UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE SISTEMAS INFORMÁTICOS

TRABAJO DE FIN DE MÁSTER

MÁSTER EN DESARROLLO DE APLICACIONES Y SERVICIOS PARA DISPOSITIVOS MÓVILES

DESARROLLO DE UN ASISTENTE VIRTUAL PARA DISPOSITIVOS MÓVILES EMPLEANDO IBM WATSON

Autor: Pablo Martín del Campo

Tutor: Francisco Serradilla García

Curso 2018-2019

Resumen

El gran desarrollo tecnológico que se está produciendo estos últimos años, y en especial, con la aparición de los primeros *smartphones* y la posterior evolución de estos, está propiciando un cambio radical en el uso y la interacción de los usuarios con las distintas aplicaciones. Mientras que hace unos años, la gran parte de las aplicaciones desarrolladas eran aplicaciones web o de escritorio, actualmente, como consecuencia de que la gran mayoría de la población posee un *smartphone*, y la creación de varios *frameworks* que simplifican la implementación de nuevo software para dispositivos móviles, se ha producido un importante aumento del desarrollo de aplicaciones móviles.

Debido a este cambio reciente en la interacción de los usuarios con las aplicaciones, se ha decidido desarrollar una aplicación móvil que integrará un asistente virtual que será capaz de entender el lenguaje natural, respondiendo a las preguntas formuladas por los usuarios, manteniendo una conversación como si se tratara de una persona humana, así como buscar información o realizar ciertas operaciones. Para esto se usará el framework para el desarrollo de aplicaciones híbridas, Ionic y los servicios proporcionados por IBM Watson.

Palabras clave

IBM Watson, asistente virtual, aplicación móvil, Ionic

Abstract

The great technological development that is taking place in recent years, and especially with the appearance of the first smartphones and the subsequent evolution of these, is leading to a radical change in the use and interaction of users with the different applications. While a few years ago, most of the applications developed were web or desktop applications, currently, as a result of most of the population owning a smartphone, and the creation of several frameworks that simplify the implementation of new software for mobile devices, there has been a significant increase in the development of mobile applications.

Due to this recent change in user interaction with the applications, it has been decided to develop a mobile application that will integrate a virtual assistant who will be able to understand the natural language, answering the questions asked by the users, maintaining a conversation as if it were a human person, as well as seeking information or performing certain operations. For this, the framework for the development of hybrid applications, Ionic and the services provided by IBM Watson will be used.

Keywords

IBM Watson, virtual assistant, mobile app, Ionic

Índice

Tabla de ilustraciones

Estudio teórico

1. Introducción

Como se ha comentado antes, el desarrollo de aplicaciones para dispositivos móviles ha avanzado mucho en estos últimos años debido, principalmente, al amplio mercado que conforman los usuarios de estos dispositivos, que actualmente es la gran mayoría de la población. También ha propiciado este desarrollo la aparición de nuevos lenguajes y herramientas, tanto nativas como híbridas y la mejora de los frameworks ya existentes, facilitando en gran medida la implementación de la estructura de las aplicaciones, de forma que es posible invertir más tiempo en el diseño y la lógica de nuestro proyecto, mejorando la calidad, la fiabilidad y la robustez del software.

Todo esto ha provocado a su vez el surgimiento de nuevos tipos de interfaces o formas de interactuar entre el usuario y la aplicación, desde las pantallas táctiles, el movimiento del dispositivo, más actualmente la voz del usuario y, probablemente, en un futuro las pantallas holográficas interactivas.

En el estudio teórico se tratarán algunos conceptos sobre los frameworks y herramientas que se han utilizado para el desarrollo de la aplicación, como Ionic o IBM Watson, y en la parte práctica se explicarán en detalle algunos de los pasos que se han llevado a cabo hasta la finalización del proyecto.

2. Objetivos

Debido a esto, el objetivo principal, dentro del marco de una gran evolución del mercado de desarrollo de aplicaciones móviles y de las formas de interacción humano-máquina, es el de permitir al usuario interactuar con una inteligencia artificial, lo más parecida al ser humano posible. De esta forma podría realizar diversas acciones mientras mantiene una

conversación de forma oral o escrita, que podría suponer un proceso más tedioso para el usuario en el caso de realizarlo con una aplicación móvil convencional, como podría ser hacer una compra en una tienda digital.

Otro objetivo más secundario, pero no por ello menos importante sería el de aplicar los conocimientos adquiridos sobre aplicaciones móviles en el máster y sobre Ionic, y adquirir nuevos conocimientos sobre Node.js, IBM Cloud y otras herramientas que se han usado, para construir una aplicación diferente a lo que se puede ver actualmente, pero que sigue la tendencia actual y futura.

3. Aplicaciones nativas e híbridas

Anteriormente, se han mencionado que existen herramientas para desarrollas aplicaciones nativas e híbridas, pero ¿qué significa que una aplicación sea nativa o híbrida? Para entenderlo mejor, se expondrán las principales características de estos tipos de aplicaciones, además de las aplicaciones multiplataforma o generadas y las aplicaciones web o *web apps* y las principales diferencias entre ellas.

Las aplicaciones nativas están desarrolladas específicamente para una plataforma en concreto, usando lenguajes específicos para esto, Objective-C o Swift para iOS y Java o Kotlin para Android. Este tipo de aplicaciones permiten tener el máximo rendimiento posible, minimizar el consumo de batería de los dispositivos, conseguir un aspecto visual más atractivo, acceder a todas las funcionalidades del dispositivo y proporcionar una mayor seguridad. Además, con desarrollos nativos es posible realizar aplicaciones para todo tipo de dispositivos como *Wearables* (Apple Watch, Android Wear), coches (Android Auto) o dispositivos IoT (Android IoT).

Las aplicaciones híbridas se desarrollan principalmente con JavaScript con el objetivo de poder ser ejecutadas en varias plataformas tanto móviles como de escritorio o navegador. Entre estas aplicaciones podemos diferenciar a su vez varios tipos:

- Las aplicaciones que utilizan un Webview o navegador nativo de la plataforma para ejecutarse utilizando HTML, CSS y JavaScript para presentar una interfaz de usuario que simula el aspecto nativo de cada sistema operativo. De esta forma, es posible ejecutar la aplicación, con mínimas diferencias, tanto en cualquier plataforma móvil, navegador o escritorio, aunque el rendimiento no sea tan bueno como en las aplicaciones nativas, y dependa más de la calidad del software, del hardware y la versión del sistema operativo. Frameworks que utilizan este enfoque son Cordova o Phonegap para aplicaciones móviles y de navegador, o Electron para aplicaciones de escritorio. Como ejemplo, la aplicación móvil la aplicación móvil que se usa en las tiendas de Diesel, se ha desarrollado usando Ionic, y Visual Studio Code con Electron, lo que indica que, aunque es más difícil, también es posible desarrollar aplicaciones de calidad con este tipo de frameworks.

- Por otro lado, existen otro tipo de aplicaciones híbridas que no usan un Webview para ejecutarse, como React Native, sino que utilizan directamente componentes nativos que se ejecutan dentro de una máquina virtual JavaScript. Aunque se sigue desarrollando con JavaScript, CSS y SGML, lenguaje parecido a HTML, las aplicaciones se comunican con el sistema operativo usando el motor V8 en Android y el motor nativo en iOS, lo que acercarse mucho más al rendimiento de las aplicaciones nativas, además de poder llamar directamente sin necesidad de plugins a cualquier funcionalidad del sistema operativo. A pesar de todo esto, debido al peso de la máquina virtual, no es aconsejable usar estas aplicaciones en dispositivos de gama baja o media, debido al elevado consumo de memoria y recursos. Un ejemplo de aplicación desarrollada con React Native sería la aplicación para dispositivos móviles de Facebook,

Las aplicaciones multiplataforma o generadas son aquellas que son desarrolladas usando lenguajes y herramientas específicas, pero que, a la hora de generar la aplicación, son compiladas a código nativo. Aunque se diga que el resultado de usar estos lenguajes son aplicaciones nativas, pero no ofrecen los mismos beneficios que las aplicaciones nativas desarrolladas específicamente para Android y iOS, por lo que también se les llaman falsas nativas.

Las *web apps* o aplicaciones web son aquellas que se basan en el uso de HTML, JavaScript y CSS, están adaptadas a dispositivos móviles y no es necesario instalarlas, ya que se ejecutan en un navegador. Como cualquier otra aplicación web, éstas correrán sobre un servidor web. Aunque son menos costosas de desarrollar, son las que peor rendimiento ofrecen de todas las mencionadas, además de no poder acceder a ninguna de las funcionalidades nativas de los dispositivos móviles. Otra de las ventajas es que estas aplicaciones son independientes de las plataformas y que son más fáciles de actualizar, puesto que el usuario no necesita instalar una nueva versión.

Partiendo de esta información, en la tabla siguiente se muestra una comparación entre las diferentes características de cada tipo de aplicaciones.

	Nativas	Híbridas	Generadas	Web apps
Rendimiento	Alto	Medio	Medio	Bajo
Coste desarrollo	Alto	Medio	Medio	Bajo
Diseño	Alto	Medio	Alto	Bajo
Acceso al dispositivo	Completo	Medio	Medio	Ninguno
Plataformas	Dependiente	Independiente	Independiente	Independiente

Como se puede ver, no existe ningún paradigma que sea mejor que los demás, la decisión de usar uno u otro depende del tipo de proyecto, o del equipo disponible para llevarlo a cabo. Por ejemplo, una gran empresa que busca la mayor captación y fidelización de sus clientes debería usar aplicaciones híbridas para transmitir sensación de calidad y rendimiento, mientras que una empresa más pequeña y que dispone de menos recursos le podría ser más útil una aplicación híbrida, para poder tener la aplicación en varias plataformas de forma más rápida y obtener mayor beneficio.

4. Apache Cordova

Apache Cordova es un framework open-source para el desarrollo de aplicaciones móviles híbridas. En un principio, Cordova fue la versión open-source del framework PhoneGap, aunque en la actualidad, ambos entornos son prácticamente iguales, salvo porque PhoneGap, que es la versión paquetizada de Adobe de Apache Cordova, provee servicios para compilar las aplicaciones sin tener que instalar las herramientas y SDK necesarios.

Las aplicaciones se desarrollan utilizando JavaScript, HTML5 y CSS3, permitiendo encapsular el código dependiendo de la plataforma, permitiendo el acceso a las APIs nativas del dispositivo. Además, desde la versión 1.9 ha permitido la mezcla de código nativo e híbrido.

Como ya se ha mencionado antes, las aplicaciones desarrolladas con Cordova, se basan en navegadores nativos o Webview, por lo que será necesario implementar estos Webview nativos en los que se va a incrustar la aplicación. Para esto, En web de Apache Cordova hay tutoriales que explican paso a paso la implementación.

Aunque hay disponibles bastantes plugins desarrollados por el equipo de Apache Cordova, que permiten el acceso a la mayoría de las funcionalidades de los dispositivos, también existe la opción de desarrollar nuevos plugins para acceder a estas funciones, implementándolos en los lenguajes nativos para los que se vayan a utilizar y publicarlos, o usar plugins existentes desarrollados por miembros de la comunidad open-source.

5. Ionic

Ionic Framework es una herramienta open-soruce para el desarrollo de aplicaciones híbridas, publicado en 2013, y que ha ido evolucionando en gran medida durante estos años. Las aplicaciones son desarrolladas en Angular y ejecutadas sobre Cordova lo que permite acceder a una gran cantidad de funcionalidades nativas gracias al uso de los plugins proporcionados por Cordova o, si es posible, al uso de los estándares web. Además, usando los mismos elementos HTML, es posible mostrar diseños nativos en función de la plataforma en la que se esté ejecutando.

Como se ha mencionado anteriormente, Ionic ha sufrido grandes cambios desde su publicación en 2013, por lo que se tratarán las características de cada una de las grandes versiones que más cambios han supuesto en el framework.

Ionic 1

Se trata de la primera versión de Ionic, publicada en 2013 y desarrollada para usar AngularJS sobre Cordova para el desarrollo de las aplicaciones. En esta versión, gracias al uso de Cordova, ya era posible incrustar la aplicación en un Webview nativo y acceder a distintas funcionalidades nativas a través del uso de plugins. En su momento fue una revolución ya que proporcionada una forma muy fácil y rápida de desarrollar aplicaciones híbridas.

Ionic 2

En 2016, se actualiza el framework AngularJS, siendo un cambio tan grande que se considera como la creación de un nuevo framwork, Angular. Debido a esto, el equipo de Ionic se ve a obligado a actualizar su framework para utilizar Angular. Gracias al uso de este nuevo framework se introducen en Ionic el uso de Typescript y la modularización del código, lo que permite separar el framework en distintas partes: core, angular, native…

Ionic 3

A partir de aquí, el equipo de Ionic decide un ciclo de releases de 6 meses, lo que permite tener un framework en constante evolución con dos "*breaking changes*" al año. Esta versión usará Angular 4 y se decide que, a partir de este momento, solo se usará la versión LTS del Angular. Debido a que esta actualización no supone un gran cambio, esta versión se centra principalmente en mejorar su framework y la integración con Angular. Entre estas mejoras se pueden destacar mejoras en el CLI de Ionic, que permite configurar, desarrollar y compilar proyectos de forma sencilla abstrayendo al desarrollador de los comandos internos, Ionic Native, que proporcionan clases propias de Ionic que recubren los plugins de Cordova, para facilitar su uso, la definición de anotaciones propias y la integración con el Routing de Angular.

