UNIVERSIDAD DE EXTREMADURA

ESCUELA POLITÉCNICA

UNIVERSIDAD DE EXTREMADURA

Escuela Politécnica

Ingeniería De Telecomunicación

Trabajo Fin de Máster

Diseño y desarrollo de una interfaz de comunicación con los elementos del entorno para personas con discapacidad motora severa

UNIVERSIDAD DE EXTREMADURA

Escuela Politécnica

Ingeniería De Telecomunicación

Trabajo Fin de Máster

Diseño y desarrollo de una interfaz de comunicación con los elementos del entorno para personas con discapacidad motora severa

Autor: Jorge Antonio Flores Román

Tutor: Rafael Martín Espada

ÍNDICE GENERAL DE CONTENIDOS

ÍNDICE DE FIGURAS

RESUMEN

En el presente trabajo se ha diseñado y desarrollado un sistema software de ayuda a personas con discapacidad (SAPD) por el cual estas personas deben ser capaces de interactuar con el entorno electrónico inmediato (elementos del domicilio) superando las limitaciones que su condición les impone.

Para ello, este sistema de ayuda SAPD se ha diseñado teniendo en cuenta las diferentes formas de interactuar con los dispositivos que conforman el entorno, a saber: mediante la comunicación por voz, de manera visual, mediante la fijación del iris, y por medio de la comunicación mediante estímulos táctiles. De esta forma, dependiendo del tipo de discapacidad, la persona podrá comunicarse y relacionarse con los diferentes elementos instalados en su propio domicilio, ayudándole a lograr una mayor independencia y garantizando, al mismo tiempo, su seguridad.

Aunque existen muchos tipos de discapacidad, la concepción y diseño de este sistema de ayuda se han enfocado para personas con una discapacidad motora avanzada ya que este tipo de discapacidad impide a la persona prácticamente valerse por sí misma en ninguna de las actividades habituales. Esta falta de independencia puede ir acompañada de un sufrimiento de la persona que lo padece y de sus familiares, que redunda en una baja autoestima y una pérdida de dignidad en la existencia que todo ser humano exige.

Es esta la razón por la que el proyecto se justifica prácticamente por sí solo, para aliviar en la medida de los posible las dificultades de este tipo de personas que puedan decidir y ejecutar determinadas tareas por sí mismas, permitiéndoles un control más extensivo sobre las acciones más cotidianas. En definitiva, convertirles en personas más autónomas y libres.

El sistema de ayuda (SAPD) que se ha desarrollado en el marco de este proyecto para este fin consta de los siguientes elementos básicos:

- Una interfaz gráfica sencilla que actúa como cliente, desarrollada en Java, para la comunicación con un uControlador.
- Aplicación realizada con Android y que permite la interacción por voz con el uControllador. Dispone de idénticas funcionalidades que las realizadas en el cliente Java.

- uContrador Arduino, el cual actúa como servidor para recibir las órdenes de la parte del cliente y ejecutar las mismas. (No incluida en esta parte del proyecto).

En los apartados siguientes se abordarán los problemas principales que acarrea el desarrollo de este tipo de sistemas, tratando de indagar en el porqué de la escasez de plataformas completas en un siglo XXI colapsado por innovaciones que no acaban de abordar problemas graves, aunque minoritarios, debido principalmente a la propia estructura económica que enmarca el desarrollo tecnológico en nuestros días.

PALABRAS CLAVE

Android, Java Fx, FXML, Discapacidad, W3C, SmartPhone, IrisBond, IoT, Accesibilidad, Arduino, Servidor, Cliente, Agile, Scrum, Trello, Git, MVC, IntelliJ IDEA, Interfaz de Usuario, Icono, UML, Inyección de dependencias, Hilo, Prueba, Conexión, Empaquetar, Ruta, Android Studio, API, Dispositivo, Entorno, Actividad, XML, Evento, Reconocimiento, Permiso, Puerto, IP, Preferencias, Terminal, Tarea, Ejecución, Ayuda.

ABSTRACT

The present document shows how a software system for physically disabled people has been designed and developed. By using it, they will be able to interact with their immediate surroundings (i.e. their home), overcoming the challenges imposed by their condition.

To do this, this aid system features different ways of interacting with devices present in their environment, such as using voice commands, visual interactions or by means of a touch screen. This way, depending on the particular type of disability, the user will be able of communicating and interact with different elements installed at his home, so that he can be more independent while his safety is guaranteed.

Even though several kinds of disabilities exist, this project is focused on aid systems for people with an advanced motor disability, since in this particular case the person would be heavily impeded, generating a lack of independence which can be linked to suffering low self-esteem.

That is the main goal of this project. Reducing, as much as possible, the difficulties experienced by people suffering from motor disabilities, so that they can decide and execute actions on their own, providing them with a greater control over day-to-day tasks. This is, in the end, providing them with freedom an autonomy.

The aid system developed in the scope of the project features the following basic elements:

- A basic Graphic User Interface acting as a client, developed using Java, communicating with a microcontroller.
- An Android app, allowing voice command-based interactions with the microcontroller, featuring identical functionalities to those implemented in the Java client.
- An Arduino microcontroller, acting as a server receiving commands from the client and executing them. (Not included in this part of the project).

Along the following sections, those main problems involved in the development of this kind of helping systems will be presented, trying to find out why there is a serious shortage of complete platforms in a 21st century, indeed collapsed by innovations that

have not just addressed such serious problems, although minority, mainly due to the economic structure that define the technological development in our days.

KEYWORDS

Android, Java Fx, FXML, Disability, W3C, SmartPhone, IrisBond, IoT, Accessibility, Arduino, Server, Client, Agile, Scrum, Trello, Git, MVC, IntelliJ IDEA, User Interface, Icon, UML, Dependency Injection, Thread, Test, Connection, Package, Path, Android Studio, API, Device, Environment, Activity, XML, Event, Recognition, Permission, Port, IP, Preferences, Terminal, Task, Execution, Help.

1. INTRODUCCIÓN

El Trabajo Fin de Máster que se expone a continuación pretende mostrar la ayuda que pueden ofrecer las nuevas tecnologías a las personas que presentan una discapacidad física importante.

Este trabajo se ha llevado a cabo en dos partes bien distinguidas:

- Una parte Software, que facilita a la persona impedida realizar la necesaria interacción con los dispositivos móviles y ordenadores de forma fácil y amena, con los medios a su alcance.
- Otra parte Hardware, que permitirá ejecutar la acción final seleccionada por la persona discapacitada sobre el entorno físico en el que se desenvuelve y satisfacer el deseo expresado.

Ambas partes se han diferenciado y separado funcionalmente para poder ser realizadas por dos alumnos distintos, por la consecuente extensión del proyecto completo y los ámbitos tan diversos que conllevan.

Los trabajos necesarios para el desarrollo de toda la parte Software es el objeto específico de este Trabajo Fin de Máster, por lo que es la que se expondrá a continuación.

2. OBJETIVOS

El objetivo del trabajo en todo su conjunto es ayudar a una persona con discapacidad motora avanzada. La persona modelo en la que se han basado los trabajos y diseños tiene una enfermedad denominada esclerosis múltiple, en un estado muy avanzado, que sufre de parálisis en todas las extremidades y tronco por debajo del cuello, lo que le impide desenvolverse de forma correcta en el entorno con el que interactúa. Por ello se ha querido desarrollar un sistema de ayuda para personas discapacitadas en su grado máximo, que posteriormente se podrá adaptar a los diferentes niveles de forma sencilla. Dicho lo cual, en virtud de la tecnología, esta persona, y muchas más, podrán interactuar en determinados contextos por diversas vías de comunicación: mediante órdenes vocales y mediante pulsos por fijación del iris.

Como se comentaba en la introducción, el propósito de la labor que nos ocupa requiere la realización de un sistema compuesto por un bloque software (parte de este proyecto) y un segundo bloque hardware (proyecto anexo), que interactúan entre sí para coordinar las actuaciones deseadas por la persona con discapacidad.

Los principales objetivos de este proyecto se reducen a cubrir mediante el desarrollo coordinado de dicho sistema la infraestructura completa para comunicarse con dos elementos básicos de bienestar, que podrán ampliarse de forma inmediata a otros actuadores, como son la televisión (encender, apagar, subir y bajar volumen, cambiar de canal) y el sistema de aire acondicionado de forma que, a través de un puerto genérico (puede ser usado para otro tipo de aparatos) se pueda facilitar su encendido y apagado y su configuración más básica.

3. ANTECEDENTES / ESTADO DEL ARTE

Según la OMS (Organización Mundial de la Salud) "la discapacidad es un término general que abarca las deficiencias, las limitaciones de la actividad y las restricciones de la participación.

Las deficiencias son problemas que afectan a una estructura o función corporal; las limitaciones de la actividad son dificultades para ejecutar acciones o tareas, y las restricciones de la participación son problemas para participar en situaciones vitales.

Por consiguiente, la discapacidad es un fenómeno complejo que refleja una interacción entre las características del organismo humano y las características de la sociedad en la que vive."[1]

Habiendo señalado la complejidad y la amplitud asociada al término de discapacidad, se han realizado diferentes clasificaciones, con el fin de acotar el término y poder desarrollar soluciones mejor adaptadas a cada circunstancia. En efecto, cada tipo de discapacidad puede limitar o afectar las diferentes acciones o interacciones de un individuo en concreto de forma particular, dependiendo de la caracterización de la misma y del grado de afectación, dándose un sinfín de estados circunstanciales.

[1] Texto Citado OMS

3.1 Tipos de discapacidad

La Organización Mundial de la Salud establece los diferentes tipos de discapacidad según la forma en la que afecte a un individuo. Pasamos a enumerarlas a continuación:

- Discapacidad Sensorial. Dentro de este grupo, se distinguen dos tipos:
  - Auditiva: La discapacidad auditiva es el déficit total o parcial de la percepción que se evalúa de la pérdida del audio en cada oído.
  - Visual: La discapacidad visual es la disminución parcial o total de la vista.
- Discapacidad intelectual: La discapacidad intelectual es aquella que presenta una serie de limitaciones en las habilidades diarias que una persona aprende y le sirven para responder a distintas situaciones en la vida.
- Discapacidad Psíquica: La discapacidad psíquica es aquella que está directamente relacionada con el comportamiento del individuo.
- Discapacidad Física o Motora: La discapacidad física es aquella que ocurre al faltar o tener invalidada una parte del cuerpo, lo cual impide a la persona desenvolverse de la manera convencional.

Resulta obvio que la discapacidad, en último término, se traduce en una desventaja del individuo que la sufre a la hora de desenvolverse en el entorno donde vive. Esto es debido a los obstáculos sobrevenidos asociados al tipo de discapacidad que posea y al diseño de los elementos sin tener en cuenta a este gran grupo de personas.

A lo largo de la historia moderna, la sociedad ha intentado solucionar la brecha existente entre una persona con discapacidad y una persona sin ella. Para ello, se ha intentado usar la normalización de los entornos inmediatos (hogar, centros de trabajo, escuela, etc...) y también de los entornos comunitarios (transporte, comunicaciones, etc...) con disposiciones de accesibilidad, reciente participación a la hora de ejercer el voto para discapacitados intelectuales[2], etc…

[2] Las personas con discapacidad intelectual pueden votar en España en las elecciones de 2018

En los últimos años, gracias al auge tecnológico que hemos vivido, se han conseguido derribar muchas de las barreras con las que se encontraba habitualmente una persona con discapacidad. También se han promovido políticas sociales para conseguir una integración de los nuevos entornos y canales de comunicación, con el fin de evitar aislar a este grupo de personas de un mundo cada vez más tecnológico.

3.2 Discapacidad en ESPAÑA

Los últimos datos del INE (Instituto nacional de estadística) sobre el número de discapacitados en España fueron de 2008, lo que ya de por sí evidencia una cierta dejadez de los gestores en este ámbito, aunque bien es cierto que se anuncia una gran encuesta para este año (Ref 7.42). En aquel momento había unos 3,85 millones de personas con algún tipo de discapacidad en España.[3] Conviene destacar que la definición que el INE ofrece es bastante lasa y es aquella reconoce a una persona con discapacidad como una persona con una limitación para desarrollar determinadas actividades.

Existe otra organización, el Instituto Nacional de Servicios Sociales (Imserso), que tiene una base de datos mucho más elaborada y actualizada, razón por la que seguramente el INE no se haya prodigado mucho en estos estudios. Según este informe, a finales de 2.016, la base de datos contenía un total de 4.563.749 registros correspondientes a personas valoradas y de ellas, 3.378.622 corresponden a aquellas personas que por haber obtenido un grado de discapacidad igual o superior al 33%, se consideran personas con discapacidad.

La conclusión obvia de estas cifras es que no es en absoluto un colectivo menor y que cualquier avance tecnológico en este ámbito beneficia a tal número de personas que nos debe animar a insistir en la eliminación de barreras y en la diseño e integración de un nuevo entorno donde la discapacidad deje de ser un elemento determinante por su integración efectiva. (Ref 7.43).

[3] Datos Oficiales INE

3.3 Discapacidad en el mundo

Según la OMS el número de personas con algún tipo de discapacidad a nivel mundial es de aproximadamente 1.000 millones de individuos, y este número va en aumento debido al envejecimiento de la población, enfermedades crónicas tales como diabetes, enfermedades cardiovasculares, cáncer y trastornos mentales.

3.4 Un mundo (Internet) accesible para todos

Un ejemplo de normalización de los entornos inmediatos es Internet, seguramente animados por la idea de una red accesible para todos, que potenció los esfuerzos en la accesibilidad, usabilidad, etc... Existe una serie de directrices que se deben llevar a cabo a nivel internacional para lograr que los medios electrónicos sean abordables para todos los usuarios. Por ello, el World Wide Web Consortium (W3C) dispuso unas Directrices sobre la accesibilidad de los contenidos en la web, que se publicaron el 11 de diciembre de 2008, con el fin de que los desarrolladores web crearan sitios que cubriesen las necesidades de los usuarios con discapacidad de una manera más efectiva.

La Convención sobre los derechos de las personas con discapacidad, que entró en vigor el 3 de mayo de 2008, también subraya en el artículo 9 que se debe promover que las personas con discapacidad tengan acceso a los nuevos sistemas y tecnologías de la información y las comunicaciones, incluidas Internet.

Aunque se ha intentado estandarizar la accesibilidad para cualquier persona, es evidente que, dependiendo del tipo de capacidades que ésta tiene se logrará en diferente medida el que esa persona pueda hacer uso de las nuevas tecnologías, evitando los obstáculos inherentes al diseño original de las mismas.

Con todo lo visto hasta ahora, se plantea un reto que debemos afrontar como sociedad, ya que el desarrollo humano debe conseguir una verdadera igualdad, independientemente de la raza, religión, sexo, clase social a la que pertenezca y por supuesto, de las capacidades que atesore. Como se mencionaba anteriormente, aunque se han conseguido muchos avances, aún falta una larga trayectoria para acortar las distancias que hay, en el ámbito de interacción con su entorno, entre una persona con capacidades reducidas y diferentes a una persona que las tiene según el desarrollo estándar.

3.5 Tecnología y discapacidad en el mundo laboral

Según un informe de la Fundación Adecco[4], en 2017 las personas con discapacidad siguen sufriendo desigualdad dentro del mercado laboral. En este informe se da a conocer que de todas las personas con discapacidad en edad laboral (1.335.100), sólo una de cada 4 trabaja (25,7%), frente al 58,2% de la población general.

Esto puede ser debido a los prejuicios que siguen existiendo en nuestra sociedad, condenando a una persona discapacitada con la inactivad, sumándose a las barreras propias de las mismas, las barreras por parte de las empresas, carencia de recursos para la búsqueda de empleo, etc…

Figura 1. Empleo personas discapacidad. (Ref 7.7)

En este estudio también se expone un dato alentador. Las nuevas tecnologías están consiguiendo acercar a las personas con discapacidad al mundo laboral. Gracias a aplicaciones inmediatas de búsqueda de empleo, portales especializados y otras tantas como Skype para realizar entrevistas, se consigue eliminar el factor desplazamiento, por ejemplo. Además, las nuevas tecnologías facilitan otras variables para ayudar a personas con discapacidad a desempeñar diferentes puestos de trabajo, lo cual, no sería posible sin la integración de las mismas en la sociedad actual.

[4] Entidad sin ánimo de lucro para la inclusión laboral de personas con discapacidad, parados de larga duración, mujeres víctimas y aquellas personas que por diversas circunstancias se encuentran en situación de riesgo de exclusión social o especial vulnerabilidad

Aquí aparecen numerosas ayudas para las personas con discapacidad tales como teclados con letras de gran tamaño, ratones virtuales, lectores de pantalla, impresoras braille, lupas, apps de intérprete de lenguaje de signos etc…

Además, una gran ayuda de las nuevas tecnologías que también aparece reflejada en este informe es el teletrabajo. Gracias a ello, personas con discapacidad pueden desenvolverse mejor en un entorno como su domicilio (en ocasiones más adaptados para ellos que el entorno laboral) y desempeñar el mismo trabajo que en la oficina.

