_REQUEST*); // active la caméra.

d.4 Notification alert

Dans le but de tester la notification, on ajoute une check-box dans la vue Serie afin de définir où la notification aura lieu. Pour chaque série selon les informations de la check-box sera enregistré dans des listes différentes représentant une position par liste (Cf classe Notification).

Nous ajoutons un bouton “at home” pour simuler notre position.

Pour réaliser une notification, on utilise des services préexistant issue du package :
import android.support.v4.app.NotificationCompat;
Ce package contient une classeNotificationCompat permettant une gestion facilitée des besoins d’une notification.
Ainsi pour instancier une notification le schéma typique est comme suit :
on ajoute une intent permettant d'être rediriger vers la fiche de la série aux cliques sur la notification.
Une notification contient toujours un unique id représenté par l’identifiant de la série issue de l’API. Ainsi, elles sont toutes stockées dans une liste statique pour pouvoir y accéder.

d.5 Géolocalisation :

Pour comprendre le fonctionnement de la géolocalistation nous créons une activité getLocation qui import le package android.location.
Ensuite on utilise un Listener pour surveiller notre localisation contenant une fonction :
(LocationManager)getSystemService(Context.LOCATION_SERVICE) permettant de récupérer notre position actuelle convertit en LocationManager pour un accès facilité.
updateView se charge d’afficher auprès de la vue la lattitude et longitude.

Pour plus de détails sur le projet vous pourrez retrouver le code sur le git :
https://github.com/smallbirdking/android-series

III)Ionic

a. Présentation :

Ionic est un framework cross-plateform embarquant plusieurs technologies :
-Angular pour la partie front-end avec son design pattern MVVC il s'utilise au travers de contrôleurs, services, directives et filtres.
-Cordova pour le build et la publication de notre application sur les différents OS du marché (l'application générée embarque une webview pour permettre l'exécution de notre application).
-Sass un pré-processeur CSS disponible et activable.
-Gulp Task Runner permettant d'automatiser toute une série de taches (fusion CSS, minification…).
-Bower pour gestion des dépendances front-end de notre projet notamment les bibliothèques JS comme Angular.

Tout est pré configuré pour travailler ensemble offrant un gain de temps certains. Gulp dispose du livereload permettant d'actualiser automatiquement le navigateur dès la détection d'une modification dans le code source.

b. Installation :

Ionic s'installe au travers de npm (=Node Package Manager) le gestionnaire de paquets de Node.js à l'aide de la commande : "*sudo npm install ionic -g*"
L'option "*-g*" permet d'installer en global Ionic.

Ensuite, on initialise notre application : "ionic start *name_project* blank" (blank permet de partir from scratch dans l'application sans design particulier).

On souhaite utiliser Sass bien que l'application souhaitée n'en ait pas réellement besoin c'est juste pour l'utilisation de cette technologie. Pour pouvoir utiliser sass qui est à la base un préprocesseur de Ruby il faut l'indiquer à Ionic via : "*ionic setup sass*".(doit disposer du gulp en global : "*sudo npm install -g gulp"* pour justement pouvoir lancer Sass a chaque modification du code)

Par défaut le projet dispose d'un module Angular chargeant la bibliothèque Ionic. Il utilise le module ui-router contrairement au ng-route par défaut fourni par Angular. C'est un ng-route plus poussé proposant système d'état.

Nous créons un projet vide afin de partir from scratch sans vue préétablie. Pour cela on exécute commande : "*ionic start* WYSIWYA *blank*"
Enfin, on lance le serveur JS en effectuant un : "*ionic serve*" à partir de là on possède une instance de notre application depuis notre navigateur web.

c. Le projet :

Nous présentons quelles solutions utilisées pour développer à bien notre application et ceux de manière chronologique afin de comprendre comment répondre aux besoins.

c.1 Structure et routing du projet

Nous prenons le choix d'une interface simpliste avec essentiellement deux parties :
-"home" qui fera office de dashboard de rappel des séries suivies en indiquant leur progression.
-"serie" pour rechercher une série et paramétrer ses besoins de rappel.
Une troisième partie setting sera proscrite dans notre exemple en étant directement simulé (comme enregistrer des lieux, temps de notification…).