Ionic 4

Esta última versión supone un cambio drástico en el framework, especialmente por el enfoque en la abstracción del framework utilizado. Para ello, se adopta el estándar WebComponents, fragmentos de interfaz de usuario creados usando tecnología web abierta y reusables. Gracias a esto, es posible desarrollar una aplicación con Ionic usando tanto Angular, React, Vue o cualquier otro framework o librería web, incluyendo código JavaScript plano con HTML. Para no abandonar a todos los usuarios actuales del framework se mantendrá en esta versión la librería Ionic-Angular que proporciona funcionalidades de Angular como Routing o Forms para su uso en el core de Ionic.

Además, en esta versión se introduce nuevas herramientas dentro del framework de Ionic: Stencil y Capacitor.

- Stencil es el generador de WebComponents usado por Ionic. Su propósito es el de generar componentes web estándar sin la necesidad de cargar ninguna librería ni framework extra. Además, también se usa esta herramienta para generar los componentes que conforman el core de Ionic. Entre las características de Stencil destaca el uso de un DOM virtual, renderizado asíncrono, enlace de datos reactivo y el uso de Typescript y JSX.

- Capacitor es un nuevo recubrimiento nativo y especializado creado por el equipo de Ionic para las plataformas actuales. Las principales diferencias que tiene con Cordova son la desaparición del concepto de plataformas y la introducción de una API estándar. Al contrario que Cordova que gestiona el build y la ejecución de cada plataforma automáticamente, Capacitor ofrece un proyecto nativo que será el wrapper para cada plataforma, pero la configuración, compilación y ejecución se realizan como si fuera una aplicación nativa. Por último, incluye una API para cubrir las funcionalidades más comunes para reducir el uso de plugins. Capacitor soporta las plataformas de Android y iOS con el posible soporte en el futuro del uso de elementos de interfaz de usuario nativos como usa React Native, PWA’s y escritorio.

Además de todas estas funcionalidades, el framework de Ionic incluye otras herramientas para facilitar el desarrollo de aplicaciones híbridas a los desarrolladores. Ionic Native es un conjunto de clases que recubren los plugins de Cordova para hacer más sencillo el acceso a funcionalides nativas. Ionic Studio es un IDE que intenta acercar el desarrollo de aplicaciones híbridas al desarrollo nativo, mostrando una interfaz de usuario más intuitiva y permitiendo arrastrar componentes a nuestra aplicación, añadiendo automáticamente la lógica y permitiendo cambiar sus propiedades, y dando la posibilidad de ver cómo va quedando la interfaz de la aplicación sin ejecutarla. Ionic Appflow es la implementación propia del equipo de Ionic de una herramienta para utilizar la metodología de DevOps en aplicaciones móviles. Proporciona un entorno de integración y despliegue continuos, permitiendo una integración sencilla con repositorios Git, y en futuro será posible automatizar la distribución a los Marketplace de Google y Apple.

Por último, Ionic facilita también la integración de herramientas de terceros útiles para el desarrollo de aplicaciones para dispositivos móviles. Por ejemplo, permite integrar fácilmente tu aplicación con SQLite o Firebase para proporcionar funcionalidades de bases de datos, con Microsoft Active Directory para la autenticación, con Apple Pay o Paypal para facilitar la gestión de pagos, con Facebook o Google Plus para permitir al usuario autenticarse mediante redes sociales o con AdMob para añadir publicidad en las aplicaciones.

6. Angular

Angular es un framework para el desarrollo de aplicaciones web, aplicaciones móviles o back-end junto con NodeJS, impulsado por Google. Angular surgió como la actualización del framework AngularJS, por lo que también es conocido como Angular 2, aunque ya ha tenido numerosas versiones, la última versión estable es la 7. En cuento al front-end, su enfoque es el de desarrollar aplicaciones SPA usando MVC para facilitar el desarrollo y las pruebas de las aplicaciones web.

Entre sus principales características destacan las siguientes:

- Multiplataforma: Permite desarrollar aplicaciones web, PWA, aplicaciones móviles híbridas usando Ionic o NativeScript, o aplicaciones para escritorio, tanto para Windows, Mac o Linux.

- Rendimiento: Permite transformar plantillas en código optimizado para máquinas virtuales de JavaScript, carga la primera vista de las aplicaciones usando NodeJS, .NET o PHP para un renderizado casi instantáneo y carga de forma más rápida las páginas gracias al router de componentes, que permite la separación del código de forma automática para que solo sea necesario cargar el código que se require para renderizar la vista actual.

- Productividad: Permite crear interfaces de usuario con una sintaxis simple y potente de plantillar, facilita la construcción y el despliegue de las aplicaciones gracias al uso de Angular CLI y ofrece un autocompletado de código inteligente, errores en tiempo real y otros tipos de feedbacks a través de sus IDEs.

- Desarrollo: Mediante el uso de Karma facilita la construcción de test unitarios, mientras que con Protractor es posible ejecutar los escenarios de test de una manera más rápida y estable, permite implementar animaciones y coreografías complejas y con un alto rendimiento con poco código y de forma intuitiva, y posibilita el desarrollo de aplicaciones accesibles mediante el uso de componentes habilitados para el uso de ARIA.

Arquitectura

Los bloques básicos de una aplicación desarrollada en Angular son los *NgModules* que proporcionan un contexto para los componentes en el momento de la compilación y agrupan el código relacionado en conjuntos funcionales. Una aplicación en Angular está formada por un conjunto de *NgModules* y siempre tendrá que tener al menos un módulo raíz que permite cargar toda la aplicación.

Los componentes definen vistas, que son conjuntos de elementos que pueden ser modificados en función de la lógica y los datos de la aplicación, y usan servicios, inyectados en los componentes en forma de dependencias, haciendo el código más modular y reusable, que proporcionan funcionalidades no directamente relacionadas con las vistas. Tanto los componentes como los servicios son clases con decoradores que indican a Angular como tiene que usarlos. Los metadatos de la clase de un componente lo asocian a una plantilla que tendrá elementos HTML con directivas de Angular que modifican el código HTML antes de renderizarlo, y los metadatos de la clase de un servicio contienen la información necesaria para ser usada a través de la inyección de dependencias.

Directivas

Las directivas modifican las vistas antes de ser renderizadas proporcionando lógica a ellas. Las directivas pueden ser de dos tipos, directivas de atributo o estructurales. Las directivas de atributo, como su nombre indica, son usadas como atributos de los elementos HTML y modifican la apariencia o el comportamiento de un elemento, componente o directiva.

```
<div [style.font-size]="isSpecial ? 'x-large' : 'smaller'" >
  This div is x-large or smaller.
</div>
```

1. *Ejemplo de directiva de atributo*

Las directivas estructurales modifican la estructura del DOM, normalmente añadiendo, quitando o manipulando elementos, como por ejemplo **ngIf*, que oculta el elemento a no ser que se cumpla la condición.

```
<div *ngIf="hero" class="name">{{hero.name}}</div>
```

2. *Ejemplo de directiva estructural*

Además de las directivas ya disponibles, es posible crear nuevas directivas usando la anotación @Directive.

Inyección de dependencias

Dependencia es cualquier objeto o servicio que una clase necesita para realizar su función. Inyección de dependencias es un patrón de desarrollo en el que una clase pide dependencias de una fuente externa en lugar de crearlas ella misma. Angular tiene su propio framework de inyección de dependencias que permite aumentar la eficiencia y la modularidad de las aplicaciones.

Para poder inyectar un servicio deberá ser marcado con la anotación *@Intectable*, y posteriormente, usar ese servicio inicializándolo en el constructor de la clase que va a usarlo.

```
@Injectable({
 providedIn: 'root',
})
```

3. Uso de la anotación @Intectable

Por defecto, cuando se crea un servicio con Angular CLI, se registra un proveedor con el inyector de la raíz para el servicio.

Data binding

Angular soporta enlace de datos bidireccionales, lo que significa que cualquier cambio que se produzca en el DOM debido a una acción del usuario, se verá reflejado en los datos de la aplicación. Este enlace de datos se consigue mediante el uso de directivas, sin necesidad de escribir código. Existen distintas formas de efectuar este enlace de datos:

- {{variable}}, muestra en el DOM el valor de la variable.
- [propiedad] = “valor”, manda el valor a la propiedad especificada.
- (evento) = “función”, llama a la función especificada cuando ocurre un evento.

- [(ngModel)] = “property” en un input, implementa un enlace bidireccional, en el que el valor de la propiedad se inyecta en el elemento, y si se produce un cambio en elemento, este fluye hasta el componente modificando el valor de la propiedad.

```
<li>{{hero.name}}</li>
<app-hero-detail [hero]="selectedHero"></app-hero-detail>
<li (click)="selectHero(hero)"></li>
```

4. *Ejemplos de enlace de datos*

```
<input [(ngModel)]="hero.name">
```

5. *Enlace de datos bidireccional*

Routing

El *Router* de Angular permite navegar entre vistas cuando el usuario realiza ciertas acciones. El Router de Angular sigue el modelo de un navegador web, acepta una dirección URL y navega a la vista que corresponde con esa URL, pudiendo pasar parámetros opcionales que pueden ayudar a decidir qué vista mostrar.

Las aplicaciones en Angular tienen un elemento raíz que se debe especificar de la siguiente forma en el fichero index.html: <base href="/">. Para configurar el routing, lo primero que será necesario es asociar las distintas rutas que va a tener la aplicación con las distintas vistas. Esta configuración se usará a la hora de crear los *NgModules*.

```
const appRoutes: Routes = [
  { path: 'crisis-center', component: CrisisListComponent },
  { path: 'hero/:id',      component: HeroDetailComponent },
  {
    path: 'heroes',
    component: HeroListComponent,
    data: { title: 'Heroes List' }
  },
  { path: '',
    redirectTo: '/heroes',
    pathMatch: 'full'
  },
  { path: '**', component: PageNotFoundComponent }
];
```

6. *Configuración de rutas*

Como se ha mencionado anteriormente, cada ruta corresponde con una vista, que son los componentes. En la segunda ruta, *:id* es un parámetro con el que se obtendrán datos para cargar posteriormente la vista. La propiedad *data* en la tercera ruta es un lugar donde almacenar ciertos datos útiles para la vista como su título o datos estáticos. La ruta vacía representa la ruta por defecto, donde se irá cuando la URL esté vacía y normalmente cuando se inicia la aplicación. Si se usa la propiedad *redirectTo*, será redirigido a la ruta establecida. Por último, la ruta ** es un comodín, se elegirá esta ruta cuando la URL no corresponde con ninguna de las configuradas.

7. Typescript

TypeScript es un lenguaje de programación open-source desarrollado y mantenido por Microsoft. TypeScript es un superconjunto de JavaScript, por lo que además de añadir tipado estático y objetos basados en clases e interfaces, también admite JavaScript plano. Además, el código TypeScript desarrollado, será transformado en código JavaScript a la hora de compilarlo. La posibilidad de definir los tipos en tiempo de diseño ayuda a evitar errores de ejecución, como ocurriría por ejemplo al pasar un tipo incorrecto a una función. Los distintos tipos admitidos son los siguientes:

- String: cadena de caracteres.
- Number: entero.
- Boolean: dato lógico que representa verdadero o falso.
- Array: dato estructurado que permite almacenar una colección de elementos.
- Enum: tipo de dato que representa una enumeración.
- Any: representa una variable que puede ser de cualquier tipo.
- Void: indica que una función no devolverá ningún valor.

Además, existen distintas formas de declarar variables, *var*, *let* y *const.* La primera de las formas supone algunos problemas debido a que tiene un ámbito global dentro la función o clase donde está declarada. Por esto se recomienda usar las otras dos formas. Cuando se declara una variable usando *let*, tendrá un ámbito a nivel de bloque, por lo que, si se declara dentro de un bloque *if*, solo será accesible desde ese mismo bloque, y no es posible declarar la variable dos veces dentro de un mismo ámbito, al contrario que cuando se usa *var*. Si se declara una variable usando *const*, su ámbito será el mismo que *let*, pero como su nombre indica, su valor no podrá ser modificado, pero si el valor de sus propiedades.

Por último, para establecer el tipo de una variable, se indica el tipo precedido de dos puntos al final de la declaración de la variable o se infiere por el valor asignado.

8. Node.js

Node.js es un entorno de tiempo de ejecución multiplataforma, de código abierto, para la capa de servidor, pero sin limitarse a ello, basado en JavaScript, en concreto ECMAScript. Node.js es un entorno asíncrono, con una metodología de I/O de datos fundamentado en una arquitectura orientada a eventos y usa el motor V8 de Google, gracias al cual es posible compilar código JavaScript del lado del servidor a altas velocidades. Estas velocidades son importantes porque el motor V8 compila el código en código máquina nativo, en lugar de ejecutarlo como bytecode.

Node.js fue creado con el objetivo principal de ser un entorno de ejecución con una gran capacidad de escalabilidad. Esto ocurre debido a que, aunque permita varias conexiones simultáneas, al estar basado en eventos, si no tiene que realizar ningún trabajo, Node estará “durmiendo”. Esto contrasta con el modelo de concurrencia en el que se usan hilos del sistema operativo. Las operaciones que se realizan basadas en hilos son ineficientes y difíciles de usar, además de que existe la probabilidad de se produzca un bloqueo en el hilo. En cambio, en Node no es posible que se produzca un bloqueo, ya que casi nunca realiza operaciones I/O bloqueantes directamente. Este diseño permite que si se realizan

numerosas conexiones desde múltiples clientes no se perjudique la velocidad y rendimiento de estos clientes.

El diseño de Node está altamente inspirado por sistemas como Event Machine de Ruby o Twisted de Python, se crea un bucle de eventos asíncrono como entorno, pero en cambio, en Node no se hace ninguna llamada bloqueante al inicio del script. Node ingresa el bucle de eventos después de ejecutar el script de inicio, y se sale de él cuando ya no queda ningún *callback* que ejecutar.