Lamentablemente, un dato de este informe afirma que un 27% de los encuestados con discapacidad podría teletrabajar pero su empresa le exige presencia en la oficina.

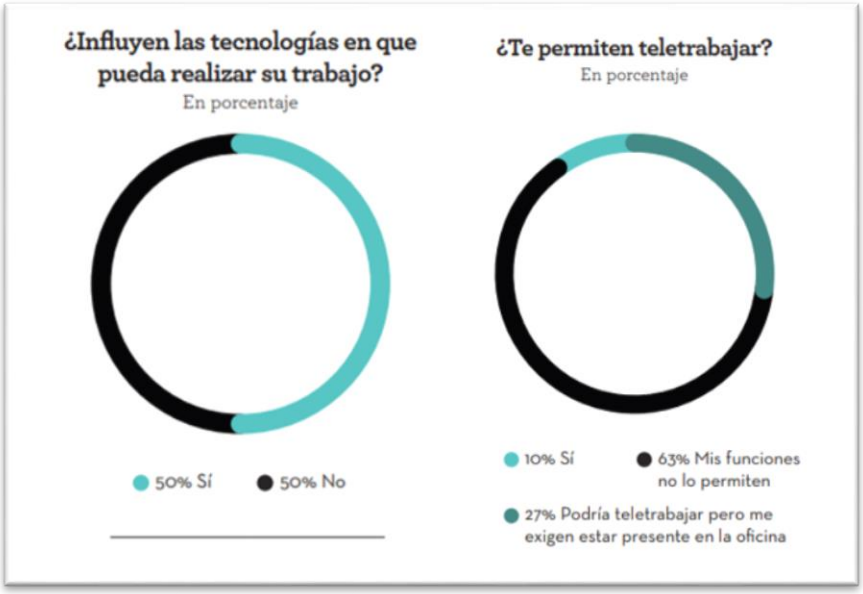


Figura 2. Tecnología y Teletrabajo. (Ref 7.7)

Otro dato que es claramente esperanzador según el informe de la Fundación Adecco relacionando personas con discapacidad y las nuevas tecnologías es el siguiente: gracias a las nuevas tecnologías, la calidad de vida de las personas con discapacidad ha mejorado. Ejemplos de mejora: Internet permite el estudio desde casa, app de gestión de citas médicas, apps de rutas por la ciudad teniendo en cuenta la accesibilidad, apps para localizar plazas de parking adaptadas, apps controladas por voz, etc…

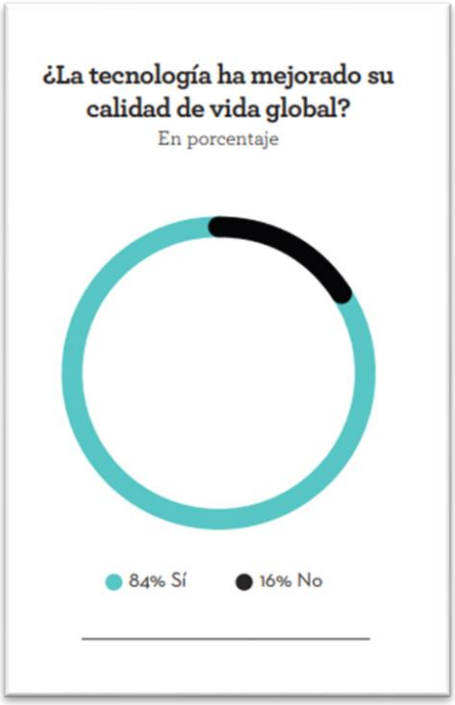


Figura 3. Tecnología y mejora de calidad de vida. (Ref 7.7)

Además, por su comodidad y su forma de uso, los Smartphones y tabletas se han vuelto una ayuda necesaria para las personas con discapacidad. Podemos ver en el siguiente grafico de la encuesta realizada el uso de las apps según la discapacidad de la persona:

Figura 4. Apps para personas con discapacidad. (Ref 7.7)

Aun así, viendo los datos reflejados en este informe, también la accesibilidad universal sigue siendo un reto ya que la mayoría de los encuestados (el 80%) afirma que el creciente desarrollo tecnológico no ha ido acompañado de medidas de accesibilidad en el entorno.

Un ejemplo de ello es el transporte público:

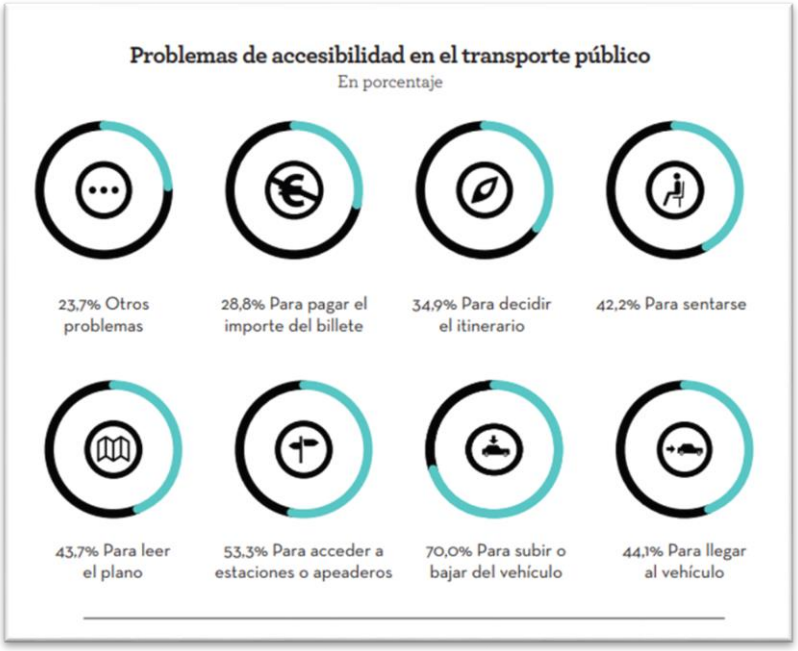


Figura 5. Accesibilidad transporte público. (Ref 7.7)

Por lo tanto, como conclusión del informe, podemos afirmar que, aunque se ha avanzado mucho en las mejoras de la integración de personas con disparidad en aspectos laborales y en mejoras de calidad de vida, se debe seguir usando la tecnología para atacar estas diferencias de accesibilidad a las que los discapacitados siguen expuestos.

3.6 Discapacidad Motora y avances históricos:

En la clasificación anterior de discapacidad de la OMS, la discapacidad motora aparece como una de las más extendidas. Ya sea por accidente o por enfermedades degenerativas, este tipo de discapacidad incapacita a la persona que la sufre para la interactuación normal con el entorno, aunque mantengan el resto de capacidades intactas.

Y es que la discapacidad motora es una de las discapacidades más restrictivas para que una persona pueda ser completamente independiente hoy en día. Hay enfermedades neurodegenerativas que pueden mermar o destruir las posibilidades de mover las extremidades y hacer que este tipo de personas no sea capaz de interactuar con prácticamente ningún elemento táctil.

Para intentar solventar esto se ha creado una serie de ayudas técnicas con el fin de establecer en primer lugar una comunicación efectiva y, tras la acción de actuadores, ayudar a esa persona a desenvolverse en contextos delimitados.

A continuación, se muestran algunas de las diferentes ayudas técnicas que se han ido realizando a lo largo de la historia para las personas con discapacidad motora:

- Tableros de comunicación: Es uno de los elementos más básicos e intuitivos a la hora de entablar comunicación con una persona con discapacidad avanzada. Hay enfermedades como el ELA (Esclerosis Lateral Amiotrofica), que impiden a la persona ejercer movimientos de toda la musculatura corporal, incluido los músculos que ayudan al habla. Es por ello que se han diseñado tableros con pictogramas para que la persona con discapacidad pueda señalar bien con la mirada, un láser, etc. Un tablero en el cual se encuentran letras, palabras más usuales, frases cortas, etc. Algunos ejemplos de estos tableros son: Comunicador tipo E-TRAN (facilita la comunicación cara a cara mediante la mirada) el cual puede verse en la *Figura 6. Tablero E-TRAN*

Figura 6. Tablero E-TRAN. (Ref 7.44)

También tenemos los tableros llamados "Sistemas Aumentativos y Alternativos de Comunicación" (SAAC) (más de 3.000 pictogramas creados específicamente para poder usarlos en tableros) encargados de mostrar qué acción se desea realizar para que el usuario pueda señalar la acción deseada con la mirada, puntero laser, etc.

Figura 7. Tablero SAAC. Ref (7.45)

Nota: La aplicación que se ha desarrollado en Java pretende ser un tablero de comunicación digitalizado que permite la interacción directa con electrodomésticos del domicilio, particularmente la Tv y el aparato de aire acondicionado.

- Comunicadores de voz: Con el paso de los años y los sucesivos avances se ha propiciado la mejora en el reconocimiento de voz para la consiguiente integración en dispositivos tecnológicos. Un ejemplo de ello son las tabletas de escritura asistida. Su funcionamiento se basa en el movimiento de los ojos y el parpadeo. Con ello, el usuario puede ir seleccionando con la mirada letras,

símbolos, etc. y seleccionándolas con el parpadeo. Un ejemplo sencillo de ello es la tableta MegaBEE[5]

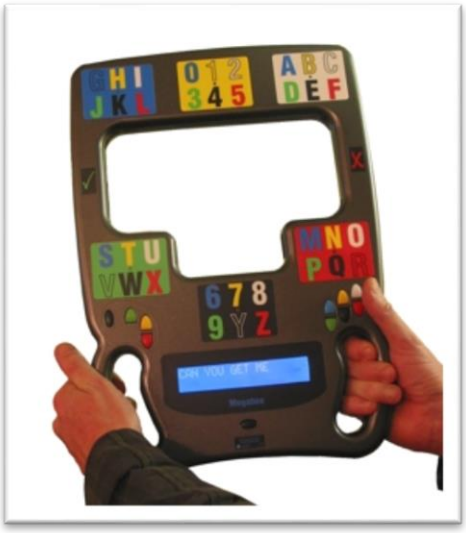


Figura 8. MegaBEE. Ref (7.46)

- Netbook, Tablet, SmartPhone: Para pacientes que poseen movilidad, el uso de cualquier aplicación instalada sobre estos dispositivos puede ayudar con la comunicación y facilitar que la persona se desenvuelva por el entorno inmediato. Además, las actuales Tablets y Smartphones permiten adaptarse a las personas con discapacidad aumentando el tamaño de letra, permitiéndose que manejen ciertas funcionalidades por voz, etc.

[5] MegaBee™ es una tableta de escritura asistida fácil de usar creada para la comunicación frecuente con personas que sufren Daño cerebral

Figura 9. SmartPhones y Tablets. Ref (7.47)

- Control del ordenador con la mirada: Hay en la actualidad muchos dispositivos e incluso móviles con cámaras y el software adecuado que permiten interaccionar con la mirada. Un ejemplo de este tipo de tecnología enfocada para personas con un grado de discapacidad alta es Irisbond ©. Este dispositivo se conecta al ordenador y permite el control del mismo con la mirada. Además, es capaz de adaptarse a Tablet, es sencillo de usar y muy intuitivo.

Figura 10. IrisBond. Ref (7.48)

Ese tipo de dispositivos son los que pueden ser utilizados junto con la aplicación desarrollada en Java que se ha desarrollado.

3.7 Otros ejemplos de tecnología asistida para personas con discapacidad motora

Aunque en la sección anterior se mostraron los dispositivos tecnológicos que permiten ayudar a las personas con discapacidad motora existen otros, no tan conocidos, que se muestran a continuación:

- Head Wand: Un ejemplo básico para ayudar a las personas con discapacidad motora avanzada es Head Wand, un casco adaptado con un puntero ajustable que permite usar una computadora acondicionada como vemos en la siguiente imagen:

Figura 11. Head Wand. Ref (7.49)

- Sip/ Puff Switch: Otro ejemplo de ingeniería básica con una idea bastante llamativa. Un dispositivo que reconoce aspiraciones y movimientos de lengua traduciendo a instrucciones simples de on/off. Esto se puede integrar en ordenadores, sillas de ruedas, etc…

Figura 12. Sip/ Puff Switch. Ref (7.3)

- Chair/Bed Ocuppancy Sensor: Un sensor muy útil que ayuda a la detección de acciones fuera de lo común de discapacitados en silla de ruedas o que necesitan estar en la cama. Se trata de una almohadilla que, dependiendo de la presión, transmite una señal vía wifi o bluetooth para detectar si la persona se ha caído de la silla de ruedas o de la cama y avisar para su auxilio.

Figura 13. Chair/Bed Ocuppancy Sensor. Ref (7.3)

- Lifeware Integra: un software para que el usuario pueda controlar el ordenador mediante movimientos de cabeza y expresiones faciales, gracias a una red de electrodos y giróscopos colocados en la cabeza.

Figura 14. Lifeware Integra. Ref (7.50)

3.8 IoT en la discapacidad

En la actualidad se están invirtiendo grandes esfuerzos en muchos proyectos en los que se pretende que el IoT (Internet of Things) facilite esa conexión contextual a personas con discapacidad, a continuación se enumeran varios proyectos que se están intentando implantar:

- Ayuda a cruzar la calle: En los países bajos se ha desarrollado un proyecto llamado Crosswalk. Se trata de una aplicación que ayuda a los peatones a cruzar la calle. Esta aplicación se instala en el teléfono inteligente del usuario y es capaz de comunicarse con el semáforo para "pedir más tiempo a la hora de cruzar".
- Ayuda para leer los alrededores: Microsoft ha desarrollado una aplicación para invidentes llamada Seeing AI. Esta aplicación usa la cámara del móvil para indicarnos lo percibido a nuestro alrededor de forma hablada e incluso alertarnos si nos dirigimos hacia alguna calle cortada. Una aplicación que proporciona asistencia a la persona invidente para que pueda desenvolverse mejor en el entorno. Además, también detecta expresiones en el rostro de las personas e informa al usuario.

- Personas con movilidad reducida: Son muchos los proyectos que se están realizando sobre casas inteligentes. Altavoces, refrigeradores, termostatos y demás aparatos electrónicos conectados a un ordenador o un dispositivo móvil ayudan a una persona con movilidad reducida para interactuar con el domicilio y desenvolverse mejor con él.
- Dispositivos werables que traducen texto, correos electrónicos, etc. al braille.
- Proyecto Smart Assist: Otro caso de IoT para la discapacidad es el proyecto Smart Assist. Con cerca de un millón de euros de presupuesto financiado por el Ministerio de Economía y Competitividad, varias empresas hacen uso de IoT y el análisis de datos masivos (Big Data Analytics) con el fin de ayudar a las personas con discapacidad prestando diferentes servicios. Estos servicios son: tele asistencia domiciliaria, tele asistencia móvil y video atención. Este proyecto contempla la integración de la solución con plataformas IoT como:

  FIWARE: Puede definirse como una plataforma de middleware abierta sobre la que desarrollar nuevas aplicaciones y servicios web.

El proyecto finaliza con el piloto de 25 usuarios donde al final se evaluarán todos sus resultados.

3.9 Estudio de la problemática a solucionar

El problema principal de las personas que padecen algunos tipos de discapacidad, como puede ser la motora, se circunscribe principalmente en el propio domicilio de la persona. Y es que, aunque en los últimos años, se ha experimentado un gran desarrollo en el mundo de las comunicaciones, especialmente la mejora de las tecnologías inalámbricas y el abaratamiento de sus dispositivos de conexión, no han sido suficiente para una integración efectiva con los elementos habituales existentes en los hogares.

Estrictamente relacionado con las comunicaciones se encuentra el mundo de la robotización del domicilio, denominado ¨Domótica¨. En la vida de personas con discapacidad, encontrarse en un domicilio domotizado se está empezando a convertir en una necesidad más que en un lujo, debido a la cantidad de dispositivos que incorporan sistemas de interacción digitales. Esto les permite a los miembros de este colectivo tomar el control del entorno y sentirse independientes dentro de su propio

domicilio, sin necesidad de contar siempre con la ayuda de terceras personas de las que frecuentemente sufren una fuerte dependencia.

Es evidente que las nuevas tecnologías ayudan a la autonomía de personas con discapacidad y permiten la integración de éstas en los ámbitos social, educativo y laboral.

La domótica ha permitido a la persona discapacitada tomar el control de los aparatos electrónicos ya sea mediante voz, pantallas o mandos a distancia. Además, agregan seguridad a esa persona, puesto que permiten integrar servicios de teleasistencia para cubrir la eventualidad de sufrir algún accidente en el domicilio.