On peut structurer nos fichiers Angular dans le dossier www/js/ comme suit : app.js à la racine puis des dossiers :
-directives : C'est la solution retenue dans la V1 d'Angular pour doper notre HTML de balise et propriété. Typiquement c'est ce qui se rapproche du concept de Web Component que le W3C est en train de mettre en place. Cependant, ce système devrait disparaître avec Angular V2.
-filters : pour réaliser des filtres sur le contenu à afficher (typiquement pour notre dashboard on pourrait l'utilisateur aurait pu filtrer par genre, date de notification, avancé dans la série…).
-controllers : chaque contrôleur embarque la partie métier de notre application traitant les données d'une fonctionnalité majeure de l'application. Il est important d'avoir un bon découpage car chaque contrôleur dispose d'un scope(porté) propre à lui-même c'est une notion clé le scope et l'isolation en JS pour bien appréhender ses comportements. Ainsi, chaque contrôleur a son propre scope donc ses propres fonction et variables et reste hermétique au reste de l'application.
-services : Il existe trois formes de services (factory, services et provider) cela peut être perçu comme la couche "model" permettant de récupérer des données de divers serveurs/sources.
-tests : contenant les fichiers de tests.

Il en résulte deux contrôleurs Angular et trois routes à charger :
-"home" (homeController.js) comme route "/home"
-"serie" (serieController.js) : recherche de série comme route "/serie" et paramétrage/ajout d'une série dans le système de notification "/serie/{id:[0-9]{1,8}}" le Regex permet de reconnaître le paramètre id comme valide si c'est un numérique d'aux plus 8 chiffres.

Ainsi dans le fichier chargeant notre module Angular app.js on ajoute une configuration en chargeant ui-router pour pouvoir faire notre routing :

```
.config(function($stateProvider, $urlRouterProvider){
 $stateProvider.state('home', {
            url : '/home',
            templateUrl : 'templates/home.html',
            controller : 'homeController'
 })
 $stateProvider.state('serie', {
            url : '/serie',
            templateUrl : 'templates/serie.html',
            controller : 'serieController'
 })
 $urlRouterProvider.otherwise('/home');
});
```

Ici chaque état est définie à l'aide de $stateProvider.state nommant l'état puis le template (vue html à charger) et le contrôleur Angular qui s'occupe de la partie logique de cette vue. Enfin, la fonction otherwise permet de rediriger automatique vers la home l'état par défaut en cas de route non existante.

c.2 Mise en place des vues :

Ionic dispose de directive proposant notamment des capacités de gestion de l'historique (pushState). Ceci permet l'édition des menus et headers de façon simple. Toutes les vues au changement de template HTML sont chargées à l'aide de la balise "<ion-nav-view>" qui est un wrapper de la balise Angular "<view>".

De plus il dispose d'une micro-bibliothèque CSS permettant d'afficher nos données en respectant les problématiques des appareils mobiles (et permettant un affichage paysage comme portrait).

Dans notre cas on définit un menu utilisant la directive <ion-nav-bar class="bar-positive"> là classe permettant de définir le menu en top bar avec une couleur prédéfinie ici positive. Il existe plusieurs couleurs que l'on peut modifier depuis sass en écrasant la variable contenant le code couleur.

Chaque template HTML chargé contiendra :

```
<ion-view view-title="Home">
  <ion-content>
  </ion-content>
</ion-view>
```

Ainsi, le contenu sera chargé à l'intérieur de la balise "<ion-nav-view>" en effectuant un appel Ajax. La propriété "view-title" de "<ion-view>" s'affichera dans la "<ion-nav-bar>" dans le but de renseigner la personne.