A pesar de que Node está construido para usarlo sin hilos, también es posible aprovechar los múltiples hilos del hardware creando procesos hijos mediante la API *child_process.fork()*, la cual permite una gran facilidad para comunicarse con el hilo principal.

A pesar de esto, también existen algunas desventajas en el uso de Node.js. Es posible que un entorno desarrollado con Node.js no funcione dependiendo de la empresa de hosting que proporcione el alojamiento del servidor, cuenta con una API inestable y que acostumbra a realizar cambios que rompen la compatibilidad hacia atrás entre versión y versión, por lo que será necesario ir realizando cambios frecuentes en el código para que siga funcionando en las versiones actuales, ausencia de una librería estándar y ausencia de librerías en general, debido a que JavaScript no ha sido popular en el lado del servidor, todas las librerías son recientes y poco maduras.

Algunos ejemplos del uso de Node.js por empresas grandes y populares serían Paypal, Netflix o Linkedin, que necesitan contar con sistemas altamente escalables, debido a que cuentan con millones de usuarios. Gracias a la implantación de Node.js afirman haber mejorado en gran medida los tiempos de carga de sus aplicaciones, haber reducido la cantidad de código y el tiempo necesario para el desarrollo, haber reducido notablemente la utilización de recursos y haber mejorado la experiencia de usuario.

9. IBM Cloud

IBM Cloud, llamado anteriormente IBM Bluemix, es un entorno de plataforma como servicio desarrollado por IBM. Soporta varios lenguajes de programación como Java, Node.js, PHP o Python, así como la posibilidad de implantar la metodología DevOps en las aplicaciones en la nube, para poder construirlas y desplegarlas automáticamente, y numerosos servicios. Está basado en la tecnología abierta Cloud Foundry, una plataforma como servicio desarrollada por VMWare, corriendo sobre la infraestructura SoftLayer.

IBM Cloud ofrece distintos servicios en la nube de forma pública, privada o híbrida en los que se incluyen servicios para el análisis de datos, inteligencia artificial, IoT o blockchain. Entre estos servicios se pueden destacar los siguientes:

- Para la computación se ofrecen servidores totalmente personalizables, Cloud Foundry para desplegar y automatizar el escalado de ellas, registros de contenedores de Docker, servicios de IBM Cloud Kubernetes o servidores para almacenamiento masivo de datos.

- Para la seguridad proporciona encriptación de datos entre aplicaciones de cliente y servidor, facilita la autenticación en aplicaciones web y móviles incluyendo autenticación a través de redes sociales, permite almacenar y centralizar distintos certificados en un repositorio seguro o proteger servidores y clientes con distintos firewalls.

- En el ámbito de bases de datos proporciona bases de datos en la nube tanto SQL como NoSQL con diversos gestores de base de datos como MongoDB, MySQL, PostgreSQL o Redis, servicios de data warehouse con DB2 o la posibilidad de poder migrar datos de forma rápida, fácil y segura entre base de datos.

- Para el análisis de datos ofrece servicios de data warehouse y análisis de datos usando DB2 con Apache Spark y Apache Hadoop o permite analizar texto, audio, video o datos geoespaciales.

- En cuando a la inteligencia artificial se cuenta con todos los servicios de IBM Watson además de servicios para desarrollar y desplegar modelos de deep learning o herramientas para preparar grandes cantidades de datos para su posterior empleo.

- Para el desarrollo de IoT, ofrece servicios que permiten conectar dispositivos y redes para su análisis o la integración de datos del tiempo en tus aplicaciones.

- Para el desarrollo de aplicaciones móviles proporciona servicios para construir aplicaciones móviles escalables, seguras con servicios de back-end en la nube, servicios para enviar notificaciones push y para monitorizar el rendimiento y el uso de las aplicaciones.

10. IBM Watson

IBM Watson es un conjunto de herramientas proporcionadas por IBM en su plataforma como servicio IBM Cloud para el desarrollo de aplicaciones relacionadas con el campo de la inteligencia artificial, incluyendo funcionalidades como el procesado de lenguaje natural, el reconocimiento de imágenes o machine learning. Las herramientas incluidas son las siguientes:

- IBM Watson Studio es una herramienta construida para impulsar la actividad de empresas dedicadas al ámbito de *data science* y *machine learning.* Esta herramienta ayuda a simplificar y aumentar la velocidad de los procesos de experimentación, exploración, desarrollo y entrenamiento de modelos, y escalado de las operaciones de *data science*. Esto es posible gracias al análisis de distintas fuentes de datos y la inclusión de predicciones en los procesos de negocio optimizando el valor de negocio, mostrando datos relevantes de forma visual y mejorando la toma de decisiones.

- IBM Watson Knowledge Studio es una herramienta visual que permite visualizar los documentos y entrenar los distintos modelos de aprendizaje de manera intuitiva, sin necesidad de escribir código y de forma colaborativa.

- IBM Watson Data Refinery permite combinando su uso con el uso de IBM Watson Studio e IBM Watson Knowledge Studio ahorrar grandes cantidades de tiempo transformando datos sin procesar en datos listos para ser consumidos y con gran calidad. Esto se puede hacer desde una interfaz visual e intuitiva sin necesidad de escribir ninguna línea de código, mientras se muestran distintas gráficas sobre los datos.

- IBM Watson Natural Language Understanding permite analizar textos en hasta trece idiomas diferentes de forma automática y extraer distintos tipos de datos de ellos, como conceptos, palabras clave, entidades, emociones o relaciones. También es posible personalizar los modelos para obtener información específica de tu organización.

- IBM Watson Natural Language Classifier permite crear de forma rápida y sencilla modelos de clasificación de texto personalizables. Simplemente es necesario subir ficheros con datos para el entrenamiento y automáticamente se asignarán las entidades que mejor coinciden con los distintos fragmentos de texto para mejorar las búsquedas en los documentos. Además, también es posible modificar estas clasificaciones mediante el uso de una interfaz visual.

- IBM Watson Discovery es una herramienta que facilita la construcción de soluciones basadas en inteligencia artificial que tratan de buscar respuestas o información relevantes en un conjunto de documentos de forma rápida y automática. Esto se consigue mediante el entrenamiento de la inteligencia para entender los distintos elementos de los documentos, extrayendo contenido con valor, y mediante el propio procesador de lenguaje natural de IBM que permite extraer sentimientos, entidades, conceptos…

- IBM Watson Text to Speech permite generar audio similar al habla humana desde texto escrito en distintos idiomas, como inglés, francés, alemán o castellano, además de detectar algunos dialectos, como el inglés británico y el americano. Gracias a esto es posible mejorar la experiencia y la accesibilidad de los usuarios, o para evitar distracciones en sistemas que se usan mientras se conduce.

- IBM Watson Speech to Text proporciona una API para convertir audio a texto de forma automática y rápida desde hasta 7 idiomas distintos. Esto se consigue combinando información sobre la estructura del idioma con la composición de la seña de audio enviada. También es posible personalizar el modelo para mejorar la precisión o incluir distintos conceptos específicos para el negocio que interesa reconocer.

- IBM Watson Language Translator ofrece una API para traducir textos de un idioma a otro, permitiendo personalizar las traducciones con terminologías específicas de tu área o negocio.

- IBM Watson Personality Insights permite analizar diferentes características de la personalidad a través de distintos textos como emails, entradas de blog o tweets de forma que sea posible una predicción de las necesidades o los gustos para ofrecer una experiencia personalizada para cada cliente recomendando distintos tipos de música, películas o productos.

- IBM Watson Knowledge Catalog es un catálogo de datos que puede ayudar a encontrar rápidamente y con precisión grandes cantidades de datos o modelos analíticos, categorizarlos y compartirlos. Es una herramienta muy útil para analistas o ingenieros de datos, ya que permite preparar los datos automáticamente para entenderlos mejor antes de trabajar con ellos.

- IBM Watson IoT Platform permite el uso de las APIs de IBM Watson y tableros visuales para mejorar y facilitar el uso de las funcionalidades de IoT que proporciona IBM. De esta forma es posible realizar predicciones en tiempo real a partir de ciertos datos usando aprendizaje automático y APIs cognitivas, integrar y comunicar entre ellos distintos dispotivos IoT y monitorizar estos dispositivos para segurar su correcto funcionamiento, rendimiento y seguridad.

- IBM Watson Visual Recognition es un servicio que permite analizar imágenes reconociendo distintos tipos de objetos o caras de personas mediante el uso de algoritmos de aprendizaje profundo. Aunque ya existe un modelo de reconocimiento de caras humanas creado por el equipo de IBM, es posible crear y entrenar nuevos modelos para reconocer objetos específicos interesantes para las organizaciones.

- IBM Watson OpenScale ofrece un tablero en el que poder monitorizar todas las operaciones de inteligencia artificial que se vayan desarrollando a lo largo del ciclo de vida de un proyecto, mostrando cierta información relevante que permite cambiar algunos parámetros para adaptarse a distintas situaciones.

- IBM Watson Tone Analyzer es un servicio que detecta emociones y tonos de lenguaje en texto escrito mediante el análisis de éste. De esta forma es posible analizar distintas publicaciones en redes sociales para predecir el estado de ánimo de una persona, monitorizar el servicio de atención al cliente para poder mejorar su satisfacción o integrarlo con un asistente virtual para ajustar la conversación en función del estado de ánimo del usuario.

- IBM Watson Machine Learning ayuda a acelerar el proceso de poner en desarrollo e integrar inteligencia artificial en las aplicaciones de las empresas, permitiendo aprovechar los beneficios del aprendizaje automático y el aprendizaje profundo de una forma sencilla y obteniendo valor de negocio. Esto se puede hacer de forma sencilla e intuitiva gracias a la integración con Watson Studio y la generación automática de APIs para usar la inteligencia artificial desde las aplicaciones.

- IBM Watson Assistant, antes llamado Watson Conversation, permite integrar asistentes virtuales en cualquier aplicación. Al contrario que otras herramientas que tratan de imitar el comportamiento humano, Watson Assistant busca la respuesta correcta de entre una base de conocimiento cuando es necesario o preguntar para que se vuelva a formular una pregunta si no la ha entendido. Gracias a una interfaz visual e intuitiva es posible crear conversaciones de forma rápida y sencilla sin necesidad de código. Además, mientras se va desarrollando una conversación, Watson Assistant va almacenando un historial del chat y soporta hasta 13 idiomas diferentes.

Desarrollo

1. Introducción

En esta segunda parte se detallarán algunos de los aspectos más importantes de la aplicación y de los pasos realizados para llevar a cabo el desarrollo. Como ya se ha mencionado anteriormente, el principal objetivo de este proyecto es el de proporcionar un asistente virtual, con interacción por voz y escrita, con el que los clientes de una empresa puedan realizar ciertas operaciones de forma más cómoda que con una aplicación convencional para ofrecer más posibilidades tanto de usabilidad como de accesibilidad, como podría ser compras online o atención al cliente, mientras mantienen una conversación.

El proyecto consta de una aplicación móvil desarrollada usando Ionic 4 sobre Angular y Apache Cordova que servirá como interfaz para el usuario, una parte de back-end desarrollada en Node.js que funcionará como un *middleware* entre la aplicación los servicios de IBM Watson y estará alojado en la plataforma IBM Cloud, y por último los distintos servicios de IBM Watson que se han usado como Watson Assistant, Watson Discovery o Watson Speech to Text entre otros.

Para desarrollar el proyecto se ha utilizado las siguientes tecnologías:

- Ionic Framework, versión 4.1.2.
- Ionic CLI, versión
- Node, versión
- IBM Cloud
- Bluemix CLI
- Visual Studio Code, IDE ligero para el desarrollo de aplicaciones que cuenta con numerosos plugins para multitud de lenguajes.
- Android Studio
- Xcode
- Sourcetree, herramienta visual para manejar el control de versiones que permite interactuar con repositorios Git de forma sencilla y visual.

2. Especificación de requisitos

Los requisitos funcionales generales del proyecto se han definido en forma de historias de usuario para facilitar su lectura y entenderlos de manera más fácil y rápida.

- **US01:** Como usuario quiero poder conversar con el asistente virtual para hacer diversas operaciones.

- **US02:** Como usuario, quiero ser respondido por el asistente virtual siempre que le diga algo.

- **US03:** Como usuario, quiero ser informado si el asistente virtual no ha entendido correctamente lo último que le he dicho.

- **US04:** Como usuario quiero poder conversar con el asistente virtual mediante texto o escrito o por voz.

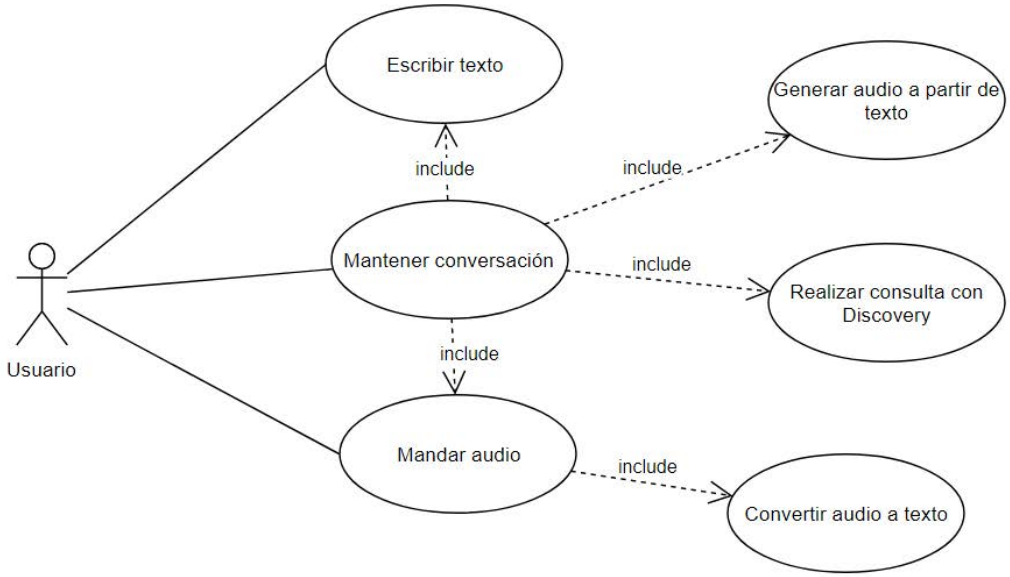
- **US05:** Como usuario quiero poder ver y escuchar las respuestas que me da el asistente virtual.