Aunque se ha avanzado mucho en estos aspectos, su uso aún es muy limitado, tanto por motivos económicos como por motivos de adaptabilidad. Un ejemplo que podemos encontrar en la problemática de adaptabilidad es el uso de dispositivos que permiten comunicarse con la televisión a través de infrarrojos, pero no permiten la comunicación con los principales gestores de contenidos, como Movistar +. Si bien este aspecto puede parecer una cuestión baladí, se debe destacar que en pacientes con una discapacidad severa, la forma de disfrutar el tiempo de ocio se convierte en una cuestión crucial para la salud psíquica y no debe ser minusvalorada. Además, la falta de interoperabilidad entre dispositivos de distintas compañías impide un control completo sobre cierto tipo de elementos del domicilio que interconectan con otros.

Este proyecto surge de las dificultades observadas en personas con esclerosis múltiple en estado avanzado, un ejemplo de discapacidad motora severa, cuando su vida se circunscribe a la movilidad que le permite la silla de ruedas, motorizada en los casos más graves, pero que conservan el resto de capacidades intelectuales intactas. La necesidad de interactuar con el entorno se vuelve perentoria para las actividades más básicas, como leer, escuchar música, manipular persianas y toldos o modificar la temperatura de la estancia. En definitiva, actividades inmediatas de la vida cotidiana.

A pesar de las noticias que fluyen en los medios sobre avances increíbles en las tecnologías, sistemas que reconocen órdenes y ejecutan comandos de la forma más sencilla, la realidad es que las soluciones completas rara vez se implementan fuera del laboratorio. Y es que hay que ser consciente que sólo ocasionalmente se diseña una vivienda y el entorno de la ciudad donde se habita para acoger a una persona discapacitada, lo que lleva a soluciones fragmentarias o aisladas, que inhiben un

desarrollo pleno de movimientos y acciones. Téngase en cuenta que el nivel de accesibilidad lo marca el punto menos accesible, esto es, se puede disponer de una vivienda totalmente adaptada, pero un simple escalón en la entrada, nos limitará totalmente los movimientos. Y ocurre que cuando se analizan los procesos cotidianos de cualquier persona, siempre se encontrará el punto en el que una persona discapacitada se topará y se les recordará sus limitaciones.

Este proyecto pretende adaptar tecnologías existentes y extendidas en el mundo de la discapacidad para articular una capa más, no definitiva, pero significativa en la operatividad con los elementos del hogar: TV, aire acondicionado, timbre, internet, etc...

3.10 Situación de Partida

Partimos de una situación real, esto es, un usuario con una discapacidad severa avanzada que ha ido incorporando elementos de ayuda a medida que han agravándose sus limitaciones. Así dispone de una silla motorizada que mueve con un sensor en la barbilla y un ordenador portátil, con un sistema de interacción por medio del iris conocido como Irisbond para realizar operaciones básicas con el ordenador. Si bien la lectura en el ordenador le resulta cómoda, está circunscrita al puesto y se limita al ámbito informático: no dispone de interfaz de interactuación con el resto de dispositivos y apenas puede encender el ordenador. Es por ello que vamos a realizar este sistema para que pueda servirle de ayuda.

4. MATERIAL Y MÉTODO

4.1 Solución Propuesta

En los comienzos de los trabajos se realizó un ambicioso planteamiento: una persona con esclerosis múltiple en estado avanzado había adquirido algunas soluciones tecnológicas para poder interactuar con el televisor y el aire acodiciado, puerta del garaje, etc., interactuar con el domicilio del usuario en definitiva. Estas soluciones que ha adquirido no satisfecho las expectativas de la forma esperada, principalmente por problemas de comunicación. Concretamente, necesitaba varias soluciones, una para cada elemento del hogar, hay problemas de interoperabilidad, etc.

Solución propuesta:

- Un sistema único, en el que se pueda comunicar con cualquier aparato del domicilio y adaptable a la incorporación de nuevos elementos bajo estándares de redes personales.
- Un sistema desarrollado que pueda actuar desde el ordenador y desde el móvil o Tablet.

Actualmente la persona tiene una forma de comunicarse con el ordenador mediante un seguimiento del iris, que calibra la fijación de un tiempo predeterminado en un espacio de píxeles concreto y permite abrir y cerrar iconos y ventanas. Por ello se ha establecido en un principio un software para el computador con el que, a través del movimiento ocular y aprovechando el mismo sistema fijador, pueda ir seleccionando acciones básicas como manejar el televisor y encender o apagar el aire acondicionado. La comunicación con estos elementos se realizará a través de la red (Wifi), que se conectan a un servidor que funciona como actuador. El servidor se estará ejecutando en un microcontrolador Arduino. El cliente enviará a este servidor una serie de códigos preestablecidos a través de protocolo TCP, interpretará esos códigos y transmitirá la información necesaria hacia los elementos, los cuales realizarán la acción que se desee. (Acción realizada a través de infrarrojos en nuestro caso, aunque según se vaya ampliando la solución pueden ir incorporándose nuevas formas de comunicación Ej: Relé para el encendido de una lámpara mediante una diferencia de tensión entre dos puntos).

Además, solventando el problema por el cual, el usuario deba situarse frente a la pantalla de ordenador para interactuar y que este deba estar encendido de forma permanente, se planteó la necesidad de la comunicación por voz, que permite una mayor flexibilidad. La comunicación por voz permite una mayor gama de dispositivos portables y facilita la interacción a través de ellos con el resto de elementos de forma más ubicua. Para ello podemos usar otro cliente software, pero en lugar de que se ejecutara en el ordenador, pudiera hacerlo en dispositivos móviles.

Para ello se pensó en Android por ser la plataforma más abierta y común en el mercado. De esta forma, seremos capaces de comunicar con el microcontrolador a través de TCP, pero esta vez desde el dispositivo móvil, que será quien interprete los comandos de voz. En efecto, se enviarán los mismos comandos y actuará de la misma forma sobre

el microcontrolador que en la plataforma gráfica desarrollada según e describía anteriormente.

Aquí también nos podemos apoyar, configurando previamente el móvil de la persona, en el sistema de voz de Google (Ok Google). Podemos ejecutar cualquier aplicación (como por ejemplo, la que desarrollemos en este proyecto) con la voz y ser capaces de interactuar con ella sin necesidad de usar las manos en ningún momento.

Cuando se pensó en el diseño de la aplicación móvil se tuvo mucho cuidado a la hora de que esta no se bloqueara, situación verdaderamente grave para este tipo de usuarios; por ello, se diseñó de la forma más sencilla posible en cuanto a la comunicación se refiere. No se realizó ningún bucle para entablar una conversación bidireccional con la aplicación, únicamente se dice un comando por voz y este se ejecuta o no según el estado del servidor. Siempre que se dice un comando se cierra la aplicación tanto como funcione como si no para evitar problemas de bloqueo.

A continuación, se expone la arquitectura de la solución:

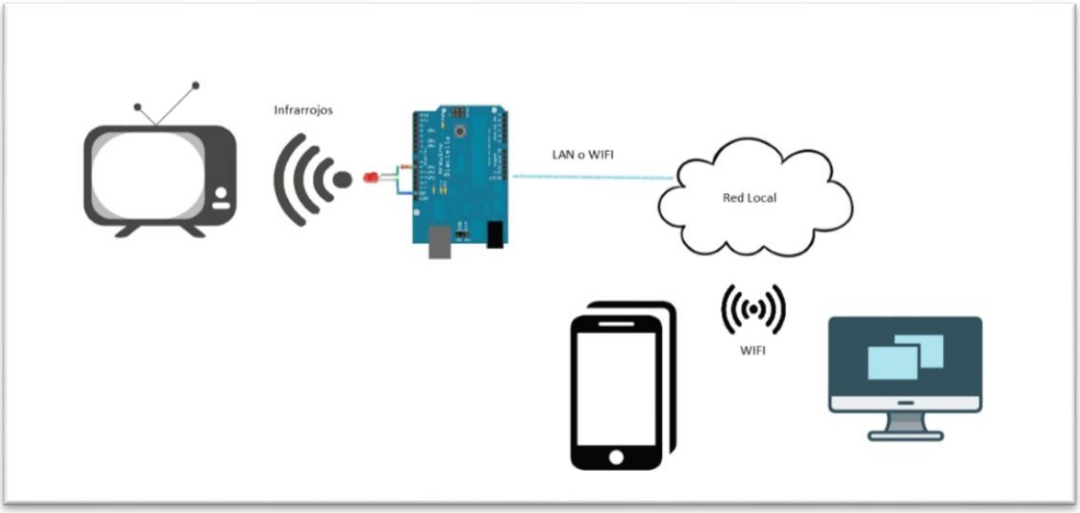


Figura 15. Solución planteada

Dada la gran carga de trabajo para un TFM, el proyecto completo se ha dividido, como se indicó en apartados anteriores en dos partes bien diferenciadas. La parte del cliente (Software para dispositivos Android y Software para ordenadores que tengan instalada la máquina virtual de Java) que se recoge en este TFM y la parte del servidor (incluye

Arduino como servidor y las diferentes comunicaciones con el entorno) desarrollos que quedan fuera del alcance de este proyecto.

4.2 Problemas a la hora del desarrollo de la solución

A la hora de abordar un problema como el planteado, se han encontrado determinados problemas, que se enumeran a continuación:

- Dificultades a la hora de interactuar con el usuario, dada su condición, lo que ha obligado a retroalimentar los desarrollos a través de fuentes secundarias e informes intermediados.
- Falta de una comunicación idónea entre los integrantes del equipo por la situación laboral de los mismos, que lleva a mantener los elementos de gestión del proyecto fuera del horario laboral, en franjas muy estrechas.
- Problemas de comunicación, debido a la distancia física entre los miembros del equipo. Uno de los miembros del equipo, quien redacta estas líneas, ha realizado el TFM desde la ciudad de Madrid y el compañero, responsable de la implementación hardware, residía en la localidad de Cáceres, aunque se ha podido soslayar este inconveniente mediante el uso de aplicativos como Skype y Google Drive.
- Falta de conocimientos en las tecnologías concretas de JavaFx y de Android, que se ha podido solventar mediante tutoriales encontrados en internet y cursos online.

4.3 Planteamiento de la metodología a usar

A la hora de realizar la solución se ha intentado llevar a cabo un método de trabajo Agile, utilizando Scrum como Framework de trabajo, por el cual, un equipo de dos personas de desarrolladores y el tutor como figura de Product Owner, iban realizando iteraciones basadas en Sprints con el fin de ir entregando cada dos semanas una solución. De esta manera, se parte de una versión muy simple del sistema y poco a poco se va incrementando su funcionalidad con el fin de, al disponer de un tiempo muy limitado, no centrar gran parte de éste en el análisis y diseño, e ir caminando en la dirección que se vaya necesitando según el tiempo, materiales y el conocimiento de cada uno de los integrantes del equipo.

Inicialmente nuestro tutor nos propuso la solución de una aplicación para ordenador y otra para móvil, según las necesidades expresadas por el usuario. Al principio desarrollamos la arquitectura de la aplicación, que se fijó en un documento escrito y se fue mejorando de forma paulatina.

Según las especificaciones iniciales del tutor, se nos pidió que la aplicación de ordenador contara con una interfaz visual simple para comunicarse con un microcontrolador y poder realizar distintas acciones para interactuar con el domicilio.

Por ello, lo primero que se intentó realizar fue un programa en Java FX (cliente), con un archivo de configuración, desde donde se obtiene la dirección del servidor (uControlador) para el encendido de un Led.

Este extremo no fue posible debido a la distancia y a motivos laborales y personales de los integrantes del equipo.

4.4 Uso de la metodología Agile en el proyecto

Una de las definiciones que encontramos al buscar Agile es la siguiente: 'Agile' es un conjunto de metodologías para el desarrollo de proyectos que precisan de rapidez y flexibilidad para adaptarse a condiciones cambiantes del sector o mercado, aprovechando dichos cambios para proporcionar ventaja competitiva. Es decir, el proyecto se "trocea" en pequeñas partes que tienen que completarse y entregarse en pocas semanas.

Anteriormente, los proyectos se hacían en cascada con sus fases de análisis, diseño, desarrollo y pruebas. Cuando se estaba realizando la fase de desarrollo o incluso en la fase de pruebas el resultado no era el esperado y se consumía mucho tiempo. Actualmente, con las metodologías agiles, se trabaja codo a codo con el cliente y en ciclos cortos, entregando funcionalidad y comprobando realmente lo que se espera del producto a entregar y adaptándose a los cambios y problemas que se van encontrando a lo largo del desarrollo.

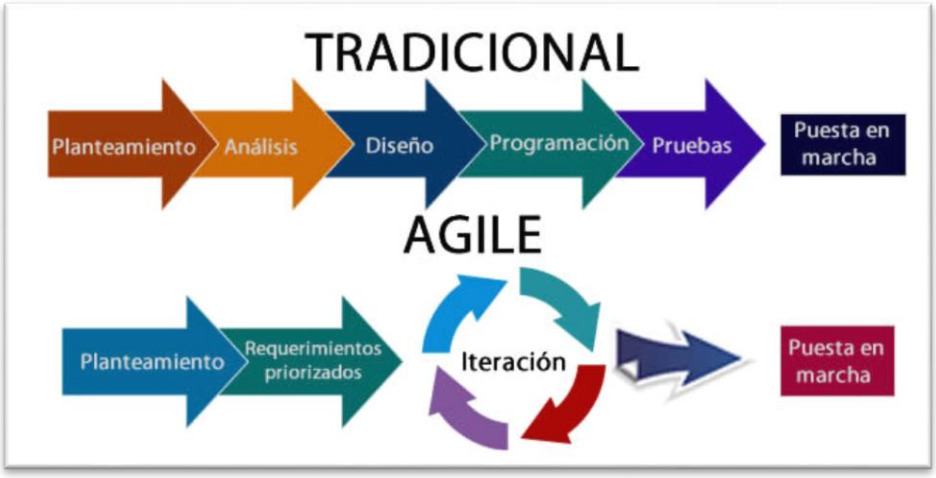


Figura 16. Waterfall vs Agile. Ref (7.51)

A la hora de realizar este proyecto se ha querido usar un marco de trabajo (framework) Agile por los motivos de rapidez y flexibilidad. Se ha intentado usar una filosofía adaptada a las necesidades de trabajo en la que el equipo de desarrollo fuera entregando funcionalidad cada cierto tiempo y realizando cambios adaptativos según la evolución del proyecto. Se creyó que sería la metodología correcta, al ser un proyecto que tenía como punto de partida una idea muy básica, y posteriormente se fue evolucionando poco a poco.

Hay diferentes metodologías Agile: XP (Extreme Programing), SCRUM, Kanban, etc.

En nuestro caso, nos hemos centrado en SCRUM, por cómo se ha definido el proyecto en un principio y la idoneidad de la misma.

SCRUM tiene su origen en 1986, en los procesos de desarrollo que se usaron de forma exitosa en Japón y EEUU para desarrollar innovaciones en cámaras de fotos Canon, automóviles Honda, Ordenadores HP, etc.

Estos productos se desarrollaron a través de requisitos muy generales, pero se necesitaban lanzar en el menor tiempo posible al mercado. La palabra SCRUM viene por la melé (palabra francesa) en Rugby o scrum (palabra en inglés) donde dos equipos de rugby se empujan de forma compacta para disputarse el balón. En esta posición, los miembros de cada equipo se unen y de forma compenetrada se adaptan en un conjunto como una unidad a la jugada. Es esto lo que se hace en un equipo donde se usa esta metodología de trabajo.

Los equipos que desarrollaron estos productos partían de requisitos muy generales, así como novedosos, y debían salir al mercado en mucho menos tiempo del que se tardó en lanzar productos anteriores. Estos equipos seguían patrones de ejecución de proyecto muy similares. En este estudio se comparaba la forma de trabajo de estos equipos altamente productivos y multidisciplinares con la colaboración entre los jugadores de Rugby y su formación de Scrum.

Realmente SCRUM es un conjunto de buenas prácticas para trabajar de forma colaborativa en un proyecto. Se ha elegido porque el trabajo se ha coordinado por medio de Skype, donde cada dos semanas aproximadamente se realizaba una reunión virtual para realizar el seguimiento del proyecto y revisar los trabajos y hacer pequeñas entregas que nuestro tutor, como Producto Owner, nos iba exigiendo. (Aunque uno de los principios del manifiesto Agile nos dice "El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara." Esto no ha sido posible debido a estar en distintas ciudades). De esta forma, obtuvimos resultados pronto y pudimos adaptarnos rápidamente a los cambios del proyecto, ya fuera por la tecnología usada, por los requisitos al principio poco definidos, por tiempo, etc.

Hemos intentado realizar ciclos de dos semanas de duración aproximadamente (Sprints) en el cual se va entregando el resultado esperado por nuestro tutor o Product Owner (que representaba al cliente y solicitaba lo que se iba necesitando).

