**MoviApp**

**Preparado para:**

Alta Consejería de las TIC’s

Martes 19 de septiembre de 2017

Versión 1.0

**Preparado por:**

Lantri S.A.S

Bogotá D.C. – Colombia

**Manual de Técnico**

Tabla de Contenido

[Objetivo 3](#_Toc493512796)

[Arquitectura principal 3](#_Toc493512797)

[Plataformas 7](#_Toc493512798)

[Notificaciones 7](#_Toc493512799)

[Plugins 8](#_Toc493512800)

[Mapa 10](#_Toc493512801)

## Objetivo

El presente documento tiene como objetivo dar a conocer los componentes principales en la creación de MoviApp. De manera general, se presentará la arquitectura utilizada por MoviApp en su interfaz gráfica, aplicación móvil y lógica de negocio en el WebAPI, diagramando técnicamente cómo funciona y está construido el software en su conjunto.

## Arquitectura principal

MoviApp está construida sobre la tecnología de Apache Cordova 7.0.1 utilizando el framework Ionic versión 3.5.0, por lo tanto, los principales lenguajes de desarrollo son JavaScript con el framework Angular Js, HTML 5 y SCSS principalmente.

MoviaApp tiene 9 módulos generales los cuales son: Inicio, Perfil, Vehículos, Mis sitios favoritos, Rutas, Buscar viaje, Historial de viajes, Bicimovilidad, y Tutorial. En cada uno de ellos y dependiendo, el caso se hace uso de los componentes proveídos por Ionic, además de, plugins adicionales que se describirán posteriormente en el documento.

Igualmente, se implementaron cuatro providers y una directiva, componentes que funcionan transversalmente en los módulos y que son utilizados para funciones específicas y repetitivas a lo largo del aplicativo.

**Módulos:**

* **Inicio:** Es la vista por defecto de la aplicación, cuando el usuario ya se ha registrado o iniciado sesión. Se verán las primeras cinco notificas del portal de movilidad de bogotá <http://www.movilidadbogota.gov.co/web/historico_noticias>, con una imagen previa, título, una breve descripción de la noticia y un enlace para visitar directamente la página.
* **Pérfil:** En la vista pérfil, el usuario podrá ver su nombre, cédula, email, celular y una foto que lo describa, igualmente él podrá cambiar cada uno de estos datos a través de los botones **Cambiar Foto** y **Editar perfil.** Estas acciones son de tipo POST y la imagen será guardado en un blob de 4 megabytes.
* **Vehículos:** El módulo de vehículos presenta, en modo de lista, los vehículos que tiene el usuario, a través de una llamada GET al métodoGetListVehicleName del API. Igualmente, hay dos botones descriptivos para eliminar o editar cada vehículo y un botón en la parte inferior de la lista que nos lleva a la función de crear un nuevo vehículo.

Para crear un nuevo vehículo, se deben tener las restricciones básicas de una placa correcta para el territorio colombiano, formato AAA000, el modelo y el cupo son campos a elegir y el color como la marca, tienen validaciones de longitud y caracteres especiales.