Ainsi avec ses propres directives Ionic embarque de la valeur métier offrant un gain de temps et surtout des interfaces pensées pour les mobiles.

c.3 Rechercher une série :

Nous allons réaliser le template de la recherche de série depuis une API : themoviedb.
pour cela on créer un fichier contrôleur dans le JS notre serieController.js :

```
$scope.search = function(serie) {
        $ionicLoading.show({
        template: 'Search...'
        });
        Series.getSearchSeries(serie).then(function(data){
                $scope.series = data.results;
                }, function(msg){
                alert(msg);
        });
        $ionicLoading.hide();
}
```

Pour requêter auprès de l'API nous utiliserons les services d'Angular. Il existe trois types de services Angular :
-factory qui permet de retourner directement les données réceptionnées depuis l'API.
-un service retourne une fonction plutôt que les données directement donc on a un objet JS en cas d'information de l'API en un format particulier.
-Et enfin, les providers retournent résultat d'une fonction ainsi il est possible d'effectuer des petites opérations avant de retourner le résultat aux contrôleurs.
L'API dispense ses données sont en JSON (=JavaScript Object Notation) c'est donc supporté nativement en JS d'où l'utilisation d'une factory "Serie".

Ainsi, notre factory possède une fonction où l'on transmettra notre query de recherche de série directement à l'API :

```
app.factory('Series', function($http, $q) {
	var factory = {
	series : false,
	serie : false,
	getSearchSeries : function(query){
		var deferred = $q.defer();
		$http.get("http://api.themoviedb.org/3/search/tv?api_key=61f7950a0c9e1089cf27fbcc524ec7db&language=fr&query=" + query)
		.success(function(data, status) {
			factory.series = data;
			deferred.resolve(factory.series);
		}).error(function(data, status) {
			deferred.reject('Erreur requete Ajax');
		});
		return deferred.promise;
	}
	return factory;
})
```

Cela retourne un ensemble de série. Une fois une série sélectionnée au touch ou clique sur <a ui-href=...""> on change d'état.
À l'aide du $watch on peut ajouter un évènement sur une variable qui dès sa modification lance une fonction. C'est exactement ce que l'on souhaite récupéré plus d'informations sur une série dès que l'URL contient l'id de la série : $scope.$watch($stateParams.id, setSerie);

De ce fait, on ajoute une fonction à la factory "Series" qui requête sur l'API pour obtenir tous les détails d'une série :

```
	getDetailSeries : function(id){
		var deferred = $q.defer();

		$http.get("http://api.themoviedb.org/3/tv/" + id + "?api_key=61f7950a0c9e1089cf27fbcc524ec7db&language=fr")
		.success(function(data, status) {
			factory.serie = data;
			deferred.resolve(factory.serie);
		}).error(function(data, status) {
			deferred.reject('Erreur requete Ajax');
		});
			return deferred.promise;
		}
	};
```

c.4 Ajout d'une série et utilisation ngCordova :

Une fois les détails de la série chargée on ajoute un formulaire permettant de : modifier image de l'affiche, paramétrer date de notification et choisir le lieu de notification.
Nous allons donc utiliser les fonctions natives de l'appareil cible tournant sous Android. Pour cela, il existe ngCordova une extension Angular permettant d'interagir directement sur les fonctionnalités du dispositif. Ainsi, ngCordova est une surcouche cordova reposant sur l'architecture Angular.
On l'installe dans notre projet à l'aide de bower : "bower install ngCordova" il reste plus qu'à inclure le .js dans index.html.

c.4.i Photo et Caméra :

Pour prendre une photo depuis la caméra on charge le module $cordovaCamera dans notre contrôleur Angular.(serieController).
$cordovaCamera.getPicture(options).then(function(imageData) {});

-options objet contenant notamment la propriété : “sourceType” permettant de déterminer si l’on souhaite une photo depuis la caméra via : Camera.PictureSourceType.CAMERA.
-imageData contenant le path vers l’image c’est donc assez simple de prendre une photo. “then” permet d’être utilisé une fois l’action réaliser c’est basé sur la technologie des promesses de JS offrant ainsi un callback fort pratique. Il suffit donc d’exécuter cette fonction au clique sur un bouton pour cela il y a la propriété ng-click d’Angular et l’on peut ajouter les icons en utilisant la banque d’icon proposé par défaut dans Ionic.