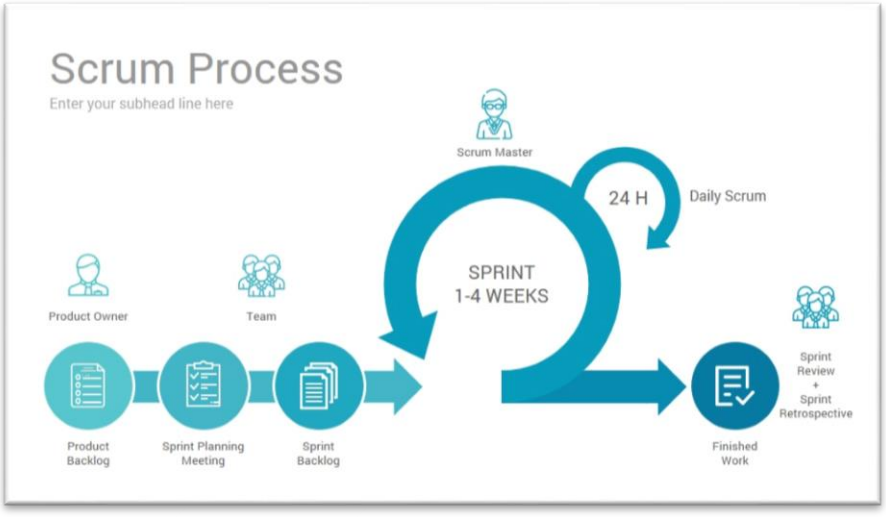


Figura 17. Scrum Process. Ref (7.52)

La dinámica es como sigue: nuestro tutor nos va proponiendo una serie de cometidos que incorporamos al backlog (lista de tareas a ejecutar y que nuestro tutor prioriza). De esta forma en cada sprint vamos sumando funcionalidad e incrementado el producto.

En cada reunión bisemanal seleccionamos los requisitos que podemos realizar en el siguiente sprint y lo planificamos de forma conjunta con el tutor. También hacemos revisiones donde se le muestra una demo de lo que se lleva desarrollado hasta ese momento para conseguir feedback y una pequeña retrospectiva donde vemos cómo podemos mejorar e indicar los obstáculos que van surgiendo así como saltarlos sin problema alguno.

También se intentó usar durante el desarrollo del proyecto un tablón virtual donde inspeccionamos cada cierto tiempo (debería ser cada día) qué se hizo la última vez, qué se ha va a hacer ahora y si hay impedimentos para cumplir esos objetivos. Para ello hemos usado Trello[6], un software de administración de proyectos de forma online donde actualizamos una To-Do-List (tareas para hacer, tareas en proceso, y tareas terminadas).

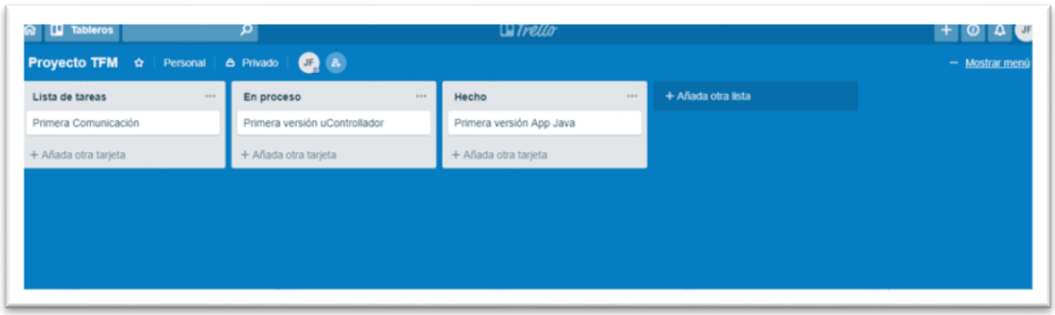


Figura 18. Tablón Trello

Incidimos pues, que por las circunstancias personales de ambas partes no se terminó usando SCRUM en la totalidad del desarrollo, ya que cada parte tuvo que ir

[6] Software de administración de proyectos : https://trello.com/

independiente una de otra y la disponibilidad de horarios, por motivos laborales, hacía imposible el uso de esta metodología.

4.5 Uso de control de versiones en la parte Software (GIT)

A la hora de escribir el código de un programa, a menudo, se quiere volver a una versión anterior del código porque nos equivocamos y sin un control de versiones no se puede. También, si varias personas están trabajando juntas de forma colaborativa es difícil interactuar sin que se sobreescriba parte del código y de lugar a errores, o es complicado unir los cambios que ha realizado cada persona.

Para ello hemos decidido usar Git, un software de control de versiones diseñado por Linus Torvalds, pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando éstas tienen un gran número de archivos de código fuente. Su propósito es llevar registro de los cambios en archivos de computadora y coordinar el trabajo que varias personas realizan sobre documentos compartidos.

La carpeta donde se guardan las copias del proyecto es el Repository (colección de todas las versiones del proyecto donde se almacenan fechas de subida, descripción de cada cambio, responsable del cambio, etc…). Para decir que una versión está finalizada se hace un Commit y se sube esa versión del lugar de trabajo local a Git.

Comandos básicos de Git que se han utilizado:

- Git init – le decimos que vamos a empezar a usar git en esa carpeta de nuestro proyecto. (Crea una carpeta oculta .git para el control de versiones) estos e hace en el área de trabajo.
- Git status: nos indica que archivos tenemos en el proyecto y su estado.
- Git add <nombre archivo> para agregar a nuestra área de trabajo ese archivo.
- Git config --global user.email “nuestro mail” para decir nuestro email de usuario.
- Git config --global user.name “nuestro nombre” para configurar git con nuestro nombre.

- Git commit para crear nuestra versión del proyecto donde se le añade un comentario para decir nuestros cambios. Además, nos da a cada archivo que subamos un id.
- Git log un log de todos los cambios.
- Git checkout --<nombre archivo> con esto se pueden revertir los cambios que se hayan realizado.
- Git diff<nombre archivo> para mostrar las diferencias del archivo de la última versión commiteada y de los cambios que están en el directorio de trabajo.
- Podemos crear un archivo. gitignore y dentro escribimos nombres de carpetas o archivos que queramos ignorar.
- Git commit –m "comentario" así se puede agregar directamente el commit y un comentario.
- Git branch muestra las ramas que tenemos (desarrollo, pre, pro, las diferentes versiones del código).
- Git branch <nombre> nos crea otra rama con otro nombre.
- Git pull para traer los cambios en remoto (GitHub en nuestro caso) a local.
- Git clone <dirección del proyecto en GitHub> para clonar los cambios de GitHub al ordenador donde estemos trabajando.
- Git checkout <nombre> nos cambia de la rama principal master a la rama de ese nombre.
- Git add . Agrega todos los archivos del entorno de trabajo.

En mi caso, para que el profesor pudiera ver el código e ir viendo el trabajo realizado, he creado una cuenta en GitHub y he usado la herramienta de repositorio online para tener un control de versiones accesible para los usuarios con los que he trabajado. Así mismo el profesor ha podido ir viendo el avance y dando feefback del desarrollo.

Además, se ha decidido usar GitHub por motivos de viaje, pudiendo acceder al código desde cualquier ordenador.

A la hora de crear el repositorio en GitHub se sincronizo el proyecto con los comandos que nos proporciona GitHub y que se muestran en la Figura 19. Sincronizar con GitHub.

Figura 19. Sincronizar con GitHub

- git remote add origin <server> con este comando podemos sincronizar con el servidor de GitHub nuestros cambios.
- git push -u origin master, comando con el que conseguimos subir a GitHub (nos pide login antes) al servidor remoto.

4.6 Arquitectura usada (Modelo Vista Controlador)

La arquitectura modelo vista controlador es un patrón de diseño de software con el fin de separar la capa de datos, la capa lógica y la vista. De esta manera se mantendrán desacopladas y podremos hacer cambios fáciles y no tocar toda la aplicación. Útil, por ejemplo, en el caso que queremos usar otra tecnología para exponer la vista al usuario.

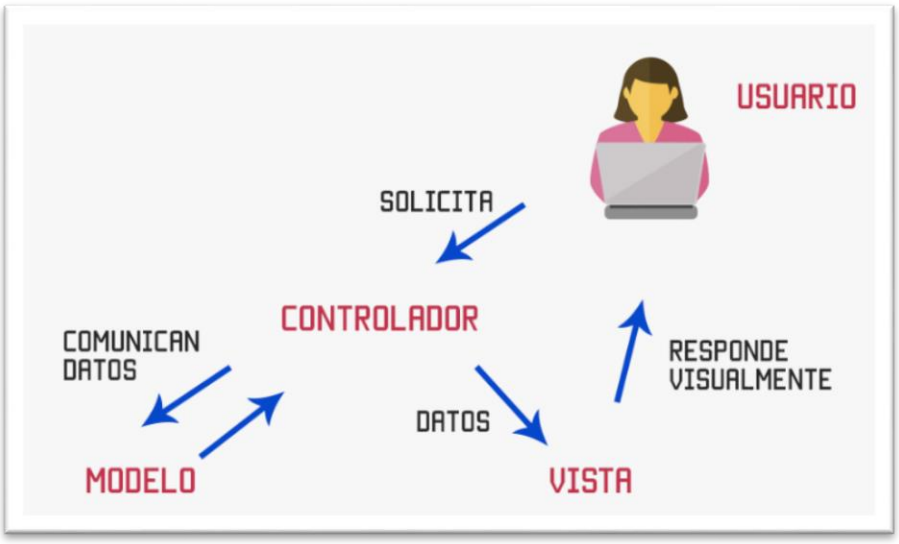


Figura 20. Arquitectura MVC. Ref (7.22)

- El modelo se utiliza para comunicarse con los datos.
- El controlador lleva toda la lógica de comunicación, recibe las peticiones del usuario y recupera los datos del modelo, los transforma y los envía a la vista para que se muestren al usuario.
- La vista es la encargada de mostrar visualmente al usuario la información de forma elegante y amena.

4.7 JAVA FX

Java FX es una librería de java usada para aplicaciones de escritorio con interfaces gráficas. Puede utilizarse en múltiples plataformas incluyendo web, móviles y aplicaciones de escritorio.

Java FX se desarrolló para reemplazar a Swing pero proporciona más funciones que Swing. Es compatible con varios sistemas operativos como Windows, Linux y Mac OS.

Una de las mejoras de JavaFX con respecto a Swing es el archivo FXML. Este archivo XML permite establecer la interfaz gráfica del usuario escrita en un XML.

4.8 IDE o Entorno de desarrollo Integrado

We choose IntelliJ IDEA©[7] as the application development IDE.

We have decided to use this IDE because I am familiar with it. Also, I created a free account to use the Ultimate version. This IDE supports several plugins and has many suggestions and autocomplete options to write the Java code.

- Note: We need level language = 10 in the Project Structure to use JavaFX without errors.

4.9 Android Studio

Android Studio[8] es el entorno de desarrollo integrado (IDE) oficial para el desarrollo de aplicaciones para Android y se basa en IntelliJ IDEA ©.

---

[7] Web Oficial de IntelliJ IDEA©. https://www.jetbrains.com/idea/

[8] Web Oficial de Android Studio. https://developer.android.com/studio/intro/?hl=ES

Para el desarrollo de la aplicación móvil se ha utilizado este entorno de desarrollo. No se tenía ningún conocimiento previo sobre Android. Para solventar esto se ha seguido un tutorial de Youtube.

4.10 Planificación

El tiempo que se ha dedicado a este trabajo fin de master ha sido de 310 horas aproximadamente. A continuación se detallan como se han distribuido las horas:

- Reuniones iniciales y de seguimiento: 12 horas.
- Búsqueda de información para JavaFX: 20 horas.
- Curso Android: 70 horas.
- Diseño y desarrollo JavaFX: 90 horas.
- Diseño y desarrollo Android: 78 horas.
- Test JavaFX: 5 horas.
- Test Android: 5 horas.
- Escritura Memoria: 30 horas.

A continuación se muestra el diagrama de Gantt para ver la planificación seguida durante el desarrollo del mismo.

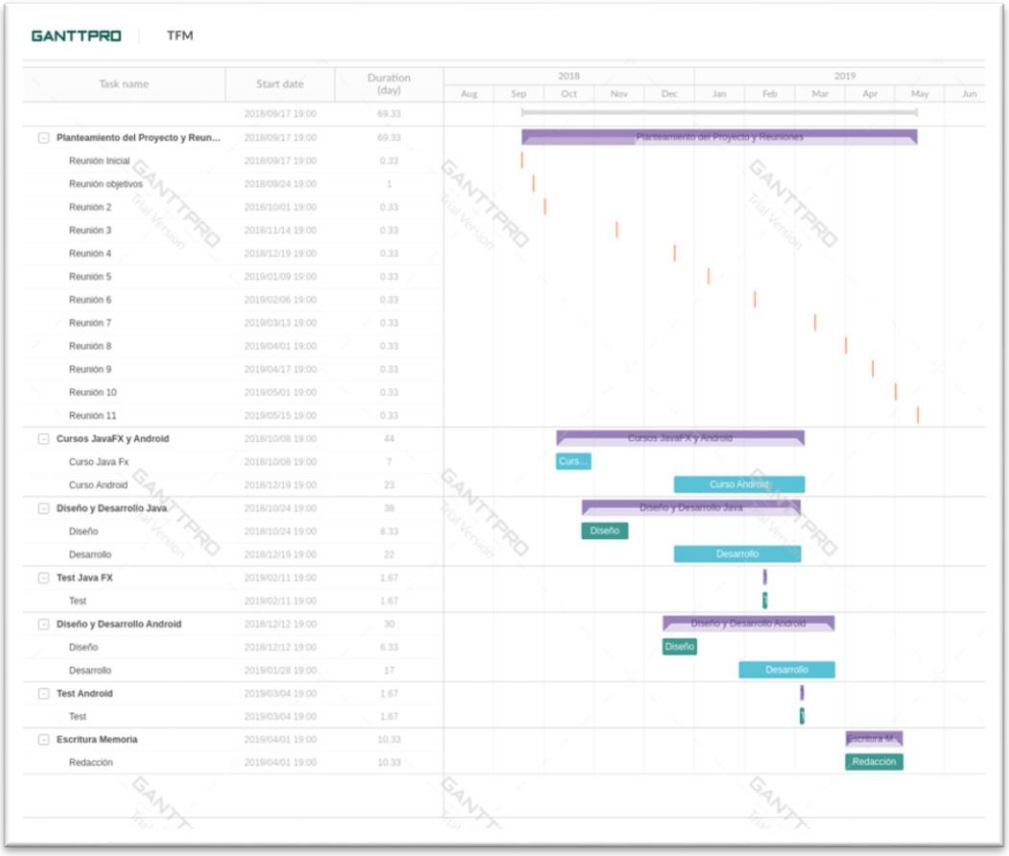


Figura 21. Diagrama de Gantt

5. RESULTADOS Y DISCUSIÓN

5.1 Introducción Aplicación de Escritorio (Java)

Aquí se va a mostrar el desarrollado que se ha llevado a cabo para hacer el software de escritorio. Esta se ha desarrollado con la tecnología JavaFx. Se ha utilizado una aproximación de la arquitectura MVC estableciendo el modelo mediante una clase singleton. Esto se ha realizado así con el fin de incluir los datos que envían del cliente al servidor y los parámetros básicos de comunicación en un archivo externo de configuración, permitiendo que sea adaptable por si se necesitara realizar cualquier cambio.

Todo el código se puede visualizar en una cuenta personal de GitHub[9]

[9] Aplicación Escritorio Java: https://github.com/jorgeantoniofloresroman/myActionApp-java

5.2 ¿Qué hace la aplicación de escritorio?

El objetivo de la aplicación de ordenador coincide con los objetivos del proyecto en sí: mostrar una interfaz al usuario donde pueda, mediante unos botones muy accesibles, comunicarse con un microcontrolador, para el control de la TV y de algún periférico más a través de un botón genérico.

Esta aplicación persigue el facilitar la vida a personas con discapacidad motora. Además, podría ser complementada por la persona afectada con un hardware que permita el tracking de los ojos e interactuar con los mismos en lugar de usar un ratón.

Se ha elegido Java para realizar esta aplicación ya que es un lenguaje ampliamente utilizado en la actualidad y lo conozco personalmente, por lo tanto, ya tengo algo de experiencia. Para la interfaz gráfica se ha elegido JavaFX. Este es un paquete de gráficos y medios que permite a los desarrolladores diseñar y crear interfaces graficas de forma sencilla.

5.3 Interface Design

I want to emphasize that the interaction with the user is the most important thing that we must pay attention to. The user of the application is a person with motor disabilities.

Our application will run on a computer and the user interacts with that through an eye tracking device.

For this reason, the design of the user interface was based on big buttons to avoid tracking errors. Furthermore, the buttons have been disabled to avoid other type of errors and we have introduced some information through label text when the connection has been established. This text has been introduced in the application with red and green colours. Colour depends of the connection result. Therefore, when the connection has been established, the colour of text is green, but if the connection has not established the colour of the text will be red.

In our application we have a configuration file where we could introduce the information that we send to microcontroller. If the file configuration is not found on our computer the label text will show us an error in black colour.