* **Mis sitios favoritos:** Similarmente a los vehículos, la sección de mis sitios favoritos presenta a manera de lista cada uno de los sitios creados con el nombre, la descripción y la dirección inicial, adicionalmente cada sitio tiene los botones de borrar o editar. En la parte inferior de la lista, se presenta el botón de crear un nuevo sitio, allí para la presentación del mapa se utilizará la directiva **my-google-maps**, descrita más adelante.
* **Rutas:** La sección de rutas tiene tres submodulos.
  1. **Crear ruta:** Para todos los formatos de creación se utiliza FormBuilder, componente de la biblioteca angular/forms. En la creación de una ruta se necesita tener previamente un vehículo, un nombre y como mínimo un punto inicial y final. Utilizando los servicios del API de direcciones de Google se traza la ruta. La ruta se actualiza cada vez que haya un cambio en los puntos, ya sea en los puntos base (inicio y fin), o en los puntos intermedios.
  2. **Mis rutas:** Se listan las rutas que han sido creadas por el usuario, con información básica del nombre, estado e identificador de la ruta. El botón **más**, nos permite borrar la ruta unicamente cuando la ruta está en estado 0 que significa creada (existen 3 estados más, activada, en curso y borrada, con 1, 2, 3 respectivamente en la base de datos), o por el contrario nos permite ver el detalle de la ruta, donde se crea un vista a modo de modal y se puede ver toda la información de la ruta, activar o desactivarla.
  3. **Rutas suscritas:** Se listan las rutas en las cuales soy pasajero, esto quiere decir que hay una suscripción activa y no se es propietario de la ruta. De manera similar a Mis rutas, se listan las rutas y se tiene un botón **más**, donde nos permite ir a la ruta, unicamente cuando está en estado 2 (en curso) o cancelar suscripción, la cual generará una notificación al dueño de la ruta.
* **Buscar viaje:** Para buscar viaje no se puede tener una suscripción activa, eso quiere decir que el usuario no puede buscar un viaje, en el momento en el cual pertenece a una ruta que está pronto a iniciar o que va en curso. Aquí se tienen dos formas de buscar:
  1. **Buscar por nombre:** Se escribe el nombre de la persona que es propietaria de la ruta, y se listan las rutas que están en estado 1 (activa) y donde el nombre del propietario, dado la siguiente relación, genere verdadero: movcom\_usuario.usuario\_nombre like ‘nombre sin espacios’.
  2. **Buscar por GPS:** El usuario elige dos puntos cualesquiera en el mapa, y obtienendo las rutas que están en estado activo, se verifica que el punto inicial esté en un circulo de 100 metros y el punto final esté aproximadamente a 500 metros. Esto se logra, utilizando el servicio de google maps <https://maps.googleapis.com/maps/api/directions/json>? Para crear una consulta con los puntos de la ruta, y obtener el poligono correspondiente.Para verificar la cercanía de los puntos se utiliza la librería de geometería de google maps con el método isLocationOnPath.

Una vez obtenidas las rutas, el usuario debe elegir dos puntos cercanos a la ruta, dónde se va a subir y dónde se va a bajar, la cercanía se verifica con el mismo método isLocationOnPath pero con solo 40 metros de distancia al polígono generado.

* **Historial de Viajes:** Para el historial de viajes se mostrarán todos los trayectos del usuario, con la suma total de pasajes de SITP que ha libreado y el total de los kilometros. Cuando el usuario completa 121,6km de movilidad compartida aparecerán los árboles recuperados. Ya que cada 121,6km se considera que se ha dejado de emitir suficiente CO2 como haber plantado un árbol.
* **Bicimovilidad:** La bicimovilidad contará con tres submodulos que hacen un simil con los viajes en carro descritos anteriormente.
  1. **Mapa bicimovilidad:** Muestra un mapa subrayando las posibles rutas en bicicleta que están implementandas en la ciudad de Bogotá. Se genera con poligonos que están fijos y se van procesando sobre el mapa de google maps a través de un objecto Polygon.
  2. **Biciviaje:** El usuario podrá empezar un biciviaje, en el cual se suscribe un evento que actualiza el viaje y va trazando la ruta del viajero, cuando el usuario determine finalizar su viaje, con la trayectoría trazada se creara un nuevo biciviaje.
  3. **Historial de viajes:** Se listan los viajes hechos por el usuario, con la funcionalidad de biciviajes, se hace la suma correspondiente del kilometraje en cada ruta, este kilometraje se obtiene a través de los servicios de google en el momento en que se traza la ruta.
* **Tutorial:** Son dos slider que tienen información pertinente para los nuevos usuarios. Una vez pasado por el tutorial, en el localstorage del aplicativo se guarda una variable que indica que no se debe volver a pasar por allí a menos que el usuario lo quiera, para ello, tiene que ir directamente por el tab de tutorial del menú.
* **Cerrar sesión:** Guarda en el localstorage una variable de unregister para que una vez iniciado el aplicativo tenga que volver a crear o registrarse como usuario.