- **US06:** Como usuario quiero poder repetir el audio de una respuesta del asistente virtual para volver a escucharla.

Como requisitos no funcionales del proyecto se encuentra que la interfaz visual debe tener el aspecto de un chat en el que el usuario puede conversar con el asistente virtual, y que el asistente virtual debe tardar unos instantes, no muy largos, en contestar para dar la sensación de estar hablando con una persona.

3. Arquitectura

3.1. Diagrama de casos de uso



7. Diagrama de casos de uso

3.2. Casos de uso extendidos

Nombre	**Mantener conversación**
Actores	Usuario
Tipo	Primario
Precondiciones	Ninguna
Curso típico de eventos	
Usuario	Sistema
1. Casos de uso “Escribir texto” o “Mandar audio”	
	2. Procesar el mensaje recibido en Watson Assistant
	3. Caso de uso “Generar audio a partir de texto”
	4. Mostrar la respuesta para el usuario en el chat

Curso alternativo al paso 4	
	3. No se encuentra respuesta, se pregunta al usuario si quiere que se busque información
4. Aceptar la búsqueda de información. Si se rechaza se termina el flujo	
	5. Caso de uso “Realizar consulta con Discovery”
	6. Mostrar el resultado de la búsqueda en el chat

Nombre	**Escribir texto**
Actores	Usuario
Tipo	Primario
Precondiciones	Ninguna
Curso típico de eventos	
Usuario	Sistema
1. Escribir el texto en el cuadro de texto que se encuentra en la parte inferior de la pantalla	
2. Pulsar el botón de enviar situado al lado del cuadro de texto	
	3. Recibir el mensaje enviado

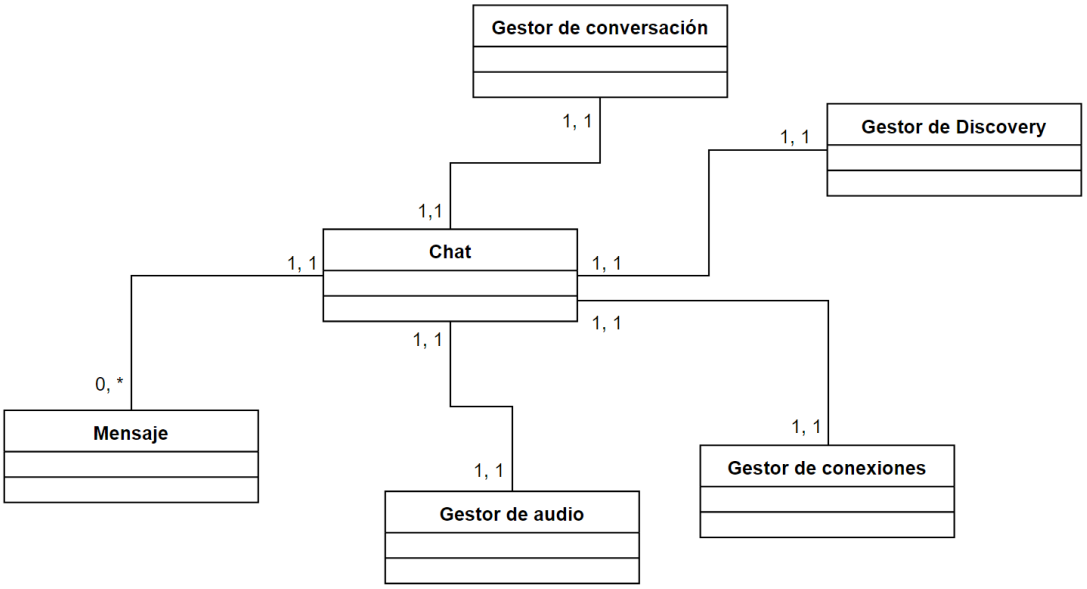
Nombre	**Mandar audio**
Actores	Usuario
Tipo	Primario
Precondiciones	Ninguna
Curso típico de eventos	
Usuario	Sistema
1. Mantener pulsado el botón de enviar situado en la esquina inferior derecha de la pantalla	
2. Dejar de pulsar el botón para enviar el audio	
	3. Recibir el audio enviado
	4. Caso de uso “Convertir audio a texto”

Nombre	**Generar audio a partir de texto**
Actores	Ninguno
Tipo	Primario
Precondiciones	Ninguna
Curso típico de eventos	
Sistema	
1. Recibir el texto que se va a responder el usuario	
2. Transformar el texto a audio mediante el uso de Watson Text to Speech	
3. Reproducir el audio	

Nombre	Convertir audio a texto
Actores	Ninguno
Tipo	Primario
Precondiciones	Ninguna
Curso típico de eventos	
Sistema	
1. Recibir el audio que ha enviado el usuario	
2. Convertir el audio a texto mediante el uso de Watson Speech to Text	
3. Mostrar el texto en el chat	

Nombre	Realizar consulta con Discovery
Actores	Ninguno
Tipo	Primario
Precondiciones	Ninguna
Curso típico de eventos	
Sistema	
1. Recibir el texto sobre el que se va a hacer la búsqueda	
2. Realizar la búsqueda mediante Watson Discovery en los documentos cargados	
3. Procesar el resultado y mostrarlo en el chat para el usuario	

3.3. Diagrama de clases de la aplicación

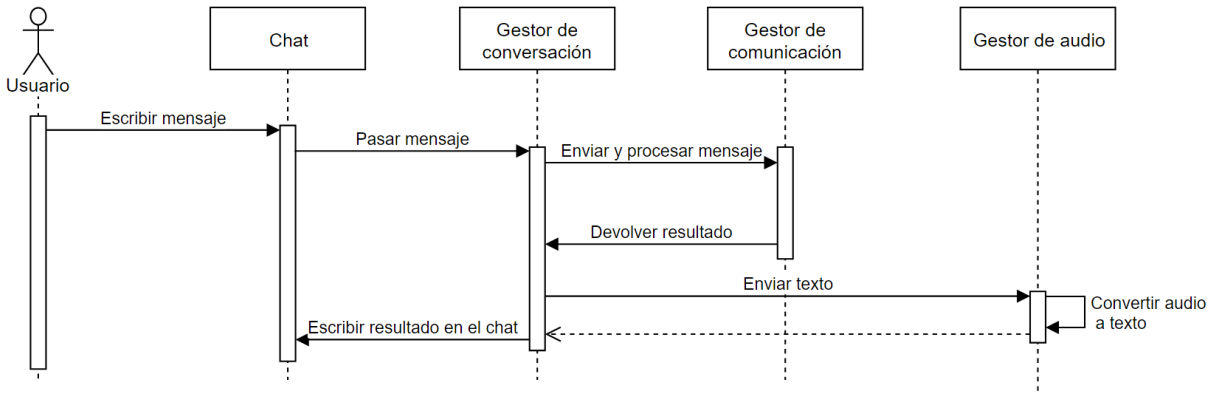


8. Diagrama de clases

En cuanto al diagrama de clases de la aplicación, se puede encontrar un componente principal, el chat, que será la única página que tendrá la aplicación. En esta página se llevará a cabo la conversación, que tendrá un conjunto de mensajes. Para que sea posible que se realice esta conversación la página del chat se conectará con distintos servicios o gestores, que proporcionan las funcionalidades que necesita como el manejo de la conversación, de las conexiones con el back-end, del audio y de las conexiones e integraciones con Watson Discovery.

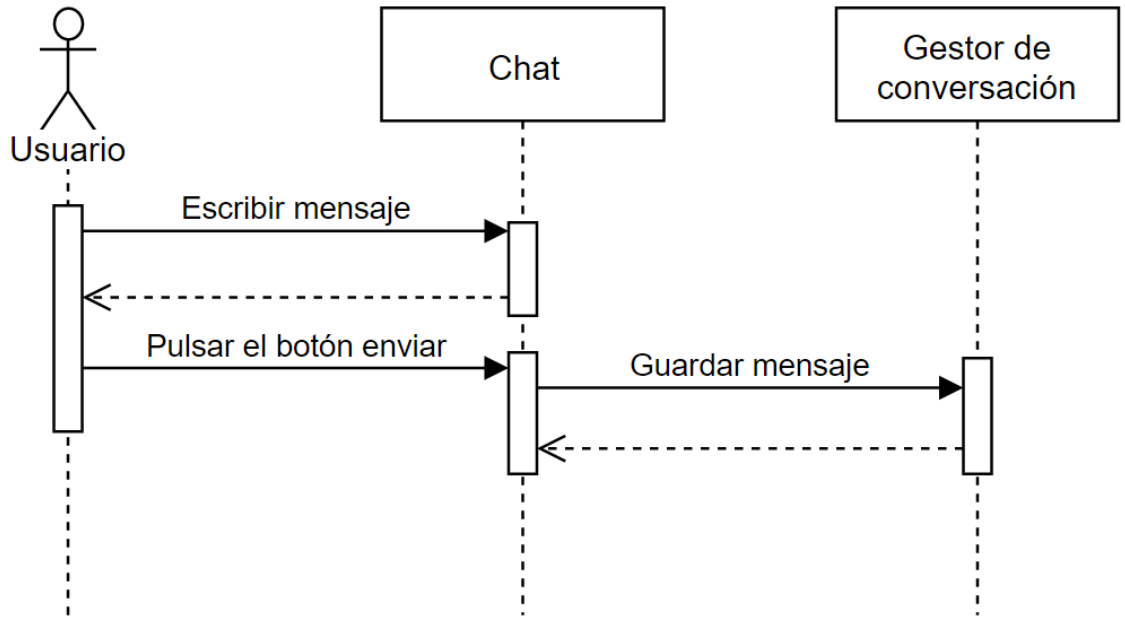
3.4.Diagramas de secuencia

Mantener conversación



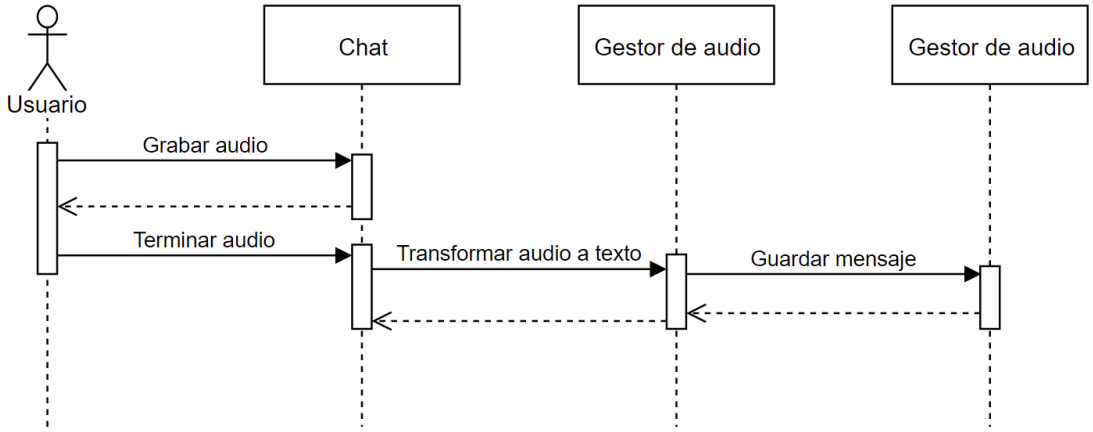
9. Diagrama de secuencia: caso de uso "Mantener conversación"

Escribir texto



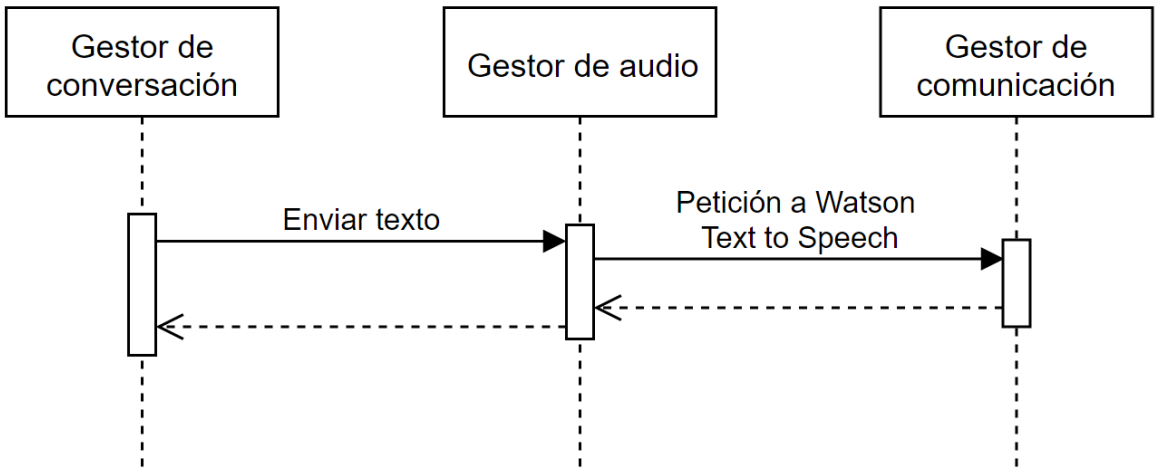
10. Diagrama de secuencia: caso de uso "Escribir texto"