We have designed a simple user interface. We must answer the next question: What does the user need? The user needs to turn on and turn off the television. Furthermore

the user needs to change channels, turn up and turn down the volume of the television. Finally, the user needs a generic port that may be used to turn on the air conditioner.

We have painted the user interface on a sheet of paper. The drawing can be seen in the Figura 22.

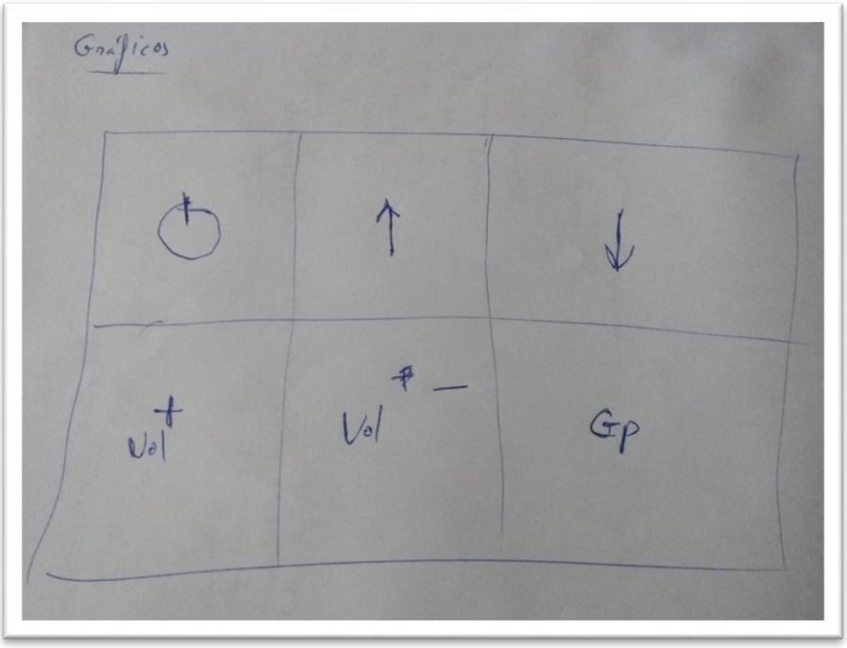


Figura 22. User Interface on sheet paper

We have created these icons in Photoshop© to be used in our application. These icons have been designed with a high resolution. The reason for this is so that we could use these icons in many different screens. Icons have a png format, 42 centimetres high and wide. Also, we choose 72 pixels per inch as a resolution.

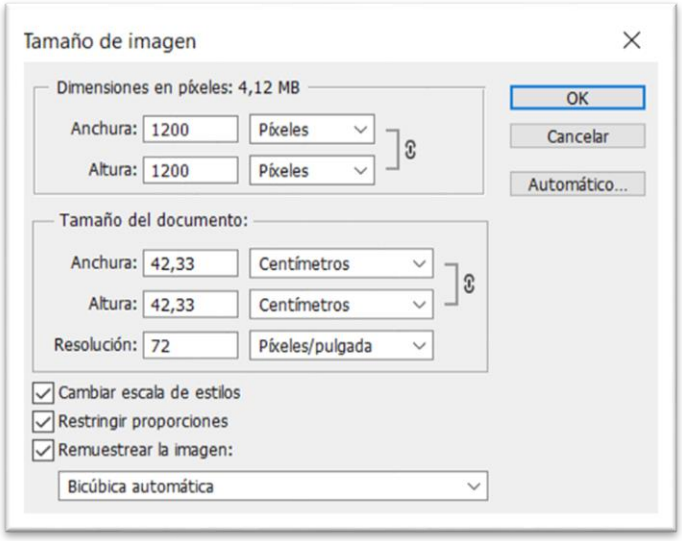


Figura 23. Icon Size

The following Figura 24 shows icon design.

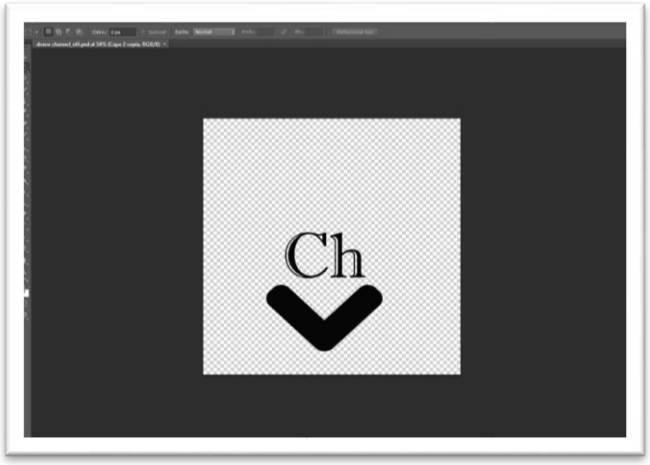


Figura 24. Down Channel Icon

We have created the same icon which shows in the next Figura 25 but in this case the icon is red. Red colour is used to represent what button was pressed.

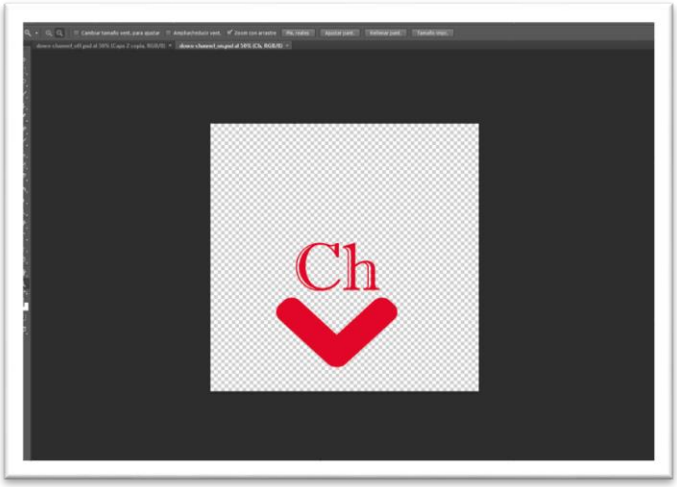


Figura 25. Red down Channel Icon

The icons that we have designed can be seen in the next image.

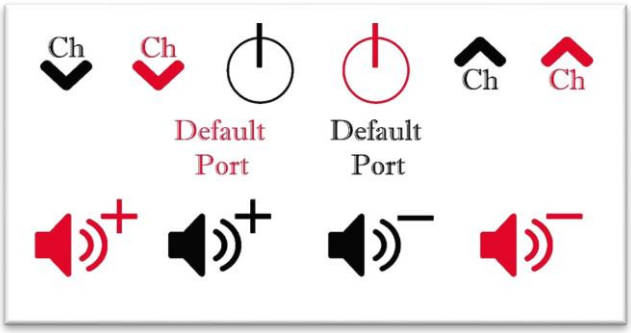


Figura 26. All Icons

5.4 Java classes and its use.

The following Figura 27 provides information about Java Classes used and the relation between them. That Figure shows us a UML diagram created with IntelliJ IDEA following a tutorial. [10]

Also, we have made other tutorial[11] where we can see the fundamentals of UML.

[10] Create UMLDiagram with IntelliJIDEA. https://www.youtube.com/watch?v=ddHXKWguxWk

[11] Diagrama de clases | | UPV. https://www.youtube.com/watch?v=JioEGJIlg88

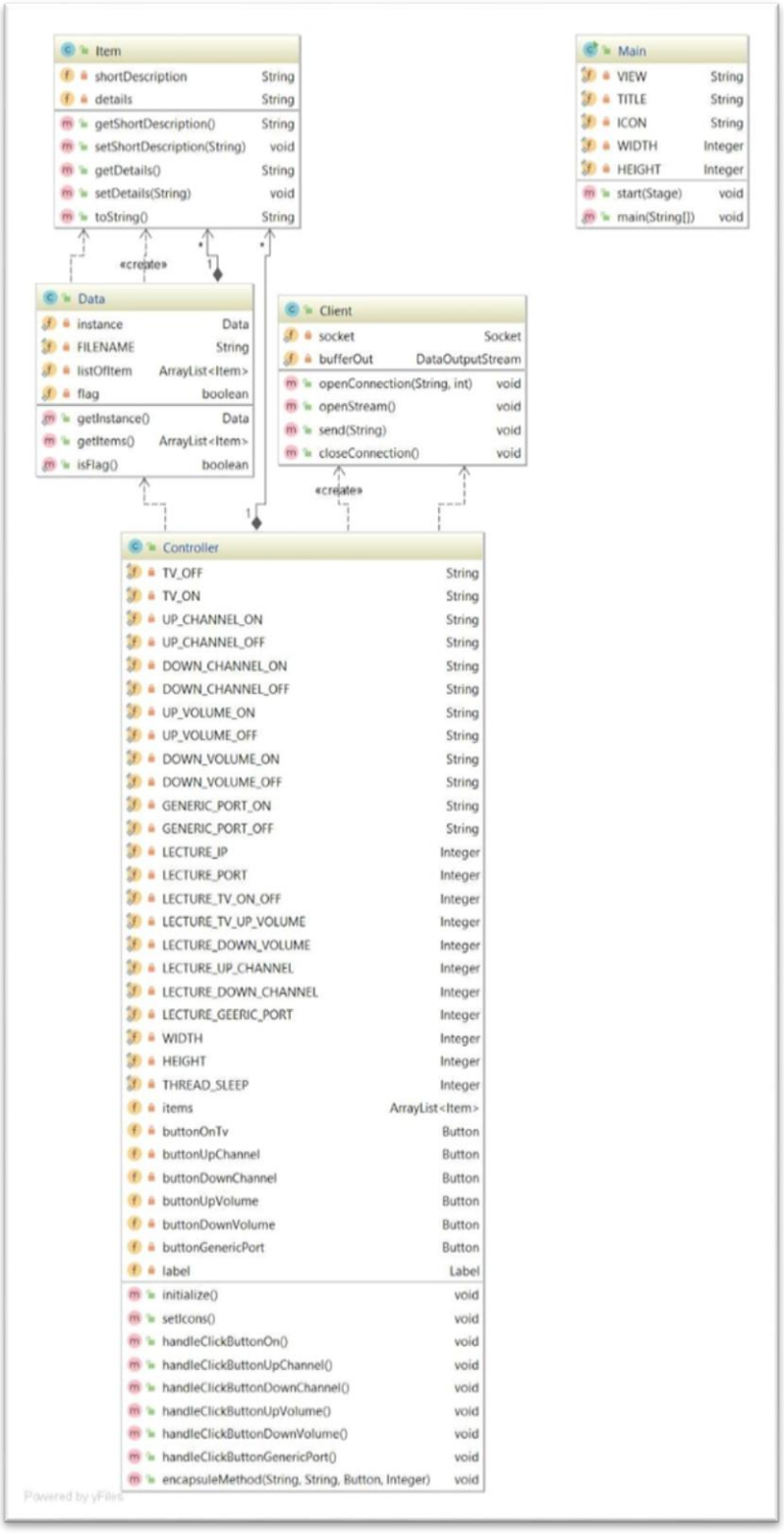


Figura 27. UML Diagram

5.5 Java Class explanation

- Item Class: This class represents a way to get the basic information that has been stored in the configuration file. This class has two attributes: shortDescription and details. Both are String and we use both to store the information that could be sent to the microcontroller. Furthermore, we recover a brief description of the port and IP direction. This IP is the direction of the microcontroller. In the next Figura 28 we can see the structure class.

```
package com.jflores.visualApp.data;


public class Item {

    private String shortDescription;
    private String details;

    //Constructor de la clase
    public Item(String shortDescription, String details) {
        this.shortDescription = shortDescription;
        this.details = details;
    }

    //Getters y Setters
    public String getShortDescription() {
        return shortDescription;
    }

    public void setShortDescription(String shortDescription) {
        this.shortDescription = shortDescription;
    }

    public String getDetails() {
        return details;
    }

    public void setDetails(String details) {
        this.details = details;
    }

    //Sobreescribe el método toString para que si imprimimos un Item devuelva la descripción
    @Override
    public String toString() {
        return shortDescription;
    }
}
```

Figura 28. Item Class

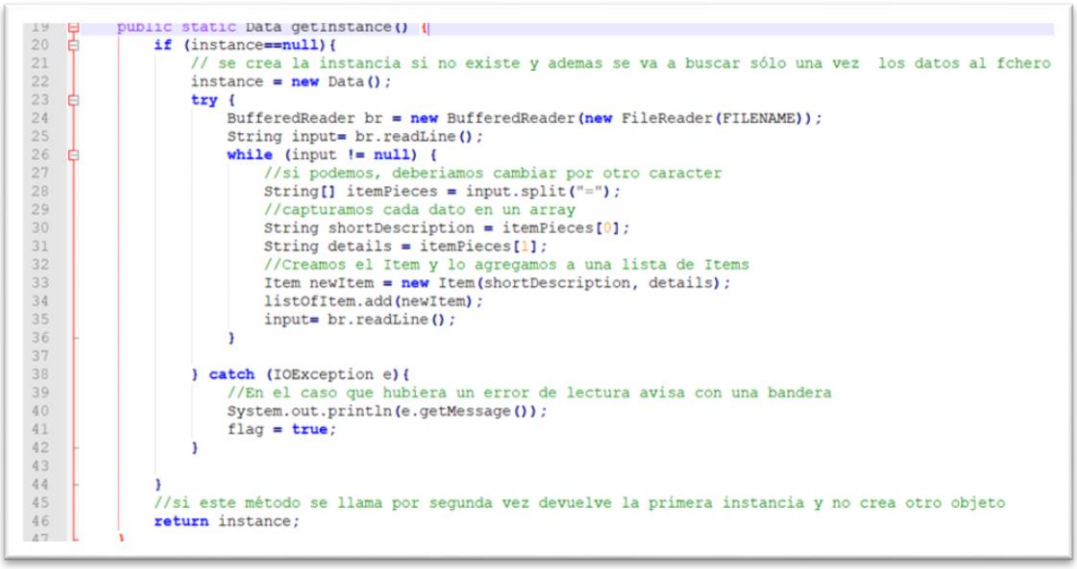
Item Class has getters and setters methods, one constructor and we override toString method for debugging purpose.

- Data Class. This class is a singleton class. In other words, we only create one instance of this class. This class is used with Item Class to get the data when the applications run. The attribute of the class are a Data instance, a file path, Item array and a Boolean flag.

```
public class Data {
    //Referencia del unico objeto que vamos a crear
    private static Data instance;
    //Nombre Fichero de configuración donde estan los datos, esos datos luego se alamacenaran en item
    private static final String FILENAME = "conf\\properties.conf";
    //Lista de Item(shortDescription,details) para guardarlos en una lista.
    private static ArrayList<Item> listOfItem = new ArrayList();

    private static boolean flag = false;
```

Figura 29. Data Class

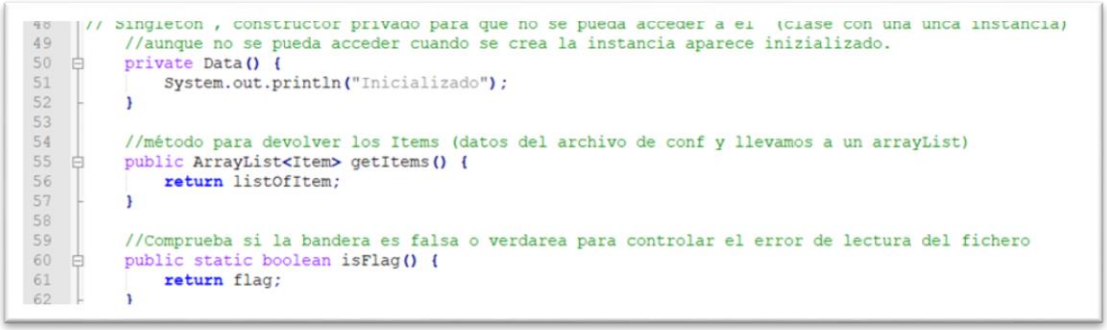
One Data class method is getinstance(), to get the file configuration helped by Buffered Reader class. When data has been read, data are stored on the Item array. If any error appears, flag will be true.



```
    public static Data getInstance() {
        if (instance==null){
            // se crea la instancia si no existe y ademas se va a buscar sólo una vez  los datos al fchero
            instance = new Data();
            try {
                BufferedReader br = new BufferedReader(new FileReader(FILENAME));
                String input= br.readLine();
                while (input != null) {
                    //si podemos, deberiamos cambiar por otro caracter
                    String[] itemPieces = input.split("=");
                    //capturamos cada dato en un array
                    String shortDescription = itemPieces[0];
                    String details = itemPieces[1];
                    //Creamos el Item y lo agregamos a una lista de Items
                    Item newItem = new Item(shortDescription, details);
                    listOfItem.add(newItem);
                    input= br.readLine();
                }

            } catch (IOException e){
                //En el caso que hubiera un error de lectura avisa con una bandera
                System.out.println(e.getMessage());
                flag = true;
            }

        }
        //si este método se llama por segunda vez devuelve la primera instancia y no crea otro objeto
        return instance;
    }
```

Figura 30. getinstance() method

The next Figura 31 shows other Data Class methods:



```
// Singleton , constructor privado para que no se pueda acceder a el  (clase con una unica instancia)
    //aunque no se pueda acceder cuando se crea la instancia aparece inizializado.
    private Data() {
        System.out.println("Inicializado");
    }

    //método para devolver los Items (datos del archivo de conf y llevamos a un arrayList)
    public ArrayList<Item> getItems() {
        return listOfItem;
    }

    //Comprueba si la bandera es falsa o verdarea para controlar el error de lectura del fichero
    public static boolean isFlag() {
        return flag;
    }
```

Figura 31. Data Class methods

We created a package named conf. In this package we can save the properties with a conf extension.