**Providers:**

* **Network Service:** Utiliza el servicio de Network por parte de la biblioteca ionic-native/network, con la cual se puede generar una suscripción al evento de conexión o desconexión por parte del usuario. Este servicio es utilizado con frecuencia en todas las vistas que necesitan verificar si existe una conexión a internet para poder mostrar el contenido, editarlo o borrarlo.
* **Notification:** Este provider es utilizado para registrar el dispositvo, a través de una consulta POST al método RegisterDevice, donde se necesitan tres parametros, el email del usuario registrado, el deviceId que se obtiene utilizando la biblioteca ionic-native/push y el registrationId que está guardado en el local storage de cada aplicativo.

Adicionalmente, el provider notification es utilizado para abrir todas las notificaciones que llegan,tiene la lógica de decisión switch case, conde cada caso corresponde a un tipo de notificación específica. El tipo de notificación está en los datos adicionales de la notificación.

* **Photo:** Este provider es utilizado para obtener una foto ya sea desde la cámara o desde la librería del celular, implementa el plugin Camera de ionic-native/camera y a través de eventos asincronos, devuelve la ruta de donde se encuentra la foto en caso de que sea la librería o una ruta temporal, en caso de que la imagen sea tomada directamente desde la cámara. Cabe notar que, se configuró el plugin para que las fotos tomadas no queden guardadas en la galería del celular.
* **Web-API:** A través de este provider se crean todos los métodos GET y POST que tiene la aplicación, los parámetros que se envían al API van en formato x-www-form-urlencoded. Para el caso de las peticiones GET, se envían en forma ordenada con separación de **/** para cada parametro.

En los métodos GET se implementó un chaché temporal, de esta manera si el usuario no tiene conexión a internet pero previamente hizo una consulta al mismo contenido, se mostrarán esos datos, en caso contrario, el usuario siempre verá los datos actualizados.

Por otro lado, en caso de que un usuario quiera cambiar una foto, el procesamiento de la consulta no sigue el mismo patrón de GET y POST descrito anteriormente, allí se utiliza el componente Transfer de ionic-native/transfer, donde a través de una consulta por POST y en modo de streaming, se envía la foto a un método del API, para que haga el cambio respectivo. De esta manera, el envío de una foto no sobrecarga la red del usuario, ni bloquea la aplicación esperando a que la consulta termine.

**Directivas:**

* **My-google-maps:** Es un componente personalizado, el cual se puede adicionar como una propiedad de html a cualquier input en las vistas de la aplicación, de esta manera el input se modifica, de tal manera que al hacer click, abrirará una vista modal con un mapa desplegaedo, donde el usuario podrá seleccionar una locación específica que será retornada a manera de arreglo por la vista modal. La primera componente, tiene la latitud del punto, la segunda componente la longitud, en la tercera componente está la dirección del punto y por último, en la cuarta posición está el index del punto para los puntos intermedios de la ruta, ( esta última posición solo se utiliza en casos especiales y puede funcionar como comodín para nuevas implementaciones).

Para que la directiva funcione dentro de las vistas, se debe adicionar el componente SharedModule en el módulo de cada vista. Cabe recordar que cada vista tiene 4 archivos, la vista **.html**, sus estilos **.scss** , un módulo **.module.ts** y un la lógica o controlador .**ts**

**MoviAPI**

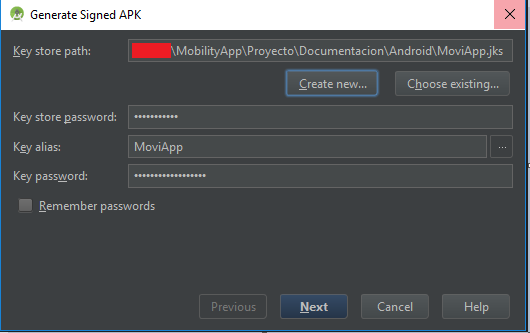
El API está desarrollado en el framework de PHP CodeIgniter, en el cual se tienen acceso a la base de datos, y se maneja la arquitectura de API RESTful, en donde los controladores acceden a los datos a través de los Models, de los cuales hacen la conexión y el manejo de los datos según corresponda su funcionalidad. Estos controladores, siempre responden con true, false o un archivo .**json** según sea el caso.

Los controladores que se implementaron están seccionados por los componentes descritos anteriormente y a su vez los modelos siguen el mismo patrón.