Pour ajouter une image depuis les fichiers de notre OS c’est exactement la même fonction on modifie simplement “sourceType”de l’objet option passé en paramètre de la fonction en utilisant la constante : Camera.PictureSourceType.PHOTOLIBRARY.

c.4.ii Formulaire datePicker :

Ensuite, on souhaite déterminer une date de départ des notifications. Nous pourrions utiliser les champs date fourni par HTML5. Afin, d’être plus interactif j’opte pour inclure un datePicker. ngCordova offre un datePicker mais nous préférons utiliser des directives Angular proposées par la communauté. Angular a le mérite d’avoir un communauté importante ainsi j’utilise ce git : https://github.com/rajeshwarpatlolla/ionic-timepicker couplé avec celui-ci https://github.com/rajeshwarpatlolla/ionic-datepicker
installable de nouveau avec Bower.

Pour la validation du formulaire on bénéficie du databinding permettant d’avoir des retours en temps réel à l’utilisateur sur les données saisies.
Tout d’abord ajouter un novalidate sur la balise <form> pour qu’Angular et la main sur la soumission du formulaire.
Ensuite, il suffit d’ajouter une propriété name à chaque input pour pouvoir associer des règles de validation fournie par Angular sur l’input directement :<input type="number" min="0" name="episode" max="{{ serie.seasons[0].episode_count }}" ng-model="serie.episode" ng-value="0" >

Ainsi, on rajoute des ng-show pour afficher les éventuelles erreurs si elle existe :

```
<span class="assertive" ng-show="formSerie.episode.$error.min || formSerie.episode.$error.max">
        The value must be in range 0 to {{ serie.seasons[0].episode_count }}!
</span>
```

Là encore on utilise les classes prédéfinies par Ionic pour afficher un code couleur des erreurs rouge pour error,,orange pour notice, vert pour success.
Sur le bouton submit est apposé la propriété : ng-disabled="formSerie.$invalid" grise le bouton de soumission indiquant à l’utilisateur que le formulaire n’est pas correctement rempli.

Enfin un ng-submit retourne l’ensemble des informations saisi par l’utilisateur dans une fonction du contrôleur. Pour enregistrer la série sans disposer de stockage local ou distant nous la conservons en mémoire. Pour que la série soit accessible au travers de toute l’application nous l’ajoutons au $rootScope qui est le scope parent à tout contrôleur permettant de partager la série au travers de l’application.

c.4.iii Notification Locale :

Une fois enregistrer on utilise le module $cordovaLocalNotification à inclure dans notre contrôleur. Cela permet d'effectuer des notifications localement sur l'appareil. Et c'est d'une puissance et simplicité consternante! En effet, on peut ajouter, supprimé, modifier une notification mais aussi la personifier : icon, titre, contenu, data…

Le plus utile pour notre cas et la fonction : $cordovaLocalNotification.schedule({}); permettant de programmer une notification toute les X unité de temps. Enfin, on utilise un écouteur pour incrémenter le compteur de l'épisode courant : $rootScope.$on('$cordovaLocalNotification:schedule', function(event, notification, state) {});
Il y a quelques mois de cela cette écouteur alors que présent dans la documentation officielle ne fonctionnait pas il fallait l'implémenter soi-même. Cela prouve que ngCordova est encore une technologie jeune mais qu'elle avance très rapidement grâce à une communauté oeuvrant tous les jours.

c.4.iiii Géolocalisation :

Enfin, il faut envoyer la notification que si on se trouve aux alentours des lieux auxquels on souhaite regarder notre série. Pour cela dès l'initialisation de l'application il faut enregistrer les coordonnées GPS. Par défaut, il prendra les coordonnées de l'école pour le test s'il ne détecte aucun lieu préétablis.

De nouveau on utilise un module $cordovaGeolocation avec la fonction :
$cordovaGeolocation.getCurrentPosition(posOptions).then(function (position) { });
Très simple et intuitif on obtient la latitude et longitude dans l'objet position.