Mandar audio



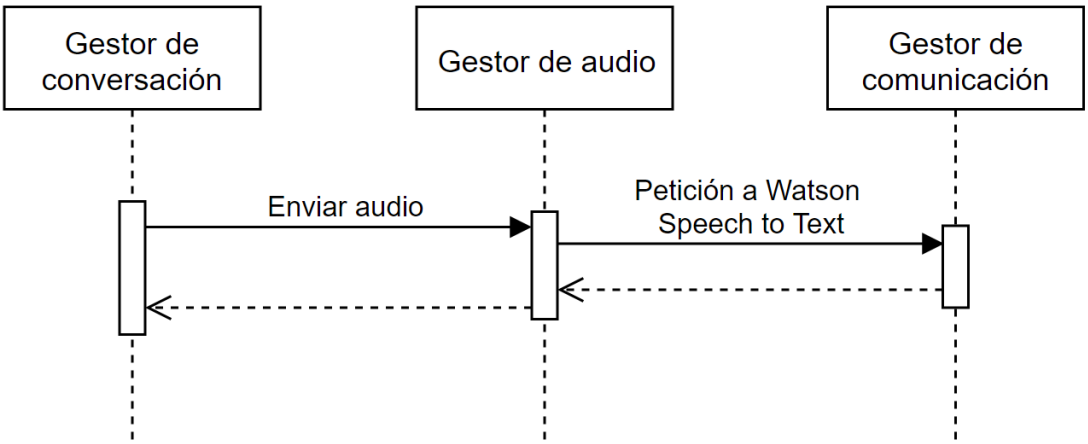
11. Diagrama de secuencia: caso de uso "Mandar audio"

Generar audio a partir de texto



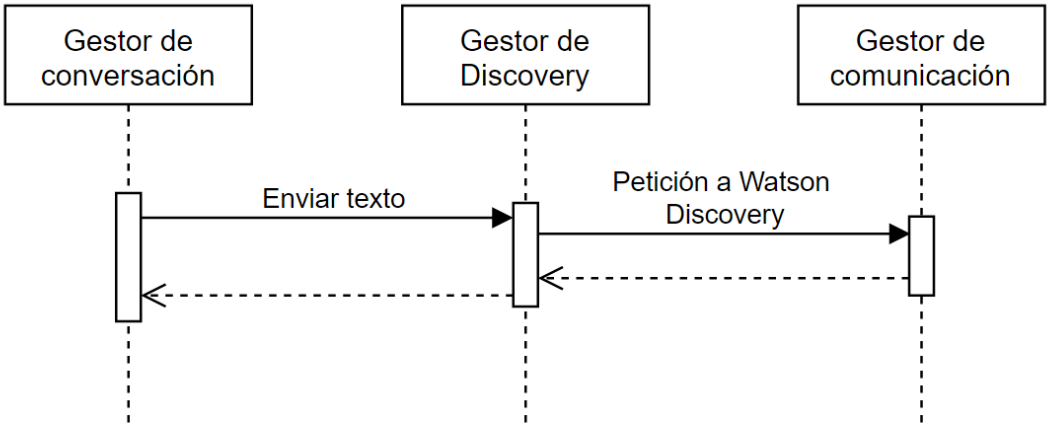
12. Diagrama de secuencia: caso de uso "Generar audio a partir de texto"

Convertir audio a texto



13. Diagrama de secuencia: caso de uso "Convertir audio a texto"

Realizar consulta con Discovery

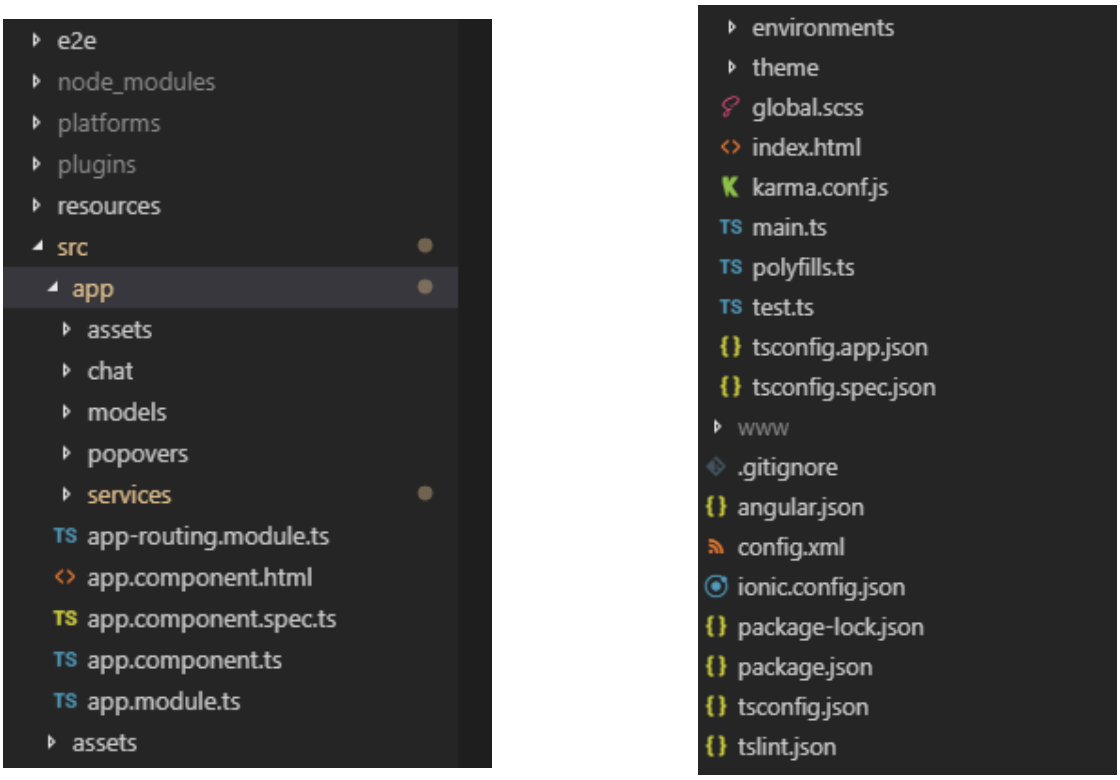


14. Diagrama de secuencia: caso de uso "Realizar consulta con Discovery"

4. Desarrollo del front-end

Para construir la parte del front-end del proyecto, se ha utilizado Ionic 4 sobre Apache Cordova y Angular para desarrollar una aplicación híbrida, que se podrá ejecutar tanto en un móvil con el sistema operativo Android como con iOS. Al estar desarrollada con Angular, la aplicación tendrá forma de SPA, que se basan en el uso de distintos componentes, que se irán creando según el usuario vaya realizando acciones dentro de un elemento HTML que funcionará como raíz de la aplicación.

Lo primero de todo, será crear el proyecto de Ionic. Usando Ionic CLI es posible crear un proyecto de forma muy rápida y sencilla. Solo es necesario ejecutar el comando 'ionic start *nombre de la aplicación'*. Este comando creará automáticamente los ficheros necesarios y la estructura básica del proyecto. También es posible crear un proyecto usando una de las plantillas proporcionadas por Ionic añadiendo el nombre de la plantilla al final del comando, *tabs*, para crear un proyecto con un diseño basado en pestañas, *sidemenu*, para tener un diseño con un menú lateral, y *blank*, para crear un proyecto con una única página en blanco.



15. Estructura del front-end

En las imágenes anteriores, se puede observar la estructura básica de un proyecto en Ionic 4. Entre estas carpetas se puede destacar la carpeta *node_modules*, donde se guardan las librerías descargadas, *plugins*, donde se almacenan los plugins de Apache Cordova o De Ionic Native que se utilizan en la aplicación, o la carpeta *resources*, donde se encuentran las imágenes que se van a utilizar como icono y splash de la aplicación. Por otro lado, también destacan los ficheros *global.scss*, donde se pueden declarar distintas variables globales referentes al estilo de la aplicación, *index.html*, donde se inicializa el webview y permite que arranque la aplicación, *config.xml*, donde se especifica información sobre la aplicación, como su nombre, identificador único, los iconos y splashes que se van a utilizar o los plugins y sus versiones, y *package.json*, donde se establecen los plugins y las librerías y sus versiones entre otras cosas.

Ya dentro de la carpeta *src/app* se encuentra todo el código que va a conformar la aplicación. Cada página de la aplicación tendrá una carpeta específica, que contará por lo menos con un fichero TypeScript, un fichero HTML y un fichero CSS. Después el resto de los ficheros se dividen según su función, por ejemplo, si son modelos, servicios o por el tipo de componente que sean. En la carpeta *assets*, se encuentran los distintos iconos e imágenes que se van a visualizar en la aplicación.

Para implementar la navegación por la aplicación en función de la interacción de usuario, se usa el *routing* de Angular, en el que primero se establecen distintas rutas dentro de la aplicación que corresponden o redirigen a una página.

```
const routes: Routes = [
  { path: '', redirectTo: 'chat', pathMatch: 'full' },
  { path: 'chat', component: ChatPage },
];

@NgModule({
  imports: [
    RouterModule.forRoot(routes, { preloadingStrategy: PreloadAllModules })
  ],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

16. Implementación de las rutas

Como esta aplicación solo cuenta con la página del chat, las rutas que se establecerán serán simplemente la ruta de la página y la ruta por defecto, que redirigirá a la página del chat cuando se inicia la aplicación. Cuando se ha creado el objeto con las rutas, éste se usa en el módulo de *routing* de Angular para crear la estructura.

```
<ion-app>
  <ion-router-outlet></ion-router-outlet>
</ion-app>
```

17. Elemento raíz de la aplicación

Luego, para que se inicie, es necesario escribir el elemento *"ion-router-outlet"* en el elemento raíz de la aplicación para permitir navegar por las distintas rutas.

Para generar las distintas páginas, el framework de Ionic permite, mediante Ionic CLI, crear una página de forma rápida y sencilla. Usando el comando 'ionic generate page *"nombre/ruta de la página"',* se crearán todos los ficheros necesarios para el uso de la página y se actualizará la configuración de las rutas para incluir esta página. Las páginas, como cualquier otro componente, están formadas por código TypeScript, para implementar la lógica, código HTML, para construir la interfaz de usuario y código SCSS para dar estilo a la interfaz.

Al igual que las páginas, también es posible crear mediante el uso del comando 'ionic generate' cualquier tipo de componente o servicios. Como ya se ha mencionado anteriormente, los componentes son fragmentos de código que están marcados con la anotación @Component y se pueden utilizar en las páginas, para de esta forma fomentar la reutilización de código y aumentar la modularidad.

```
@Component({
    selector: 'app-options-popover',
    templateUrl: './options-popover.component.html',
    styleUrls: ['./options-popover.component.scss'],
})
export class OptionsPopoverComponent implements OnInit {
```

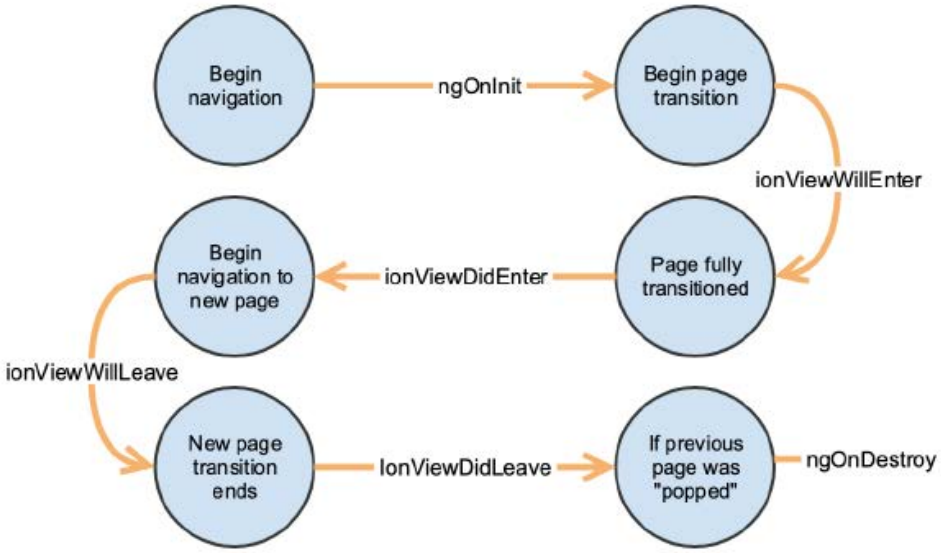
18. Creación de un componente

Por otro lado, los servicios son clases marcadas con la anotación @Inyectable y que pueden ser inyectadas en cualquier componente o en otros servicios para aportar funcionalidades transversales y que podrían ser usadas desde distintos componentes de la aplicación.