## Plataformas

MoviApp está construida para dos plataformas, tanto Android como iOS, con versión 6.2.3 y 4.3.1 respectivamente. Los archivos **.apk**  y **.ipa**  son generados por el ambiente de Ionic al ejecutar ionic cordova build Android y ionic cordova build ios, ya sea el caso.

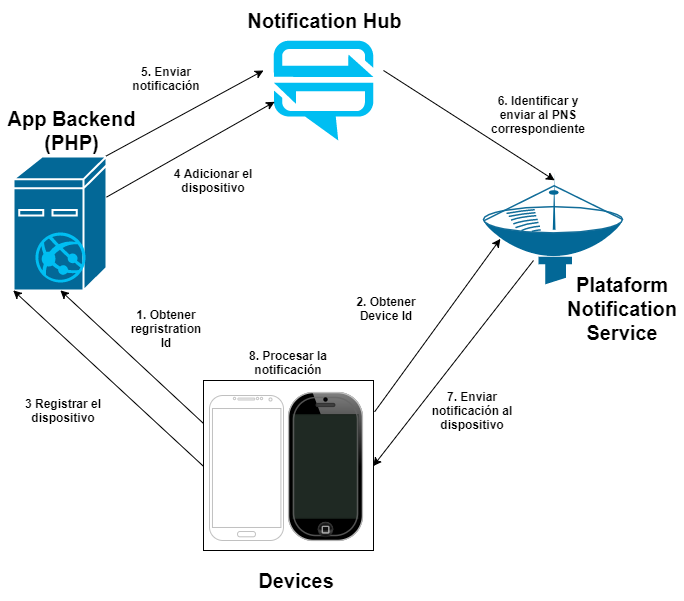
Para Android el archivo. **jks** o llave con la que se firman las versiones está ubicada en la carpeta MobilityApp > Proyecto > Documentación > Android. Para el correcto uso de esta llave, se necesitan dos claves, la primera de ellas es: **MoviApp2017** y la segunda es **AltaConsejeria2017** con el Alias MoviApp.



Para iOS se necesita el ambiente de desarrollo Xcode 9 el cual da soporte a la versión de iOS 11. Para la creación de nuevas versiones **.ipa** se debe firmar con la cuenta de la Alta Consejeria de las TIC’s ([altaconsejeriadetic@alcaldiabogota.gov.co](mailto:altaconsejeriadetic@alcaldiabogota.gov.co) ), archivar el código y subirlo a [itunesconnect.apple.com](http://www.itunesconnect.apple.com)  a través de Xcode.

## Notificaciones

Las notificaciones que son usadas por MoviApp están basadas en el servicio de Azure Notification Hub, el cual permite una interacción rápida y fácil con los dispositivos luego de hacer una configuración correspondiente a cada plataforma. En el siguiente gráfico se explicará la arquitectura principal de este componente y su funcionalidad.



En este gráfico tenemos los 8 pasos que corresponde a la creación, envío y procesamiento de las notificaciones.

* 1. Para la configuración de un dispositivo en el Notification Hub se necesita un ID por parte del notification y otro parte del PNS, para que se puede identificar cada celular. En el primer paso a través del método CreateDevice, el dispositivo solicita un RegistrationID al API Backend.
  2. Con el plugin Push Notification, se envía una petición de registro al servidor PNS correspondiente para cada plataforma (Android y iOS utilizan PNS diferentes). Teniendo los dos ID’s Registration ID y Device ID vamos al registro en el Notification Hub.
  3. Se genera la solicitud al API Backend a través del método RegisterDevice del API.
  4. El dispositivo adiciona el nuevo celular al Notification Hub, con el cual podrá referirse para mensajes. En este punto el registro del dispositivo finalizó.
  5. Cuando inicia el proceso de enviar las notificaciones, el API Backend inicia el proceso llamando al Notification Hub, enviándole el parámetro de tipo de plataforma (Android o iOS), el mensaje y el dispositivo identificado con los ID’s y el correo del propietario.
  6. El Notification Hub identifica el PNS correspondiente y traduce el mensaje enviado desde el API Backend al formato correcto del PNS, para que sea enviado a los dispositivos.
  7. El PNS identifica el mensaje, el dispositivo y la aplicación correspondiente, para enviar una notificación push.
  8. A través del método OpenNotification en el provider Notificacion se hace el procesamiento de las diferentes notificaciones que se implementaron en la aplicación.