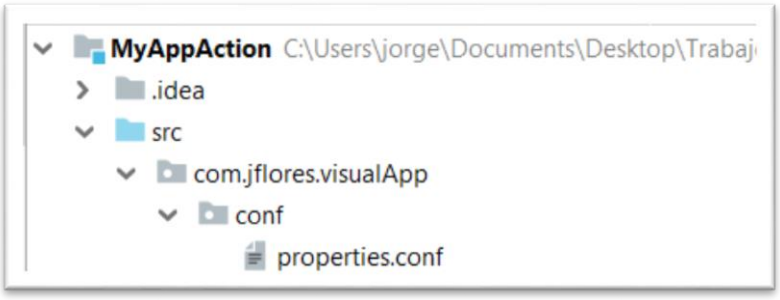


Figura 32. Configuration File

Properties file contents are shown in the next Figura 33:

```
1  Ip del uControlador primario=192.168.0.161
2  Puerto de Conexion=5050
3  1.Encender-Apagar Tv=encenderApagarTV
4  2.Subir Volumen Tv=subirVolumenTV
5  3.Bajar Volumen Tv=bajarVolumenTV
6  4.Subir Canal Tv=subirCanalTV
7  5.Bajar Canal Tv=bajarCanalTV
8  6.Puerto Generico=puertoGenerico
```

Figura 33. Properties file contents

All Images which are used in the application have been created on the img package as shown in Figura 34.

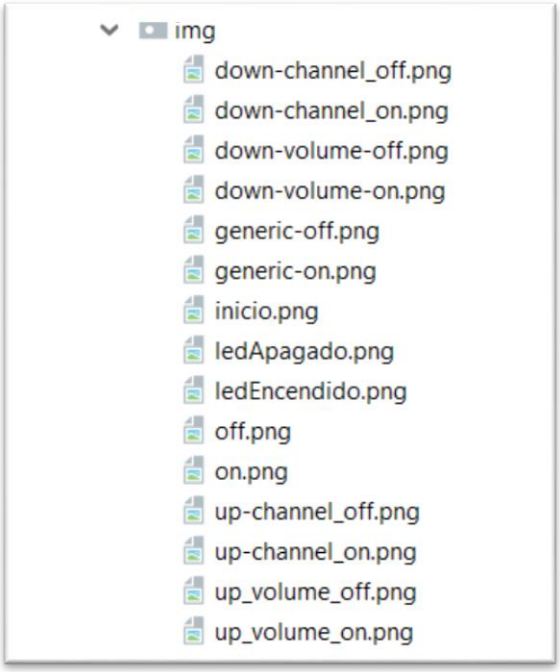


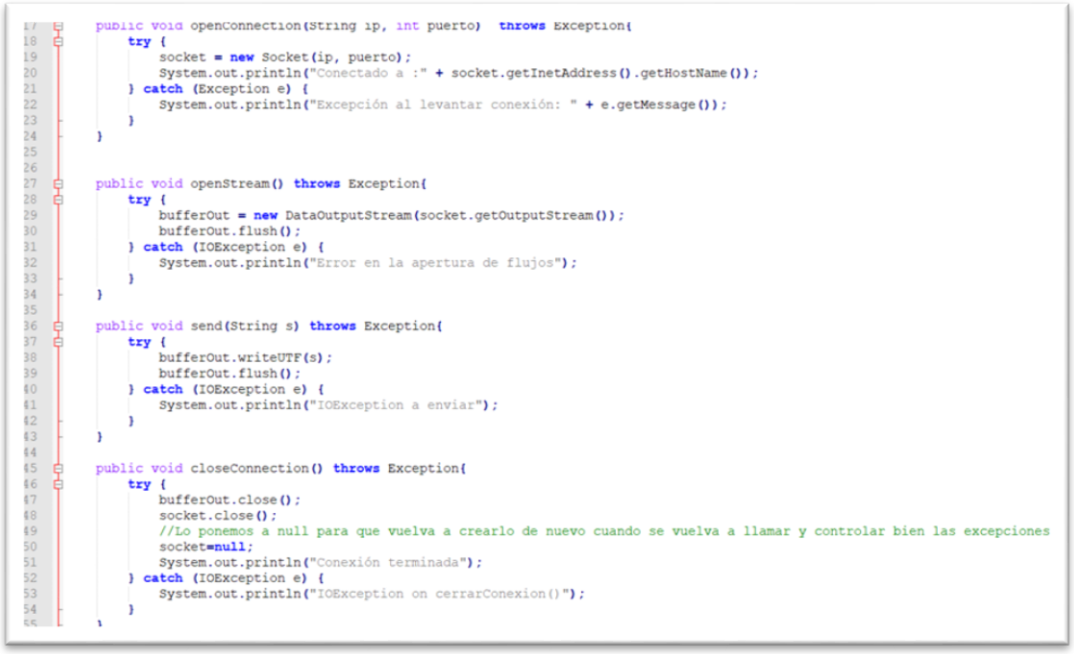
Figura 34. Application Icons

- Client Class. This class is used to create the socket and do the connection with the microcontroller. Client class have two attributes: a socket object and Data Output Stream object.

```
10  public class Client {
11
12      private static Socket socket;
13      private static DataOutputStream bufferOut = null;
```

Figura 35. Client Class

Furthermore, this class has all methods to do the connection with the microcontroller. We have one Method to open the connection, one method for opening stream a method to send data and one method for closing connection.

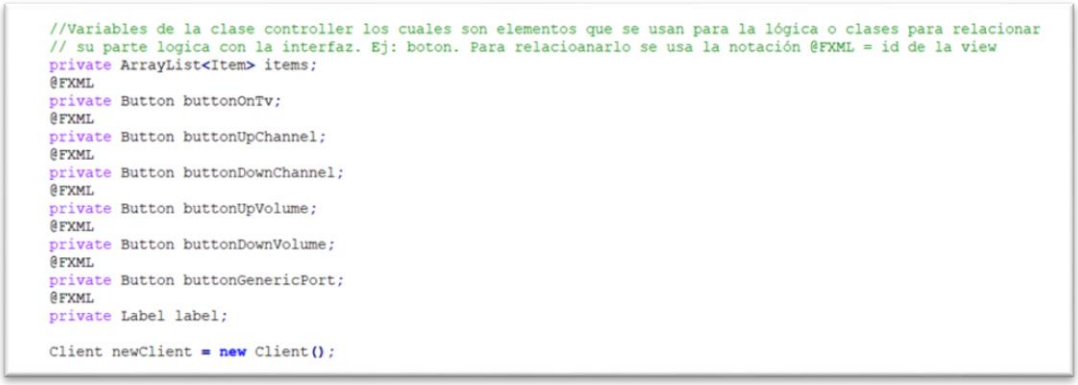


```
public void openConnection(String ip, int puerto)  throws Exception{
    try {
        socket = new Socket(ip, puerto);
        System.out.println("Conectado a :" + socket.getInetAddress().getHostName());
    } catch (Exception e) {
        System.out.println("Excepción al levantar conexión: " + e.getMessage());
    }
}


public void openStream() throws Exception{
    try {
        bufferOut = new DataOutputStream(socket.getOutputStream());
        bufferOut.flush();
    } catch (IOException e) {
        System.out.println("Error en la apertura de flujos");
    }
}

public void send(String s) throws Exception{
    try {
        bufferOut.writeUTF(s);
        bufferOut.flush();
    } catch (IOException e) {
        System.out.println("IOException a enviar");
    }
}

public void closeConnection() throws Exception{
    try {
        bufferOut.close();
        socket.close();
        //Lo ponemos a null para que vuelva a crearlo de nuevo cuando se vuelva a llamar y controlar bien las excepciones
        socket=null;
        System.out.println("Conexión terminada");
    } catch (IOException e) {
        System.out.println("IOException on cerrarConexion()");
    }
}
```

Figura 36. Client Class Method

- Controller Class has been used to communicate with the view and doing the logic application. For that, this class has an Items Array List which we have used to store all the values of the configuration file. Also, we have other attributes that can be used on the view. In these attributes we use @FXML notation. This is called dependency injection and is basically providing the objects that an object needs (its dependencies) instead of having it to construct them itself. We used that for creating buttons and labels.

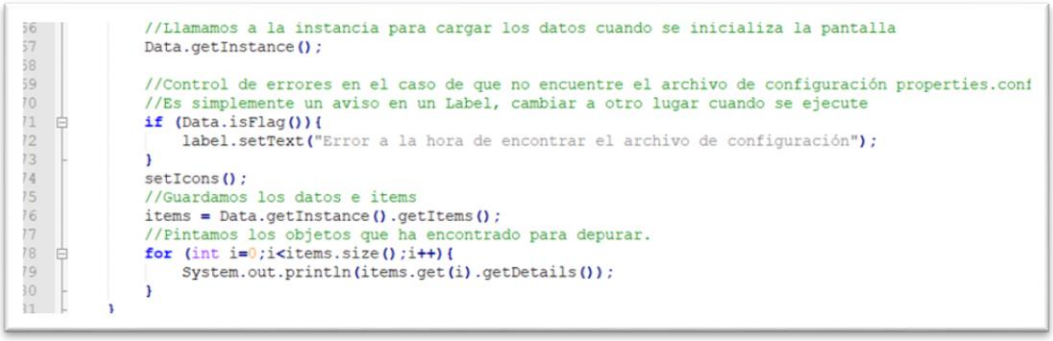
  In this class we created a Client instance to do the connection.



```
//Variables de la clase controller los cuales son elementos que se usan para la lógica o clases para relacionar
// su parte logica con la interfaz. Ej: boton. Para relacioanarlo se usa la notación @FXML = id de la view
private ArrayList<Item> items;
@FXML
private Button buttonOnTv;
@FXML
private Button buttonUpChannel;
@FXML
private Button buttonDownChannel;
@FXML
private Button buttonUpVolume;
@FXML
private Button buttonDownVolume;
@FXML
private Button buttonGenericPort;
@FXML
private Label label;

Client newClient = new Client();
```

Figura 37. Dependency injection

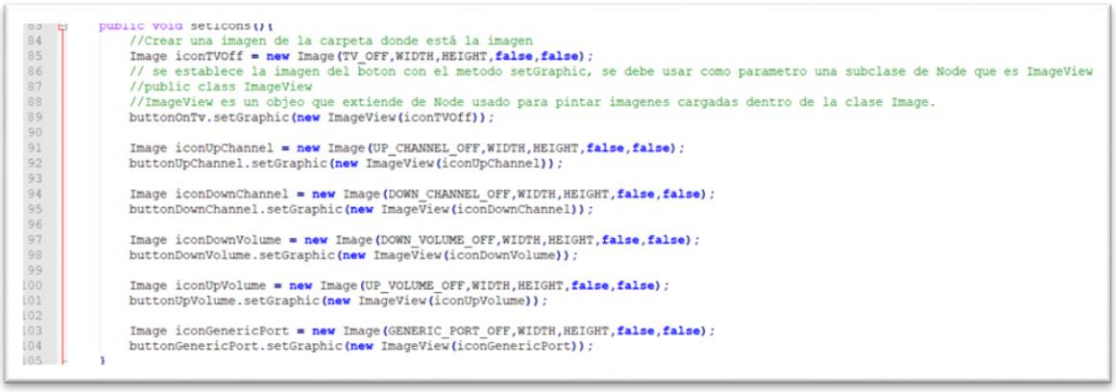
Also, in this class we have a method called initialize(). This method is used to create and initialize the objects on the view for the first time. We create the Data singleton instance and verify any error with the Data Flag.



```
        //Llamamos a la instancia para cargar los datos cuando se inicializa la pantalla
        Data.getInstance();

        //Control de errores en el caso de que no encuentre el archivo de configuración properties.conf
        //Es simplemente un aviso en un Label, cambiar a otro lugar cuando se ejecute
        if (Data.isFlag()){
            label.setText("Error a la hora de encontrar el archivo de configuración");
        }
        setIcons();
        //Guardamos los datos e items
        items = Data.getInstance().getItems();
        //Pintamos los objetos que ha encontrado para depurar.
        for (int i=0;i<items.size();i++){
            System.out.println(items.get(i).getDetails());
        }
    }
```

Figura 38. Create Instance and other actions

In addition, we create other method called setIcons() where we set icons that we use in the application. These icons are located inside the buttons.



```
    public void setIcons(){
        //Crear una imagen de la carpeta donde está la imagen
        Image iconTVOff = new Image(TV_OFF,WIDTH,HEIGHT,false,false);
        // se establece la imagen del boton con el metodo setGraphic, se debe usar como parametro una subclase de Node que es ImageView
        //public class ImageView
        //ImageView es un objeo que extiende de Node usado para pintar imagenes cargadas dentro de la clase Image.
        buttonOnTv.setGraphic(new ImageView(iconTVOff));

        Image iconUpChannel = new Image(UP_CHANNEL_OFF,WIDTH,HEIGHT,false,false);
        buttonUpChannel.setGraphic(new ImageView(iconUpChannel));

        Image iconDownChannel = new Image(DOWN_CHANNEL_OFF,WIDTH,HEIGHT,false,false);
        buttonDownChannel.setGraphic(new ImageView(iconDownChannel));

        Image iconDownVolume = new Image(DOWN_VOLUME_OFF,WIDTH,HEIGHT,false,false);
        buttonDownVolume.setGraphic(new ImageView(iconDownVolume));

        Image iconUpVolume = new Image(UP_VOLUME_OFF,WIDTH,HEIGHT,false,false);
        buttonUpVolume.setGraphic(new ImageView(iconUpVolume));

        Image iconGenericPort = new Image(GENERIC_PORT_OFF,WIDTH,HEIGHT,false,false);
        buttonGenericPort.setGraphic(new ImageView(iconGenericPort));
    }
```

Figura 39. Icon set

Also, we create other methods containing the prefix “handle” which manages the click event. We achieve that by using the event called onMouseClicked() in the view.

```
11      //Metodo para manejar el encendido de la TV cuando se hace click sobre el boton On/Off.
12      @FXML
13      public void handleClickButtonOn(){
14          //Como no es posible mandar un metodo con parametros a traves de la vista o al menos no lo encontre en la doc
15          //si que vi la forma de encapsular un metodo y pasar hay parametros de forma muy sencilla
16          //Con el fin de no repetir codigo y tenerlo todo mas organizado
17          encapsuleMethod(
18                  TV_OFF,
19                  TV_ON,
20                  buttonOnTv,
21                  LECTURE_TV_ON_OFF
22          );
23
24
25      }
26
27      @FXML
28      public void handleClickButtonUpChannel(){
29          encapsuleMethod(
30                  UP_CHANNEL_OFF,
31                  UP_CHANNEL_ON,
32                  buttonUpChannel,
33                  LECTURE_UP_CHANNEL
34          );
35      }
```

Figura 40. Click

In this method we used other Task Class to manage the user interface. It is used when we do click on a button. Then, the icon changes and shows the same icon but in red. For a few seconds the button is disabled to avoid any errors.

```
178 public void encapsuleMethod(String imagenApagada,String imagenEncendida, Button botonPulsado, Integer datoAEnviar){
179
180     //imagenes para hacer el cambio y mostrar que hay un cambio y se esta enviando el mensaje.
181     Image iconApagado = new Image(imagenApagada,WIDTH,HEIGHT,false,false);
182     Image iconEncendido = new Image(imagenEncendida,WIDTH,HEIGHT,false,false);
183
184     //https://docs.oracle.com/javase/8/javafx/api/javafx/concurrent/Task.html
185     Task<Void> myTask = new Task<Void>() {
186         @Override
187         protected Void call() throws Exception {
188             try {
189                 Thread.sleep(THREAD_SLEEP);
190             } catch (Exception e) {}
191             return null;
192         }
193
194         @Override
195         protected void succeeded() {
196             botonPulsado.setGraphic(new ImageView(iconApagado));
197             botonPulsado.setDisable(false);
198         }
199
200         @Override
201         protected void running() {
202             botonPulsado.setGraphic(new ImageView(iconEncendido));
203             botonPulsado.setDisable(true);
204         }
205
206     };
```