```
@Injectable({
    providedIn: 'root'
})
export class ConversationService {
```

19. Creación de un servicio

Como todos los elementos visuales que se muestran en una aplicación se encuentran dentro de componentes, que a su vez estarán dentro de la raíz de la aplicación u otros componentes, será necesario controlar el ciclo de vida de estos componentes. Para esto, existen distintos eventos tanto de Angular como de Ionic para realizar acciones cuando se va a entrar en una nueva vista, cuando se ha salido ya de una vista o cuando se destruye un componente. En la imagen siguiente se pueden ver los eventos disponibles.



20. Ciclo de vida de los componentes en Ionic

Cuando se navega a una nueva página, Ionic mantiene la página antigua en el DOM, aunque ya no se vea. De esta forma es posible mantener los datos que tenía la página y hacer la transición de vuelta más eficiente debido a que no se tiene que crear de nuevo. Las páginas solo se eliminan del DOM cuando se sacan de la pila de vistas, por ejemplo, pulsando el botón de atrás. Debido a esto, es posible que los eventos *ngOnInit* y *ngOnDestroy* no se disparen cuando se piensa

Debido a que Ionic es un framework híbrido, es decir, el código va a ser el mismo para todas las plataformas en las que se ejecute, será necesario tener algunas consideraciones a la hora de optimizar las aplicaciones para las distintas plataformas y sistemas operativos. Todas las funcionalidades nativas, como la cámara o la geolocalización, necesitan hacer llamadas a las APIs del sistema operativo para acceder al hardware. Debido a esto, si se está ejecutando la aplicación en un navegador web, no se podrán ejecutar estas funcionalidades. A pesar de esto, los plugins de *Ionic Native* incluyen lógica para detectar cuando la aplicación se está ejecutando en un entorno nativo, de forma que no se detendrá la ejecución de la aplicación si se llama a alguna funcionalidad nativa y no tiene acceso a ningún hardware nativo, simplemente se mostrará un mensaje de aviso.

Adicionalmente, desde el código TypeScript es posible detectar la plataforma en la que se está ejecutando la aplicación. El uso de esta funcionalidad es muy recomendado antes de hacer una llamada a las APIs nativas y es muy útil si se va a acceder a una funcionalidad específica de ciertas plataformas.

```
if (this.platform.is('android')) {
    this.androidPermissions.requestPermissions([
        this.androidPermissions.PERMISSION.READ_EXTERNAL_STORAGE,
        this.androidPermissions.PERMISSION.WRITE_EXTERNAL_STORAGE,
        this.androidPermissions.PERMISSION.RECORD_AUDIO
    ]);
```

21. Detección de plataformas

En el ejemplo de la imagen superior, se puede ver como se piden los permisos en Android. Como se trata de una funcionalidad específica del sistema operativo Android, será necesario detectar la plataforma y comprobar si es Android.

También, si se quiere desarrollar la aplicación para que sea usada desde navegadores o en escritorio y en móviles, será necesario asegurarse que la interfaz esté adaptada correctamente a los distintos tamaños posibles de pantalla usando diferentes elementos y diferentes tamaños y estilos en función del tamaño de la pantalla.

Por último, cualquier aplicación necesitará almacenar algún tipo de dato de forma local, ya sea un token de autenticación o el resultado de una llamada, para más tarde poder trabajar offline. Para esto, Ionic ofrece una gran cantidad de opciones mediante el uso de la librería de Ionic Storage, la cual elige la mejor solución para el entorno en el que se está ejecutando la aplicación. Entre estas opciones se encuentran bases de datos SQLite para entornos nativos, IndexedDB, WebSql o Local Storage. De esta forma, siempre se podrá la aplicación siempre podrá acceder a algún tipo de almacenamiento, siendo esto transparente para el desarrollador.

5. Desarrollo del back-end

La parte del back-end del proyecto se ha desarrollado usando Node.js con Express, un framework para Node.js que permite crear una API de forma rápida y sencilla, manteniendo el gran rendimiento que ofrece Node.js. La función de esta parte del proyecto es el de actual como intermediario o middleware entre la aplicación móvil y los servicios de IBM Watson, mientras realiza cierto procesamiento de datos, de forma que la aplicación móvil tenga que hacer menos operaciones, y por lo tanto se mejore el rendimiento.

Se ha elegido este framework debido a la facilidad para desarrollar una API relativamente potente, ya que para las necesidades del proyecto era suficiente, y por la gran escalabilidad que ofrece. Como se ha comentado previamente, al estar basado en eventos y no en hilos, aunque haya distintos clientes haciendo peticiones al servidor de forma concurrente, no llegará a bloquearse, debido a que rara vez se realizan operaciones pesadas de I/O. Además, también debido al uso de los eventos para responder las llamadas de los clientes, cuando no se esté recibiendo ninguna petición, no habrá ningún proceso ejecutándose, el servidor estará en reposo.

El servidor contará con una serie de llamadas al API que cualquier aplicación que pueda actuar como cliente podrá llamar. Estas peticiones se implementan mediante el uso del framework Express. Cada una de estas llamadas será un evento que admite el envío de datos en el cuerpo de la petición, y envía de vuelta información sobre como ha ido la llamada en el código HTTP, y el resultado producido.

Cada llamada, a su vez, realiza una llamada al API proporcionada por IBM Watson para acceder a sus servicios desde Node.js. Para esto, habrá que importar las dependencias de las librerías previamente descargadas, mediante *require()*, y posteriormente crear los objetos tienen los métodos para hacer las peticiones. Para crear estos objetos es necesario indicar la URL y las credenciales del servicio entre otros parámetros como la versión del servicio que se va a utilizar. En la próxima sección se verá como poder conocer estos parámetros.

```
var TextToSpeechV1 = require('watson-developer-cloud/text-to-speech/v1');
var SpeechToTextV1 = require('watson-developer-cloud/speech-to-text/v1');
var ToneAnalyzerV3 = require('watson-developer-cloud/tone-analyzer/v3');
var LanguageTranslatorV3 = require('watson-developer-cloud/language-translator/v3');
```

22. Importación de dependencias en Node.js

```
var text_to_speech = new TextToSpeechV1({
    username: "[redacted]",
    password: "[redacted]",
    url: "https://stream-fra.watsonplatform.net/text-to-speech/api"
});
```

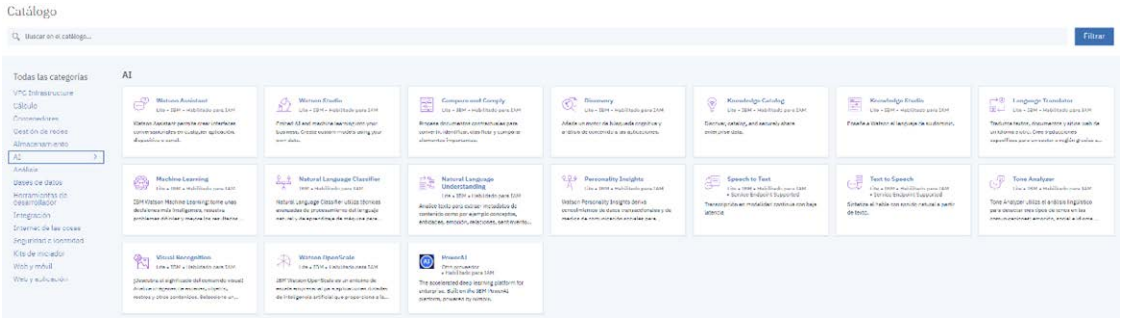
23. Creación de objetos de IBM Watson en Node.js

```
text_to_speech.synthesize(params, (err, audio) => {
    if (err) {
        response.status(500).send(`error: ${err}`);
    } else {
        response.send(audio);
    }
})
```

24. Uso de los objetos de IBM Watson en Node.js

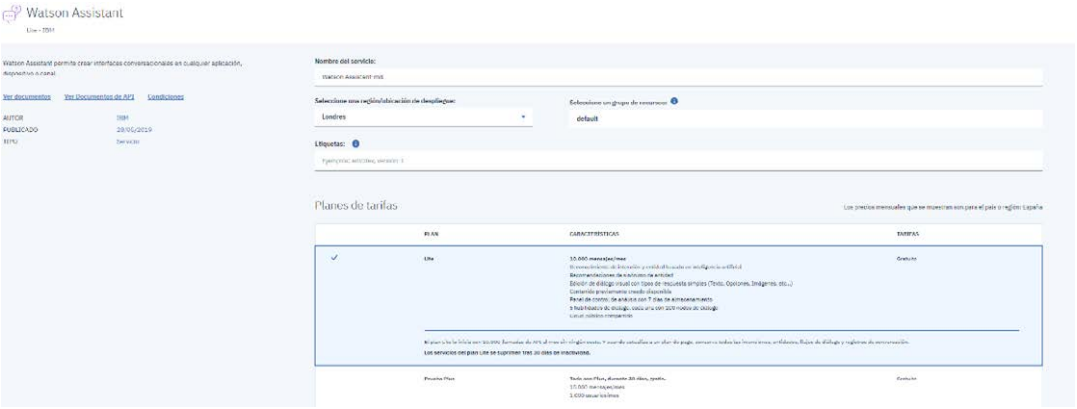
6. Desarrollo de los servicios de IBM

Para poder desarrollar y usar los servicios de IBM desde la aplicación, lo primero que será necesario hacer es crear una cuenta gratuita de IBM Cloud en https://cloud.ibm.com/registration. Una vez se ha creado una cuenta y se ha accedido de forma correcta, se mostrará un tablero con distinta información, como los servicios y aplicaciones existentes, información sobre el uso de estos o el estado de las distintas localizaciones de los servidores de IBM. Si no se había usado nunca IBM Cloud, y la cuenta es nueva, no se mostrará ningún servicio ni aplicación. Entonces, para crear nuevos recursos y aplicaciones, habrá que hacer click en el botón "Crear recurso", en la parte superior derecha de la pantalla. Esto nos llevará a una nueva pantalla en la que podremos elegir el servicio que queremos crear.



25. Catálogo de servicios de IBM

Una vez se ha elegido el servicio que se va a crear, nos lleva a una pantalla desde donde es posible acceder a la documentación del servicio, y se puede cambiar el nombre del servicio, la región del servidor en el que estará ubicado el servicio, el grupo de recursos del que formará parte el servicio y el plan de tarifas que seguirá el servicio.



26. Creación de un nuevo servicio de IBM

Por lo general, la mayoría de los servicios permiten elegir un plan de tarifas entre un plan Lite, gratis, pero con bastantes limitaciones, Estándar, con menos limitaciones, pero hay que pagar por cada llamada realizada al API, y otros planes como Plus o Premium, que ofrecen muchas más capacidades sin casi limitaciones, para lo que es necesario contactar con el personal de ventas de IBM.

Con el servicio ya creado, en el tablero general ya se mostrará el nuevo servicio que se acaba de generar. Si se hace click en ese tablero nos llevará a la lista de todos los recursos existentes en la cuenta. Al hacer click en alguno de los servicios se podrá ver algunos detalles del servicio y nos permitirá consultar la documentación del servicio, cambiar el plan del servicio y consultar las credenciales del servicio. Además, en algunos casos, como en el caso de Watson Assistant, también tendremos un botón para navegar a distintas herramientas online que nos facilita el uso y desarrollo de los servicios

27. Detalles de un servicio de IBM

Conociendo el proceso, habrá que crear el resto de los servicios que se van a utilizar en la aplicación, Watson Text to Speech, Watson Speech to Text, Watson Discovery y Watson Assistant, que ya se ha creado.

Después de haber creado el servicio, se entrará en los detalles del servicio para ver las credenciales y entrar en la documentación del servicio si se quiere conocer más a fondo la implementación y el funcionamiento.

Lo primero será crear el objeto que tiene los métodos para llamar al API de Watson. Para esto, copiamos las credenciales de los detalles de los servicios y los usamos para crear el objeto.

```
var text_to_speech = new TextToSpeechV1({
    username: "[redacted]",
    password: "[redacted]",
    url: "https://stream-fra.watsonplatform.net/text-to-speech/api"
});
```

28. Creación del objeto Text to Speech

Teniendo el objeto creado, se llama al método para llamar al API que convierte el texto en audio. Este método acepta los siguientes parámetros:

- **Text:** El texto que se va a convertir.
- **Accept:** El formato del audio que se va a obtener. Permite valores como *audio/mp3*, *audio/mpeg* o *audio/wav*.
- **Voice:** La voz que va a tener el audio. Permite elegir voz de hombre o mujer entre varios idiomas como inglés, francés, alemán, español o japonés. La voz por defecto es voz de hombre en inglés.

```
var params = {
    text: request.body.text,
    voice: request.body.person_code,
    accept: 'audio/mp3'
};

// Pipe the synthesized text to a file.
text_to_speech.synthesize(params, (err, audio) => {
    if (err) {
        response.status(500).send(`error: ${err}`);
    } else {
        response.send(audio);
    }
})
```

29. Llamada al API de Text to Speech

Después de haber creado el servicio, se entrará en los detalles del servicio para ver las credenciales y entrar en la documentación del servicio si se quiere conocer más a fondo la implementación y el funcionamiento.

Lo primero será crear el objeto que tiene los métodos para llamar al API de Watson. Para esto, copiamos las credenciales de los detalles del servicio y las usamos para crear el objeto.

```
var speech_to_text = new SpeechToTextV1({
    url: "https://stream-fra.watsonplatform.net/speech-to-text/api",
    username: "[redacted]",
    password: "[redacted]"
});
```

30. Creación del objeto Speech to Text

Una vez se tiene el objeto creado, se llama al método que reconoce el audio a través de WebSockets y lo convierte en texto. Este método admite los siguientes parámetros:

- **Model:** El modelo de banda que se va a utilizar para reconocer el audio.
- **Content-type**: El formato del audio.
- **Interim_results:** Si es *true,* el resultado será una transmisión de objetos JSON, si es *false*, el resultado será un único objeto con el resultado final.
- **Max_alternativas:** El número máximo de textos alternativos que el servicio va a devolver.
- **Word_confidence:** Si es *true*, el API devuelve un valor entre 0.0 y 1.0 que corresponde con la confianza del resultado.
- **Timestamps:** Si es *true*, el servicio devuelve marcas temporales para cada palabra.

Además, también será necesario convertir el audio que llega desde la aplicación en formato .aac y en base64 al formato .mp3 mediante el uso de Cloudconvert.