Le reste n'est plus que du code pour faire tourner l'application et ne requiert en aucun cas une description.
Vous pourrez retrouver le code sur le git : https://github.com/GuillaumeUnice/WISIWYA

d. Tester l'application :

d.1 Test unitaire :

Pour les tests unitaires on peut utiliser n'importe quelle framework de test JS Mocha, nunit, qunit, nodeunit…
Jasmine est tout de même le Framework utilisant le paradigme Behavior Driven Development pour tester nos applications Angular.
À cela on installe Karma pour pouvoir lancer des navigateurs web et effectuer les tests depuis la console de manière automatisée. Karma s'installe à l'aide :
"*npm install karma --save-dev*" et "*npm install -g karma-cli*" pour disposer ligne de commande.
Ensuite, on effectue la commande "*karma init*" pour configurer avec quel navigateur tester, ou se situe les fichiers de tests… Enfin, on exécute "karma start" qui lancera le navigateur et affichera dans la console le résultat des tests.

d.2 Test fonctionnelle :

Pour effectuer des tests fonctionnels de notre application plusieurs solutions subsistent.

Tout d'abord, au travers du navigateur web dans ce cas on ne pourra en aucun cas tester les fonctionnalités natives des dispositifs mobiles (accéléromètre, notification…).
Ensuite, il y a Ripple c'est une extension navigateur (encore en version Beta) permettant de simuler au travers du navigateur les fonctionnalités de l'appareil (accéléromètre, géolocalisation...). Cependant, il est encore instable et ne propose pas toute les fonctionnalités natives (comme la notification dans notre cas).

Pour ces deux procédés pas besoin de builder l'application car on est encore sur de la technologie web uniquement. On dispose de la console des navigateurs web pour les erreurs qui malheureusement pas très explicite. Comme Ionic effectue pas mal de tâche avec Gulp les lignes d'erreurs correspondent rarement ainsi que l'erreur elle-même reste souvent abstraite.

Après, il y a les solutions avec build donc pour Android on effectue la commande : "ionic build android" qui est juste un wrapper de la commande cordova pour effectuer la compilation en Android.

Une fois le build réalisé on peut utiliser IonicView c'est une application pour tablette ou mobile permettant de récupérer depuis les serveurs Ionic un projet ainsi on envoie notre projet à l'aide de la commande : "*ionic upload*". Puis, on réceptionne sur notre application IonicView notre projet en effectuant une synchronisation. Cela utilise un compte email avec mot de passe pour pouvoir récupérer et uploader le projet.

La dernière des solution est de connecter notre mobile au PC une fois le périphérique détecté (éventuellement installer les pilotes) on lance : "adb install -r path_project.apk" le projet Android obtenu après build qui se trouve dans le dossier /plateforms du projet.

Conclusion:

Pour conclure, Android et Ionic sont deux technolgies très disparates mais dont le but est commun construire des applications mobiles. Toutes deux bénéficient d'une importante communauté.
Cependant, les services Android sont globalement un peu plus fournies offrant plus de possibilités bien que cela se restreint aux mobile Android.
Avec Ionic, on vise le multiplateforme, mais dès lors que l'on sort d'Andoid et IOS il se pose des problèmes de compatibilité.

Avec Android, on est sûr d'une évolution pérenne tandis que Ionic reposant sûr de nombreux frameworks encore jeune voir instable peu se heurter a des problèmes d'évolutivité. Le passage d'Angular V1 vers la V2 risque de se réaliser dans la douleur.

Quoi qu'il en soit on ne peut pas aborder Android sans connaissance Java et on ne peut pas aborder Ionic sans connaissance web.

Si l'on désire une application Android très fine nous devons passer par de l'Android.
Si l'on souhaite du cross-plateform Ionic est une solution mais pas la seule, en effet, elle oblige l'utilisateur à recourir à certains frameworks (Angular, gulp…). Ainsi il peut être plus judicieux selon les connaissances et besoins de s'orienter vers du From Scratch avec un Phonegap où l'on pourra par dessus y inclure les frameworks souhaités.