Figura 41. Encapsulate Method I

```
        boolean x=false;
        @Override
        protected Void call() throws Exception {
            try { //subcestible de ser comentado
                newClient.openConnection(items.get(LECTURE_IP).getDetails(),Integer.parseInt(items.get(LECTURE_PORT).getDetails()));
                newClient.openStream();
                newClient.send(items.get(datoAEnviar).getDetails());
                newClient.closeConnection();
            }catch (Exception e){
                x=true;
                System.out.println("Falla conexión"); //capturar el throws, mirar que es eso
            }
            return null;

        }

        @Override
        protected void succeeded() {
            if(!x){
                label.setText("Orden:  " + items.get(datoAEnviar).getShortDescription() + "  enviada.");
                label.setTextFill(Color.color(0.2,1,0.1));
            }else{
                label.setText("Error al conectar con el uC");
                label.setTextFill(Color.color(1,0,0));
            }

        }
    };
```

Figura 42. Encapsulate Method II

```
239     new Thread(myTask).start();
240     new Thread(myTask2).start();
```

Figura 43. Threads

- View.fxml: This is the fxml view. In this view we have created all graphical components. We have used a view called GridPane, buttons and labels. The id buttons must match with name of the Controller Class. Then, we have used onMouseClicked() to manage the click event.

```
<?import javafx.geometry.Insets?>
<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.layout.GridPane?>

<!--Comentarios,
Archivo FXML con esa notación para la interfaz gráfica, separando la parte gráfica (View) de la parte de la lógica (Controller)
-->
<!--Usamos Grid Pane como layout de la app, modelo de rejilla-->

<GridPane alignment="TOP_LEFT" hgap="10" prefHeight="218.0" prefWidth="306.0" vgap="10" xmlns:fx="http://javafx.com/fxml/1" xmlns="http://javafx.com/javafx/8.0.121"
fx:controller="com.jflores.visualApp.Controller">

    <padding><Insets bottom="10" left="25" right="25" top="25" /></padding>
    <!--Añadimos un boton con un id para relacionarlo con el controller, al igual que el evento que le damos
    al boton (onMouse) para relazionar ese evento al metodo del controller hanldeClick-->
    <Button fx:id="buttonOnTv" onMouseClicked="#handleClickButtonOn"  GridPane.columnIndex="1" GridPane.columnSpan="2" GridPane.rowIndex="1" />
    <Button fx:id="buttonUpChannel" onMouseClicked="#handleClickButtonUpChannel"  GridPane.columnIndex="5" GridPane.columnSpan="2" GridPane.rowIndex="1" />
    <Button fx:id="buttonDownChannel" onMouseClicked="#handleClickButtonDownChannel"  GridPane.columnIndex="10" GridPane.columnSpan="2" GridPane.rowIndex="1" />
    <Button fx:id="buttonUpVolume" onMouseClicked="#handleClickButtonUpVolume"  GridPane.columnIndex="1" GridPane.columnSpan="2" GridPane.rowIndex="2" />
    <Button fx:id="buttonDownVolume" onMouseClicked="#handleClickButtonDownVolume"  GridPane.columnIndex="5" GridPane.columnSpan="2" GridPane.rowIndex="2" />
    <Button fx:id="buttonGenericPort" onMouseClicked="#handleClickButtonGenericPort"  GridPane.columnIndex="10" GridPane.columnSpan="2" GridPane.rowIndex="2" />
    <!--Usamos d emomento labels para depurar-->

    <Label text="Datos Salida: " GridPane.columnIndex="1" GridPane.columnSpan="2" GridPane.rowIndex="3">
    </Label>
    <Label fx:id="label" GridPane.columnIndex="1" GridPane.columnSpan="2" GridPane.rowIndex="4">
    </Label>

</GridPane>
```

Figura 44. Fxml View

- Main Class is used to create the main window. We established height and width of this window. In the main() method we used the launch() method to run the JavaFx Application.

```
public class Main extends Application {

    private static final String VIEW = "view.fxml";
    private static final String TITLE = "Visual Action App";
    private static final String ICON = "com/jflores/visualApp/img/inicio.png";
    private static final Integer WIDTH = 920;
    private static final Integer HEIGHT = 600;

    //sobreescribimos el método start para cargar la interzaz
    @Override
    public void start(Stage primaryStage) throws Exception{
        //Cargar el fxml de la vista con javafx
        Parent root = FXMLLoader.load(getClass().getResource(VIEW));
        //establecer el nombre de la app
        primaryStage.setTitle(TITLE);
        //Establecer icono de la interfaz y agregarlo
        primaryStage.getIcons().add(new Image(ICON));
        //Establecer el tamaño de la ventana
        primaryStage.setScene(new Scene(root, WIDTH, HEIGHT));
        //Mostrar  el escenario (ventana)
        primaryStage.show();

    }


//ejecutamos el metodo launch de Application
    public static void main(String[] args) {
        launch(args);
    }

}
```

Figura 45. Main Class

5.9 Test Server

For testing purpose we have created a server to emulate the microcontroller behaviour. This server was made in Java. We have only one class named Server which we have the following objects that are shown in the next Figura 46.

```
public class Server {


    private Socket socket;
    private ServerSocket serverSocket;
    private DataInputStream bufferDeEntrada = null;

```

Figura 46. Server Attributes

We have created several methods:

A method called levantarConnexion() where we create a ServerSocket instance.

Figura 47. Levantar Conexion Method

Other method called flujos() which tries to create new Data Input Stream.

Figura 48. Flujos Method

For receive data we use recibirDatos() method witch uses readUTF() for read data.

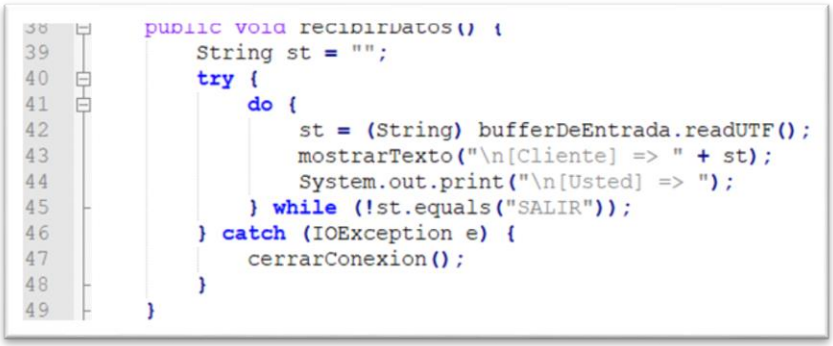


Figura 49. Recibir Datos Method

Furthermore, we have created other method called mostrarTexto() to print in the console the result using system.out.print().

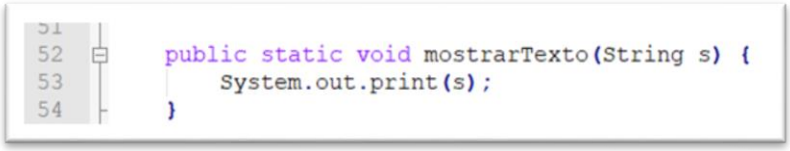


Figura 50. mostrarTexto() Method

To close connection we have created a method called cerrarConexion() which helps us to close the connection.

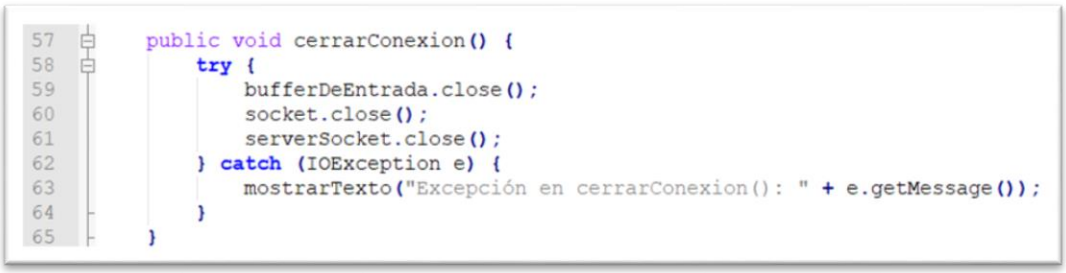


Figura 51. cerrarConexion() method

We have created ejectuarConexion() method which waits for data with a while loop.

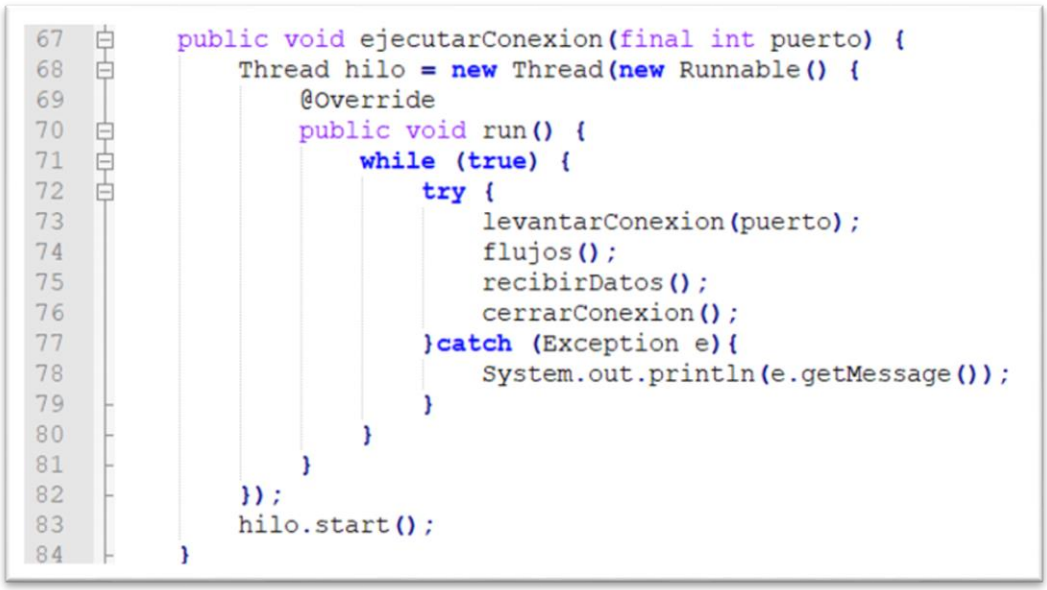


Figura 52. ejecutarConexion() method

Finally, we have created the main() method in order to create the instance of the class and run the server on port 5050.

```
86      public static void main(String[] args) throws IOException {
87          Server s = new Server();
88          String puerto = "5050";
89          s.ejecutarConexion(Integer.parseInt(puerto));
90      }
```

Figura 53. Main() Method

5.10 Test the Java Application

To test the application we opened the server and the client in two different windows as we could see in the next Figura 54.

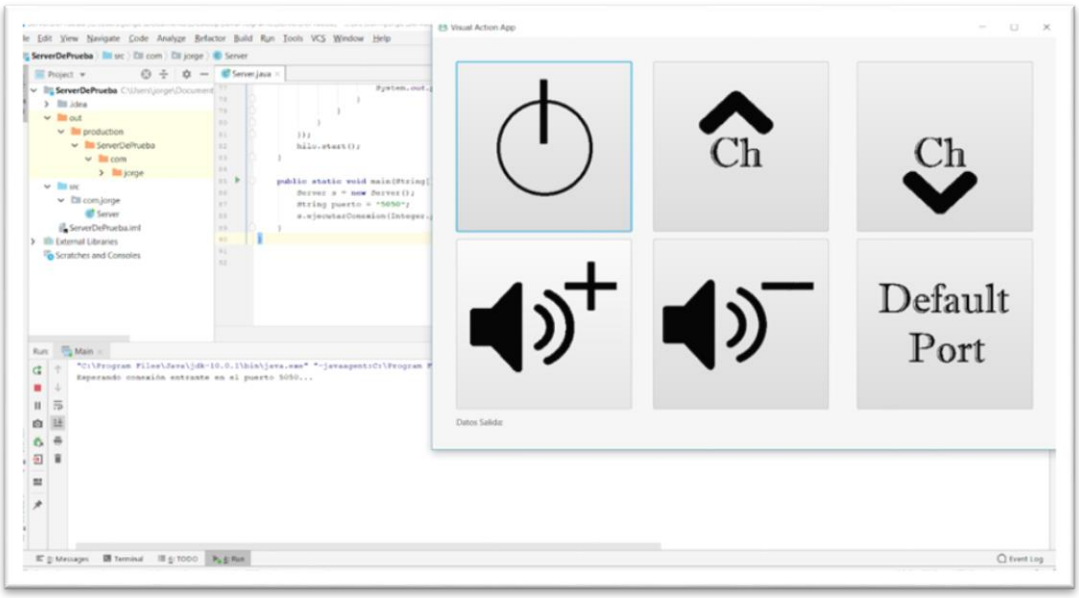


Figura 54. Test I

When we press the button of the client we could verify the information on the terminal server.

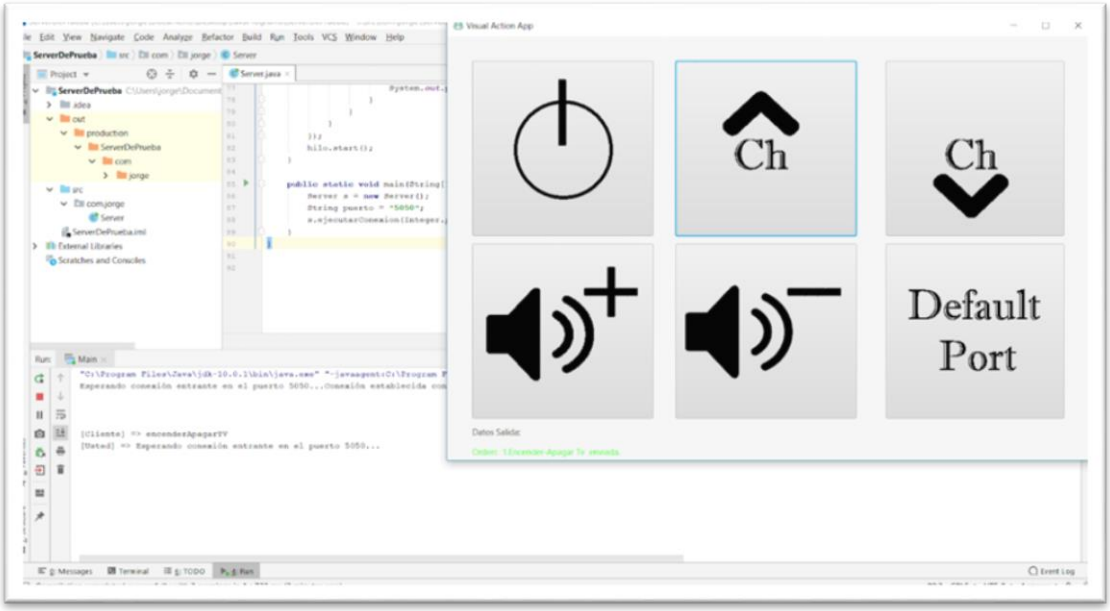


Figura 55. Test II

As we could see in the next Figura 56 the button is disabled for a while when we press the button.

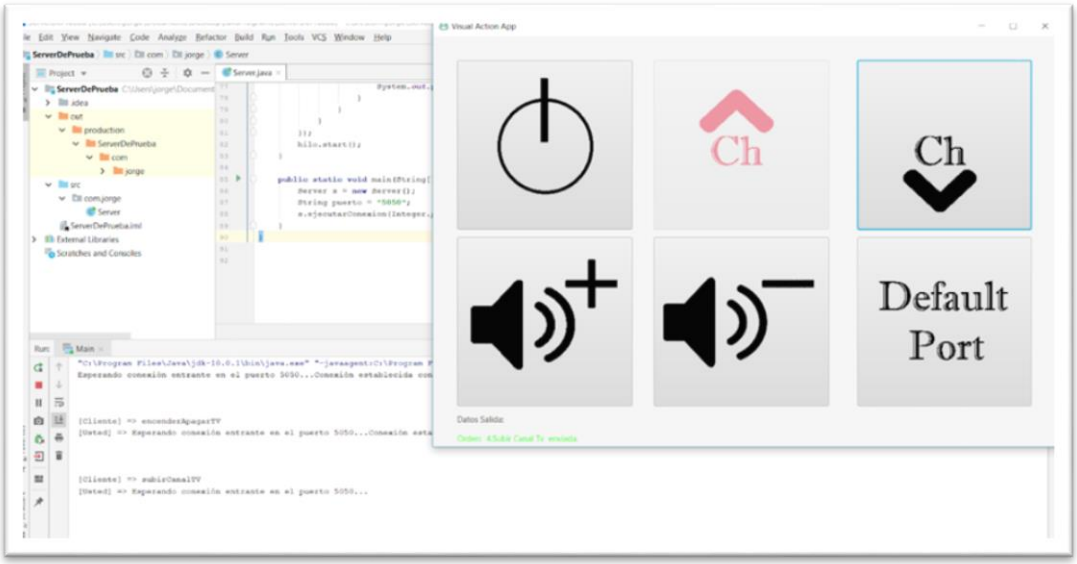


Figura 56. Test III

If we close the server we can see in the next Figura 57 that we get a red error.