```
var params = {
    model: 'es-ES_NarrowbandModel',
    content_type: 'audio/mp3',
    'interim_results': true,
    'max_alternatives': 1,
    'word_confidence': false,
    timestamps: false
};

// Create the stream.
var recognizeStream = speech_to_text.recognizeUsingWebSocket(params);

cloudconvert.convert({
    "inputformat": "aac",
    "outputformat": "mp3",
    "input": "base64",
    "file": request.body.file,
    "filename": "prueba.aac"
}).pipe(recognizeStream);
```

31. Llamada al API de Speech to Text

Watson Discovery

Tras haber creado el servicio de Watson Discovery, se entrará en los detalles del servicio para ver las credenciales y revisar la documentación si se quiere conocer más a fondo como usar el servicio.

Lo primero que se debe hacer es crear el objeto en Node que nos permite hacer las llamadas a la API de Watson. Para esto copiamos las credenciales del servicio de Watson Discovery y las usamos para crear el objeto.

```
var discovery = new DiscoveryV1({
    version: '2019-03-25',
    iam_apikey: '[redacted]',
    url: 'https://gateway-fra.watsonplatform.net/discovery/api'
});
```

32. Creación del objeto Discovery

Una vez creado el objeto, llamamos al método que nos va a permitir buscar en una colección de documentos. Pero antes de poder hacer la llamada será necesario crear subir la colección de documentos en la nube de IBM, entrenar el modelo para obtener mejores resultados y enriquecer los documentos con campos que nos interesen, aunque IBM facilita ya una colección pública de noticias a las que se podría hacer peticiones. A pesar de que es posible hacer toda la configuración haciendo llamadas a la API del servicio, IBM proporciona una interfaz visual, a la que se puede acceder desde los detalles del servicio, para poder configurar la colección de documentos de forma más sencilla.

Al entrar en esta interfaz, lo primero que se tendrá que hacer en crear una nueva colección de datos. Para esto existen dos opciones, pulsando el botón *"Upload your own data"* permite crear una colección de datos y subir los ficheros a mano, mientras que con el botón *"Connect a data source"* nos permite importar los ficheros desde distintas plataformas como Sharepoint o IBM Cloud Object Storage. Para este proyecto, se va a elegir la opción de *"Web Crawl"*, la cual permite rastrear una URL en busca de todos los documentos y almacenarlos en una nueva colección de forma sencilla. Solo es necesario indicar la URL en la que queremos que se haga la búsqueda, cada cuanto tiempo queremos que se haga y el idioma de los documentos. Después de haber creado la colección y cargado todos los documentos, podemos añadir desde la pantalla principal de la colección campos y enriquecimientos adicionales a los que se incluyen por defecto. También es recomendable entrenar el modelo mediante la interfaz a la que se puede acceder desde la pantalla de consultas. Añadiendo nuevas consultas y consultas que se han realizado recientemente y valorando el resultado de estas consultas, es posible mejorar la confianza de las posteriores búsquedas que hagan.

Después de configurar la colección donde se van a realizar las búsquedas, ya si que es posible llamar a la API para hacer estás búsquedas desde la aplicación. El método que permite hacer la petición admite los siguientes parámetros entre otros:

- **Environment_id:** El identificador del entorno que contiene la colección.
- **Collection_id:** El identificador de la colección que contiene los documentos sobre los cuales se va a realizar la búsqueda.

- **Natural_language_query:** Consulta en lenguaje natural que devuelve los documentos más relevantes con el uso de modelos de datos entrenados y la comprensión de lenguaje natural de IBM Watson.
- **Return_fields:** Los campos que nos interesa que devuelva la consulta.
- **Count:** El número de resultados que va a devolver la petición.

```
var params = {
    environment_id: '[redacted]',
    collection_id: '[redacted]',
    natural_language_query: request.body.text,
    return_fields: 'title, extracted_metadata.title, url, text, metadata.source.url',
    count: 5
}

discovery.query(params, (err, result) => {
    if (err) {
        response.status(500).send(`error: ${err}`);
    } else {
        response.json({ res: result });
    }
})
```

33. Llamada al API de Discovery

Watson Assistant

Tras haber creado el servicio de Watson Assitant, se entrará en los detalles del servicio para ver las credenciales y revisar la documentación si se quiere conocer más a fondo como usar el servicio.

Lo primero que se debe hacer es crear el objeto que permite hacer las llamadas a la API de Watson. Para esto copiamos las credenciales del servicio de Watson Assistant y las usamos para crear el objeto.

```
var assistant = new AssistantV1({
    username: '[redacted]',
    password: '[redacted]',
    url: "https://gateway-fra.watsonplatform.net/assistant/api",
    version: '2019-02-28'
});
```

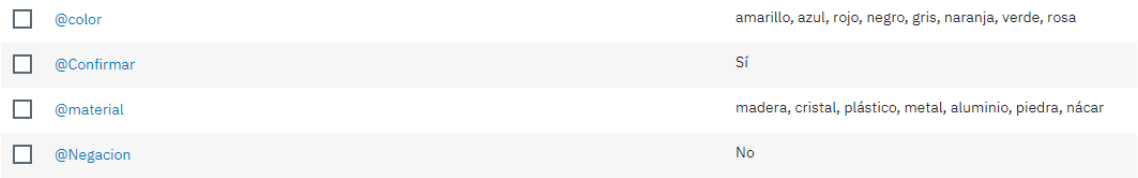
34. Creación del objeto Assistant

Después de haber creado el objeto, habrá que hacer la llamada a la API mediante los métodos que proporciona el objeto, pero será necesario antes crear una conversación para el servicio. Al igual que en Watson Discovery, se puede haciendo peticiones a la API de IBM, pero está disponible una interfaz para crearlas de forma mucho más sencilla e intuitiva. Se puede acceder a esta interfaz desde los detalles del servicio. Una vez se ha accedido será necesario crear la conversación haciendo click en el botón *"Create skill"* dentro del apartado *"Skills"*.

Ya creada la conversación hacemos click sobre ella para acceder a sus detalles. Aquí tenemos varias secciones, pero las que nos interesan para crear la conversación son *"Intents", "Entities"* y *"Dialog"*. Los intents son las intenciones de los usuarios cuando conversan con el asistente y permite reconocer sus preguntas mediante reconocimiento de lenguaje natural a partir de varios ejemplos dados.

35. Intents de la conversación

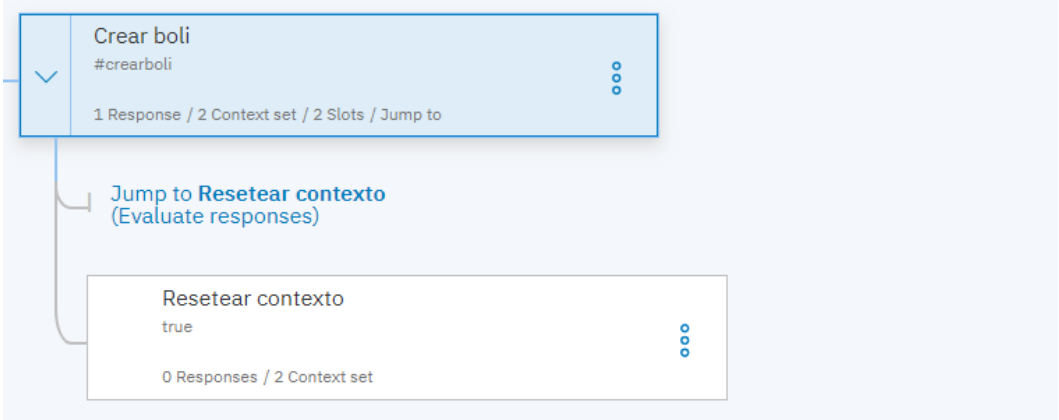
Las entities o entidades son categorías de valores que pueden ser reconocidos por el asistente para realizar ciertas operaciones que necesiten un grupo de valores específicos.



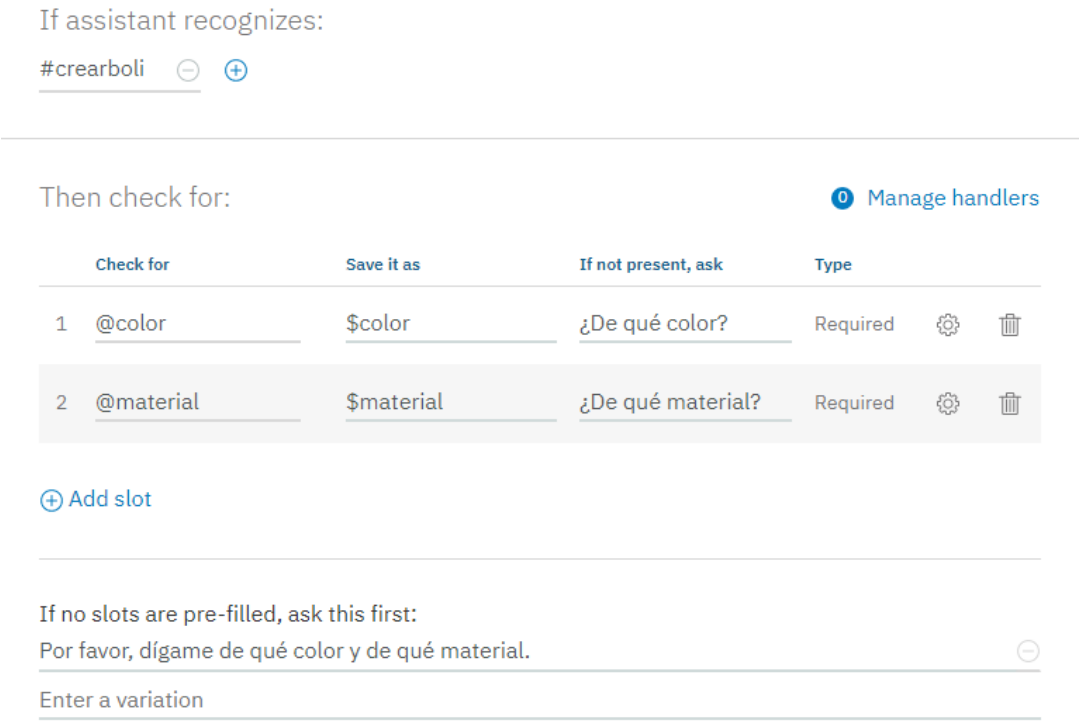
36. Entities de la conversación

En el dialog es donde se configura la conversación. La conversación se basa en el flujo a través de un conjunto de nodos en forma de árbol en función de las intenciones del usuario que está interactuando con el asistente. Cada nodo corresponde con una intención, una entidad, una variable de contexto o una condición común, que serían *welcome*, que se reconoce al iniciar la conversación y *anything_else*, donde entraría si no se encuentra un nodo correspondiente con la entrada del usuario. Los nodos pueden tener a su vez hijos, que serán recorridos en la siguiente interacción del usuario.

Este sería el aspecto de un ejemplo de nodo con hijos.



37. Ejemplo nodo Assistant



38. Configuración nodo Assistant

En este ejemplo, cuando se reconoce el intent #crearboli, se pedirá al usuario que se introduzca el color y el material del boli. Hasta que no reconozca las dos entidades, seguirá preguntado por ellas. Cuando haya reconocido todas, guardará sus valores en el contexto de la conversación y responderá al usuario. Después de esto, se saltará automáticamente al siguiente nodo que quita estos valores del contexto.

Después de haber configurado la conversación ya será posible hacer las peticiones a la API para mantener la conversación desde la aplicación. El método que se va a utilizar para esto necesita que se le pase el identificador de la conversación, el texto que ha escrito el usuario, el contexto de la conversación, donde se almacena un historial y distintas variables, y una función que se ejecutará cuando nos devuelva un resultado la API. Además, en este momento será cuando se integre Watson Assistant con Watson Discovery.

```
var text = request.body.text;
var context = request.body.context;

function processResponse(err, res) {
    if (err) {
        console.log(err);
        response.status(500).send(`error: ${err}`);
        return;
    }

    console.log(`last message: ${lastMessage}`);
    if (res.output.text == 'discoverySearch') {
        doDiscovery(lastMessage).then(result => {
            response.json({ document: result.results[0], context: res.context });
        }, error => {
            response.status(500).send(`error: ${error}`);
        })
    } else {
        lastMessage = request.body.text;
        response.json({ text: res.output.text, context: res.context });
    }