Para la configuración correcta de un Notification Hub en Android se siguió la guía:

<https://docs.microsoft.com/en-us/azure/notification-hubs/notification-hubs-android-push-notification-google-fcm-get-started>.

Para la configuración de iOS:

<https://docs.microsoft.com/en-us/azure/notification-hubs/notification-hubs-ios-apple-push-notification-apns-get-started>

Para configurar el backend con lenguaje PHP se remitió a:

<https://docs.microsoft.com/en-us/azure/notification-hubs/notification-hubs-php-push-notification-tutorial>

## Plugins

Para la construcción de MoviApp se necesitaron de varios complementos de cordova los cuales son listados en la siguiente tabla

|  |  |  |  |
| --- | --- | --- | --- |
| **Nombre** | **Descripción** | **Fuente** | **Licencia** |
| cordova-plugin-tts | Permite leer textos a través del dispositivo | <https://github.com/vilic/cordova-plugin-tts> | MIT License |
| cordova-plugin-device  cordova-plugin-device-motion  cordova-plugin-device-orientation | Permite saber la orientación del dispositivo, y genera el evento en caso de que ésta cambie. | <https://github.com/apache/cordova-plugin-device>  <https://github.com/apache/cordova-plugin-device-motion>  <https://github.com/apache/cordova-plugin-device-orientation> | Apache 2.0  Apache 2.0  Apache 2.0 |
| cordova-geolocation | Permite tener la geolocalización del dispositivo. | <https://github.com/apache/cordova-plugin-geolocation> | Apache 2.0 |
| Cordova-inappbrowser | Permite embeber una página web a través del iframe. | <https://github.com/apache/cordova-plugin-inappbrowser> | Apache 2.0 |
| Cordova-plugin-network-information | Permite conocer el estado de conexión del dispositivo a internet. | <https://github.com/apache/cordova-plugin-network-information> | Apache 2.0 |
| Cordova-plugin-x-socialsharing | Permite compartir un texto o link a través de las redes sociales ligadas a nuestro dispositivo. | [https://github.com/EddyVerbruggen /SocialSharing-PhoneGap-Plugin](https://github.com/EddyVerbruggen/SocialSharing-PhoneGap-Plugin) | MIT License |
| cordova-plugin-file-transfer  cordova-plugin-file-transfer  cordova-plugin-file | Permite la transferencia de la imagen a través de streaming al servidor PHP | <https://github.com/apache/cordova-plugin-file> | Apache 2.0 |
| cordova-sqlite-storage | Permite generar una base de datos locales donde estarán almacenadas todas las variables locales | <https://github.com/litehelpers/Cordova-sqlite-storage> | MIT |
| phonegap-plugin-push | Se utiliza para obtener el deviceId y suscribirse a los eventos de registro y notificacion | <https://github.com/phonegap/phonegap-plugin-push> | MIT |

Adicional a los complementos mencionados anteriormente, por defecto el proyecto tiene otros plugins que no han sido utilizados en el proyecto, pero que por tema de escalabilidad y disponibilidad se han decidido dejar. Estos plugins adicionales los podemos ver en la siguiente lista, donde todos tienen licencia Apache 2.0:

* cordova-plugin-media-capture
* cordova-plugin-media
* cordova-plugin-file
* cordova-plugin-media
* cordova-plugin-compat
* cordova-plugin-whitelsit
* cordova-plugin-console
* ionic-plugin-keyboard
* cordova-plugin-splashscreen
* cordova-plugin-splashscreen

## Mapa

El procesamiento de los mapas en la aplicación, fue soportado por el servicio de google maps llamando al API a través de la siguiente dirección <http://maps.google.com/maps/api/js?key=AIzaSyClxKEmmoBA_x3w2iBvccqVTJiDqBa99Fg&libraries=geometry,places>

Como se puede evidenciar, la clave utilizada que permite procesar los mapas tanto Android como para iOS es, **AIzaSyClxKEmmoBA\_x3w2iBvccqVTJiDqBa99Fg**