}

assistant.message({ workspace_id: workspace_id, input: { text: text }, context: context }, processResponse);
```

39. Llamada al API de Assistant

De esta forma se realiza la llamada al API con el texto y el contexto de la conversación que llegan desde la aplicación móvil. Una vez el servicio de Assistant ha contestado, se procesa el resultado. Si devuelve el texto específico *"discoverySearch"*, se realiza una búsqueda con Discovery con el último texto que el usuario ha escrito. Si no, se le manda a la aplicación móvil el resultado, el texto y el contexto.

Despliegue del back-end

Además de todos los servicios que se usan, también se desplegará el back-end en la plataforma cloud de IBM. Para esto, teniendo ya una cuenta de IBM Cloud, lo primiero será crear el contenedor donde se desplegará la aplicación de Node.js. Esto se realiza desde el tablero general, pulsando el botón *"Crear recurso"*, dentro de la categoría cálculo y se elige la opción *"SDK for Node.js"*. En la siguiente pantalla podremos elegir el nombre de la aplicación, el dominio, la región de despliegue y el espacio de recursos de la cuenta en el que estará contenido.

Después de haber creado el contenedor, para encontrarlo, dentro de la lista de recursos, elegimos la opción *"Apps de Cloud Foundry"* y dentro de esta buscamos el recurso que se acaba de crear. Dentro de este recurso podemos ver ciertos detalles sobre él, como las instancias que están ejecutándose actualmente, la cantidad de memoria que tiene cada instancia, el coste que tiene el tiempo que lleva ejecutándose la aplicación o un registro de errores.

Una vez tenemos el recurso creado, habrá que desplegar el código que se ha desarrollado en el recurso que hemos creado. Para esto, desde la consola de comandos, en el directorio donde hemos desarrollado la aplicación de Node, debemos iniciar sesión en IBM Cloud con el siguiente comando: "ibmcloud login -u *nombre de usuario* -o *organización* -s *espacio"*. Después de esto, para desplegar la aplicación se usa el comando "bluemix app push *"nombre del recurso"*. Si se ha desplegado correctamente, ya podemos realizar peticiones desde la aplicación a la URL que se indica en el recurso.

7. Resultado

Una vez se ha desarrollado e integrado tanto la parte del front-end, del back-end y los servicios de IBM, ya es posible ver el resultado del proyecto entero. Para esto será necesario compilar y ejecutar la aplicación tanto para Android como para iOS.

Android

Para generar el proyecto de Android y posteriormente compilarlo, es necesario ejecutar desde la línea de comandos, en la carpeta de la aplicación, el comando *"ionic cordova platforms add android"*, para añadir la plataforma, el concepto de aplicación para cordova. Esto nos creará el proyecto de Android con todos los plugins que hayamos añadido al proyecto anteriormente. Después para compilarlo y ejecutarlo a un dispositivo móvil o emulador, se ejecuta el comando *"ionic cordova run android"* o *"ionic cordova run android -l"* para soportar los cambios en caliente si todavía se están haciendo cambios en la aplicación.

Cuando ya haya finalizado la aplicación, o se quiera subir a la Play Store, será necesario generar un APK, el paquete que contendrá nuestra aplicación para Android. Para esto será necesario tener el SDK de Android instalado y las rutas del JDK de Java, del SDK de Android y de Gradle correctamente configuradas. Usando el comando *"ionic cordova build android --prod --release"* se generará un fichero APK que contendrá la aplicación sin firmar. Para firmar la aplicación primero será necesario generar una clave privada. Para generar esta clave, es necesario ejecutar el comando *"keytool -genkey -v -keystore my-release-key.keystore -alias alias_name -keyalg RSA -keysize 2048 -validity 10000*" desde el directorio de la aplicación, lo que generará un fichero de extensión .keystore, que será necesario guardar para futuras actualizaciones. Para firmar el APK, copiamos estos dos ficheros a la carpeta *bin* dentro del JDK que tengamos instalado y ejecutamos el comando *"jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore my-release-key.keystore app-release-unsigned.apk alias_name"* desde esta carpeta. Por último, para optimizar el APK, lo movemos al directorio *build-tools/version/zipalign* dentro del SDK de Android y ejecutamos desde allí el comando *"zipalign -v 4 app-release-unsigned.apk app-release.apk"*. Con esto ya tenemos el APK listo para subirlo al Google Play.

El aspecto final de la aplicación en Android sería la siguiente.

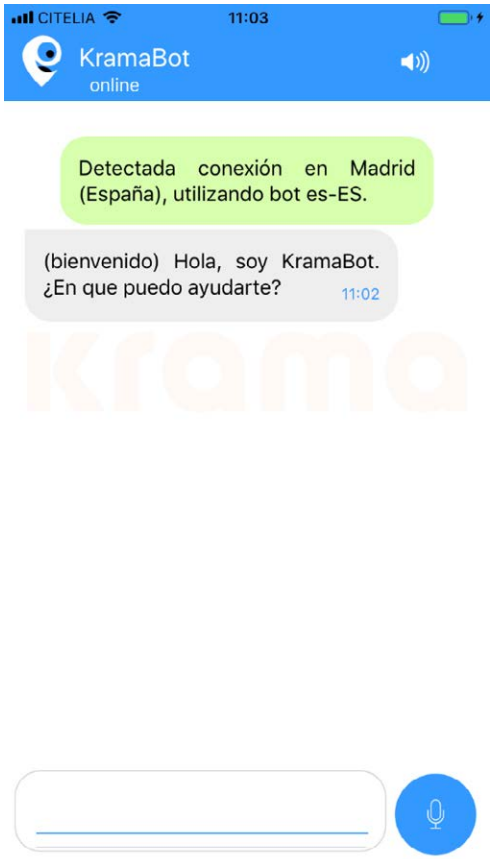
40. Interfaz en Android

iOS

Para generar el proyecto de iOS y, posteriormente compilarlo, al igual que con Android será necesario ejecutar el comando *"ionic cordova platforms aad ios"* en la raíz de la aplicación para añadir el proyecto de iOS al proyecto. Al igual que el proyecto de Android, contará con todos los plugins que se hayan instalado previamente. Una vez se tiene el proyecto creado, para compilarlo y ejecutarlo es posible hacerlo desde la línea de comandos con el comando *"ionic cordova run ios"* o compilarlo con el comando *"ionic cordova build ios"* y abrir el proyecto con Xcode y ejecutarlo desde allí en un dispositivo móvil o en un emulador.

Para generar una versión para el despliegue, se puede ejecutar el mismo comando de antes, *"ionic cordova build ios --prod"* para generar el proyecto con el código de la parte web minificado. Después, es posible seguir la documentación de Apple para generar y firmar los certificados necesarios y subir la aplicación al App Store, pero debido a que me he centrado en Android, para este proyecto no se ha llegado a generar certificados, simplemente se ha ejecutado la aplicación en un dispositivo móvil.

El aspecto final de la aplicación en iOS sería el siguiente.



41. Interfaz en iOS

Conclusiones

Como conclusión, después de estos meses en los que se ha desarrollado el proyecto, se podría decir que he adquirido una gran cantidad de conocimientos, no tanto de Ionic, que ya llevaba usándolo bastantes meses, sino de Node.js, nuevas funcionalidades de Angular que se introducen en Ionic 4 y, sobre todo del uso de la plataforma y los servicios de IBM Cloud.

El desarrollo de este proyecto puede aportar diferentes soluciones para muy diversos problemas, enmarcadas dentro de la evolución de la interacción de las personas con los dispositivos electrónicos, como podría ser el obtener una mayor accesibilidad para el uso de aplicaciones móviles gracias al reconocimiento de voz.

De momento, el proyecto se ha desarrollado a modo de demo, de una forma que es sencillo realizar cambios pequeños en cualquiera de las partes sin que afecten demasiado a las demás. De esta forma, mientras se vaya mostrando el proyecto a distintos clientes interesado y hablando con ellos, se podría ir mejorando y cambiando ciertos aspectos según sus necesidades y el feedback recibido por su parte.

Por último, se comentará el por qué se han elegido las tecnologías usadas y algunas de sus ventajas. Por ejemplo, Ionic es un framework híbrido, lo que significa que desarrollas aplicaciones para distintas plataformas, de forma que, aunque no tenga la misma calidad que una aplicación nativa, se puede ahorrar gran parte del presupuesto a la vez que se genera más beneficios por tener la aplicación desplegada en varias plataformas. Node.js permite el desarrollo de APIs a las que conectar las aplicaciones móviles de forma muy fácil, además de que, al no estar basado en hilos, es más eficiente que otras tecnologías y no puede llegar a bloquearse en caso de una gran concurrencia de usuarios. Para terminar, la plataforma cloud de IBM, es una de las plataformas más avanzadas en el tema de la inteligencia artificial y proporciona una gran cantidad de servicios de inteligencia artificial y de procesado muy útil para el proyecto desarrollado.

Mejoras futuras

Como posibles mejoras a realizar en un futuro se podrían considerar varios desarrollos debido a la naturaleza del software, con el que siempre se tiene la necesidad de evolucionar y mejorar en el tiempo ya sea para cumplir con las necesidades de los usuarios actuales o para captar usuarios nuevos frente a la gran competencia que existe en el mercado de las aplicaciones móviles.

Entre estas posibles mejoras se encuentran las mencionadas que se podrían obtener gracias al feedback de los clientes. Además de estas mejoras se podrían incluir las siguientes:

- Conexión del back-end con una base de datos en la que se incluyan por lo menos distintos usuarios para poder autenticarse y los identificadores de diferentes conversaciones en función del cliente.
- Soporte para varios idiomas, tanto para texto como para voz, con el uso de otros servicios de IBM.

Bibliografía

Apache. (s.f.). Documentation. Recuperado el 18 de mayo de 2019 de https://cordova.apache.org/

Batanero, A. y Sesma, M. (10 de julio del 2017). VERSUS: Apps híbridas VS Apps Nativas [Mensaje en un blog] Recuperado el 15 de mayo de 2019 de https://www.paradigmadigital.com/dev/versus-apps-hibridas-vs-apps-nativas/

Hernández Fernández, A. (s.f.). Ionic: historia de uno de los principales frameworks visuales. Recuperado el 15 de mayo de 2019 de https://www2.deloitte.com/es/es/pages/technology/articles/Ionic-principales-framework-visuales.html

IBM. (s.f.). Cloud. Recuperado el 12 de junio de 2019 de https://www.ibm.com/cloud/

InnovaAge. (s.f.). Apps Híbridas vs Nativas vs Generadas. ¿Qué decisión tomar? Recuperado el 20 de mayo de 2019 de https://www.innovaportal.com/innovaportal/v/696/1/innova.front/apps-hibridas-vs-nativas-vs-generadas-que-decision-tomar

Ionic. (s.f.). A showcase of the most beautiful apps built with Ionic. Recuperado el 6 de junio de 2019 de http://showcase.ionicframework.com/apps/top

(s.f.). Ionic Framework. Recuperado el 6 de junio de 2019 de https://ionicframework.com/docs

NetConsulting. (30 de septiembre de 2015). Node.js: ¿Qué es y para que sirve NodeJS? Recuperado el 28 de mayo de 2019 de https://www.netconsulting.es/blog/nodejs/

Node.js. (s.f.). Documentación. Recuperado el 28 de mayo de 2019 de https://nodejs.org/es/

TypeScript. (s.f.). Documentation. Recuperado el 20 de mayo de 2019 de https://www.typescriptlang.org/

TypeScript. (s.f.). En Wikipedia. Recuperado el 20 de mayo de 2019 de https://es.wikipedia.org/wiki/TypeScript

ANEXO A: Aspectos éticos, económicos, sociales y ambientales

1. Introducción

Los principales objetivos del proyecto son el de hacer una aplicación para Android y para iOS que permita realizar distintas operaciones que se harían mediante click en otras aplicaciones mientras se mantiene una conversación tanto de forma escrita como oral, dentro de un marco de gran desarrollo tanto de las aplicaciones móviles como de nuevas formas de interacción humano-máquina.

2. Descripción de impactos relevantes relacionados con el proyecto

En relación a los impactos más relevantes que conlleva el desarrollo de este proyecto, se podría hablar de la privacidad de los usuarios. También podría considerarse como impacto negativo el gasto eléctrico que supone el uso de ordenadores, dispositivos móviles y otros aparatos para el desarrollo del proyecto, como en cualquier otro desarrollo de software.

3. Análisis detallado de alguno de los principales impactos

Algunos de los usuarios menos familiarizados con la tecnología podrían considerar que al reconocer lo que escriben o lo que su habla, es una forma de restringir su privacidad y que los audios que mandan van a ser almacenados. Por eso sería importante asegurar a los usuarios que los audios que envíen no se van a almacenar en ningún sitio, y que solo se usarán en ese momento para reconocer el habla sin llegar a reconocer la identidad de las personas. También sería interesante el impulsar el uso de electricidad generada de forma limpia y respetuosa con el medioambiente.

4. Conclusiones

Desde mi punto de vista, quitando los dos puntos mencionados en los dos apartados anteriores, no se trata de un proyecto que afecte de manera muy negativa en los distintos aspectos. Pero el hecho de utilizar recursos y electricidad de forma respetuosa con el medioambiente si que podría proporcionar un gran valor en un amplio sector de personas, en especial en la actualidad, con la concienciación general de la sociedad frente al progresivo cambio climático que se está produciendo.

ANEXO B: Presupuesto económico

COSTE DE MANO DE OBRA (coste directo)	Horas	Precio/hora	Total
	200	10 €	**2.000 €**

COSTE DE RECURSOS MATERIALES (coste directo)	Precio de compra	Uso en meses	Amortización (en años)	Total
Ordenador portátil Windows	1.300 €	6	5	130 €
Ordenador portátil MacOS	1.500 €	6	5	150 €
Smartphone Android	500 €	6	5	50 €
Smartphone iOS	700 €	6	5	70 €

COSTE TOTAL DE RECURSOS MATERIALES	**400 €**

GASTOS GENERALES (costes indirectos)	15%	sobre CD	**360 €**
BENEFICIO INDUSTRIAL	6%	sobre CD+CI	**166 €**

SUBTOTAL PRESUPUESTO		**2.926 €**
IVA APLICABLE	21%	**614 €**

TOTAL PRESUPUESTO	**3.540 €**

ANEXO C: Siglas y acrónimos

- **IBM:** International Business Machines Corporation, es una reconocida empresa multinacional estadounidense de tecnología y consultoría con sede en Armonk, Nueva York.
- **iOS:** iPhone OS.
- **IoT:** Internet of Things.
- **HTML:** HyperText Markup Language.
- **CSS:** Cascading Style Sheets.
- **SGML:** Standard Generalized Markup Language.
- **SDK:** Software Development Kit.
- **API:** Application Programming Interface.
- **LTS:** Long Term Support.
- **CLI:** Command Line Interface.
- **JSX:** JavaScript XML.
- **XML:** Extensible Markup Language.
- **PWA:** Progressive Web Application.
- **IDE:** Integrated Development Environment.
- **SPA:** Single Page Application.
- **MVC:** Model View Controller.
- **ARIA:** Accessible Rich Internet Applications.
- **DOM:** Document Object Model.
- **URL:** Uniform Resource Locator.
- **I/O:** Input/Output.
- **SQL:** Structured Query Language.
- **NoSQL:** Not only SQL.
- **US:** User Story.
- **HTTP:** Hypertext Transfer Protocol.
- **APK:** Android Application Package.
- **JDK:** Java Development Kit.