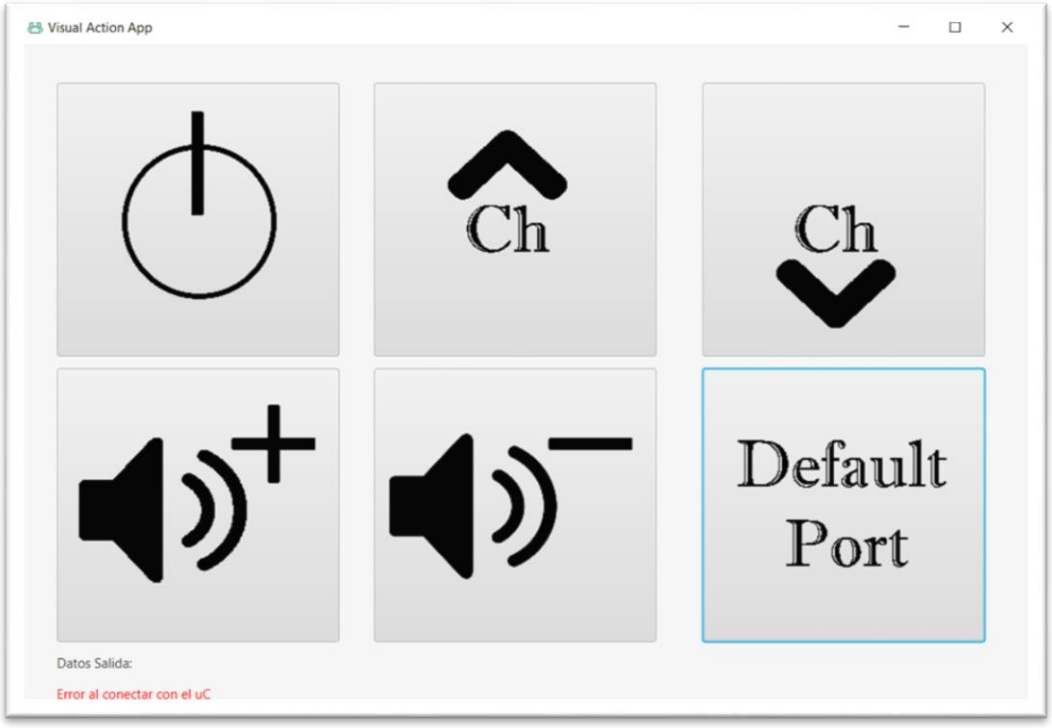


Figura 57. Test IV

5.11 Application Packaging

For the end user we need to create a jar file which runs on the Java Virtual Machine. To do the package we see several tutorials on IntelliJ IDEA © 2019.1 Help.

To do that, we go to File/Project Structure/ Artifacts

We create a new configuration with the plus symbol and we choose JAR From modules with dependencies.

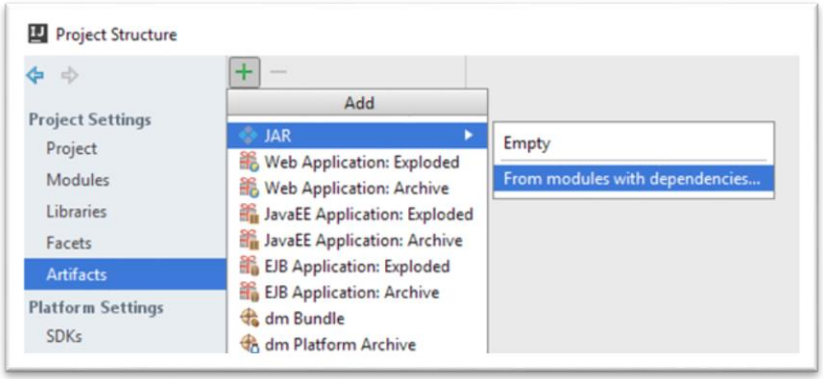


Figura 58. Create Artifacts

Then, we choose the main class as we could see in the next Figura 59.

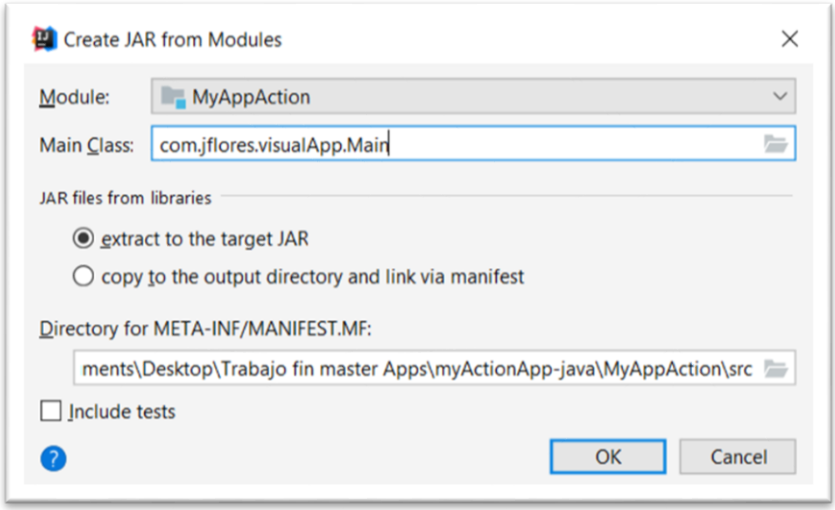


Figura 59. Choosing the main class

Next, we finish the configuration. To do that, we need to click in Ok button.

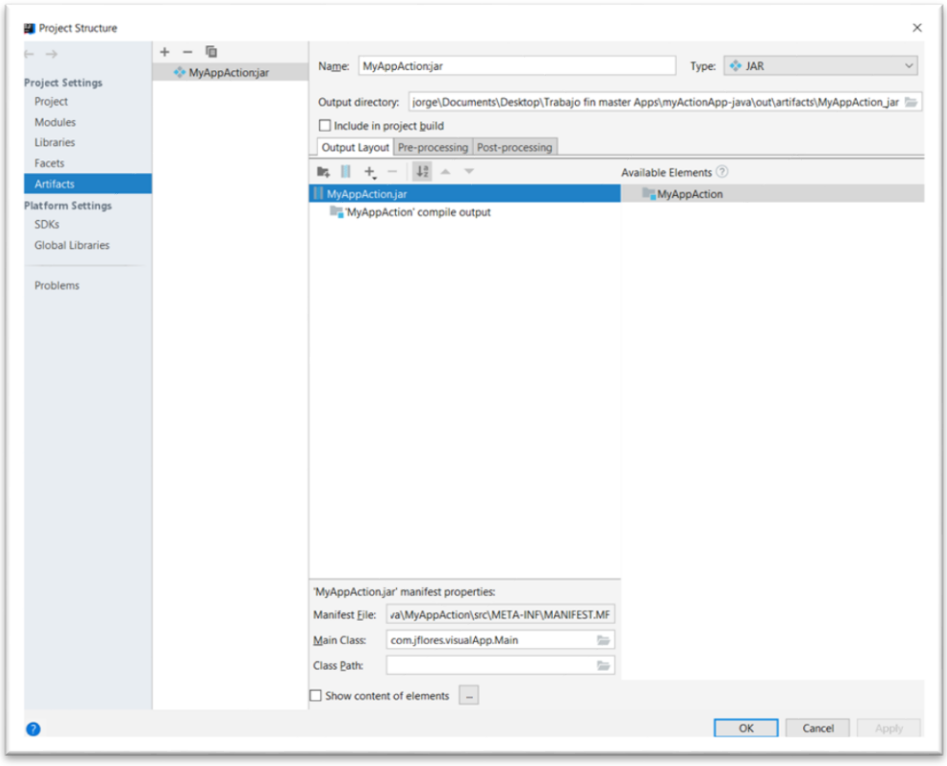


Figura 60. Finish the Configuration

Finally, we go Build, Build Artifact, Build. We found our jar file in the next path: out/artifacts/nuestro_archivo.jar

But, when we run the application we found the next error as show as below.

Figura 61. Configuration Error

This error was caused by the path of the properties.conf file. We need to change the path and use a relative path.

```
public class Data {
    //Referencia del unico objeto que vamos a crear
    private static Data instance;
    //Nombre Fichero de configuración donde estan los datos, esos datos luego se alamacenaran en item
    private static final String FILENAME = "conf\\properties.conf";
```

Figura 62. Relative Path

Now, we must create in the same level of the jar file a folder named conf and save in the properties.conf. as we could see below.

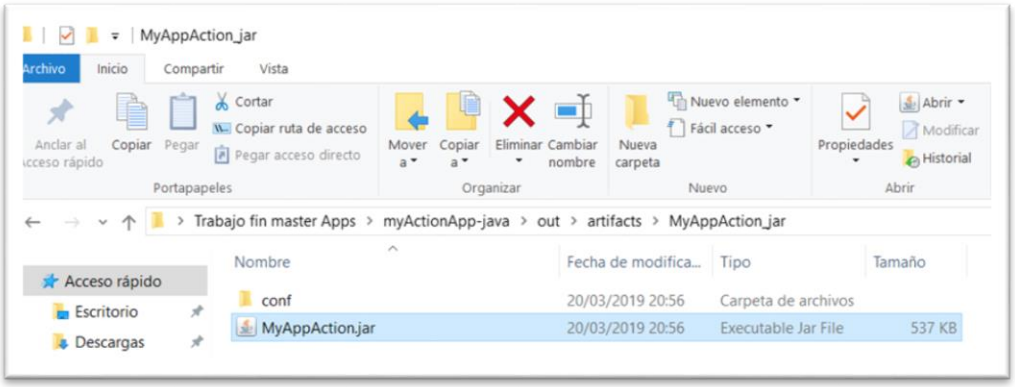


Figura 63. Structure Folder

And we could run this application on every computer that supports Java (The computer needs install JVM).

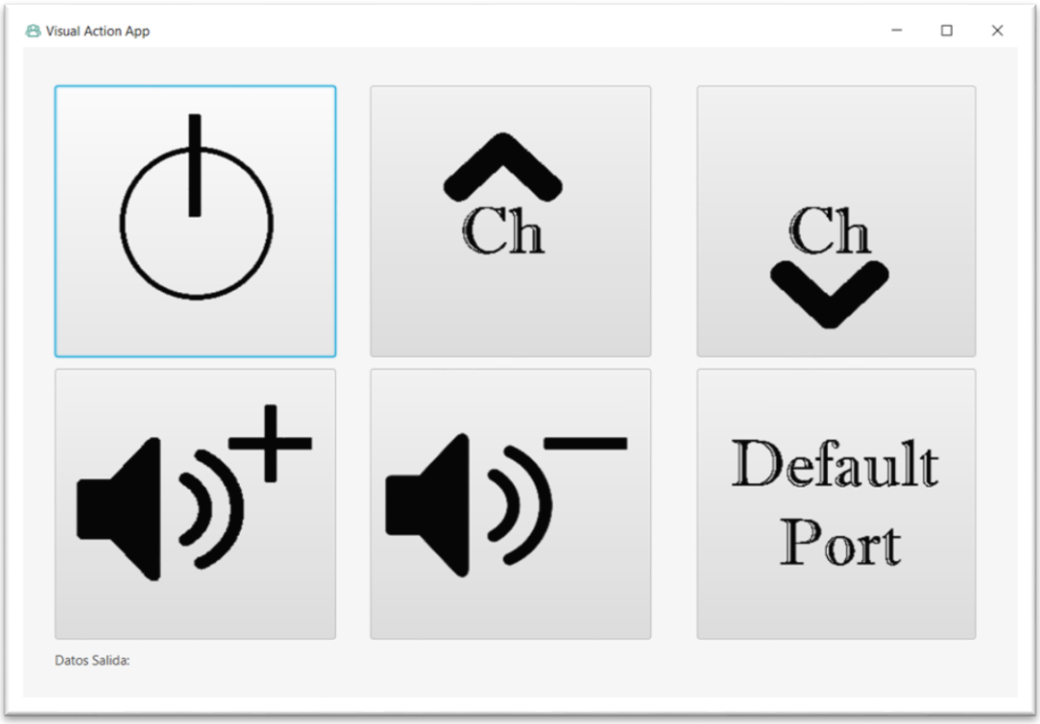


Figura 64. Application works

5.12 Android Client

Once we have verified that the Java Application works properly we will create another client application. This client will execute the same commands but using voice to perform the communication. We are going to use Android Studio.

5.13 Creating the project under Android Studio

We are going to create a new application in Android Studio.

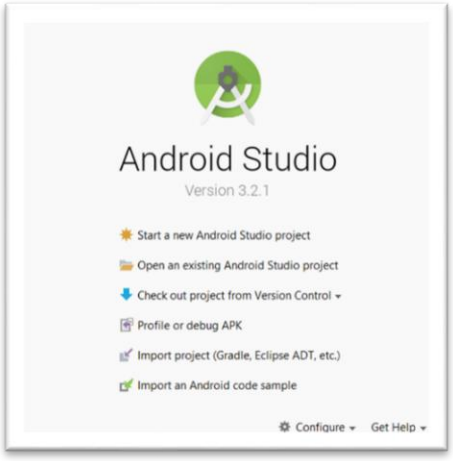


Figura 65. First Window Android Studio

The project name is Interaction.

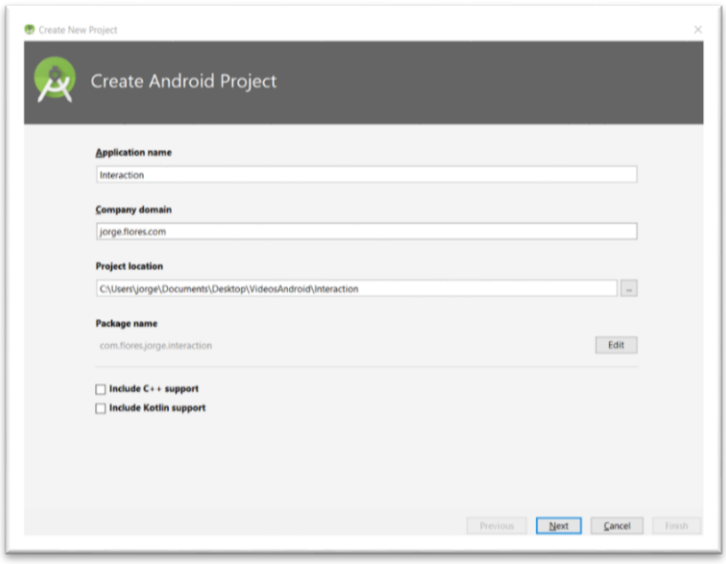


Figura 66. Name of the project

The API we're going to use is 23. This API works on approximately 62.6% of current devices as we could see below.

Figura 67. API 23

We could see in the next Figura 68 the android studio API help:

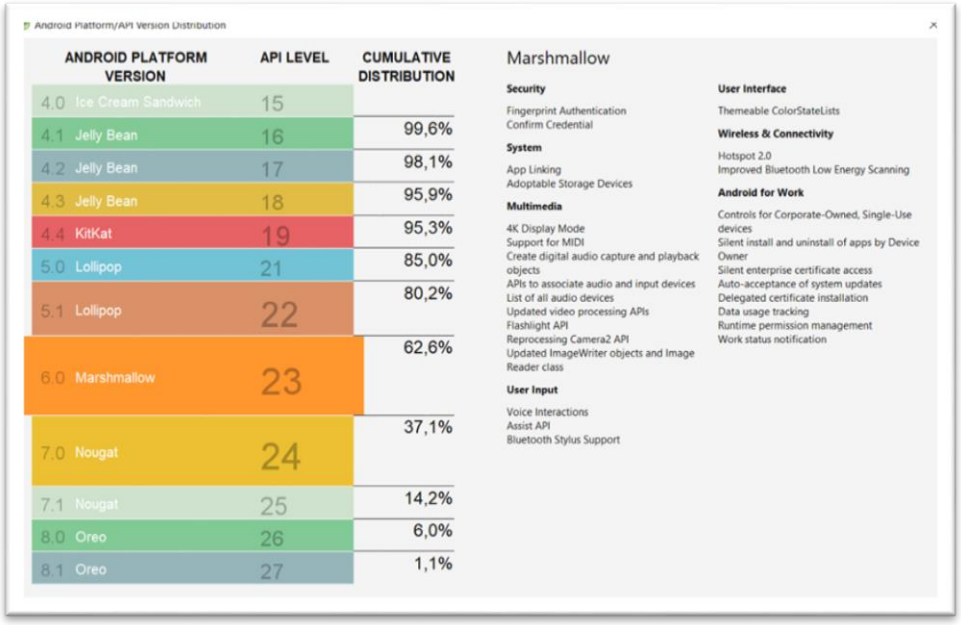


Figura 68. Android Studio API Help

Of course, we have created an application that runs on Smartphones and Tablets.

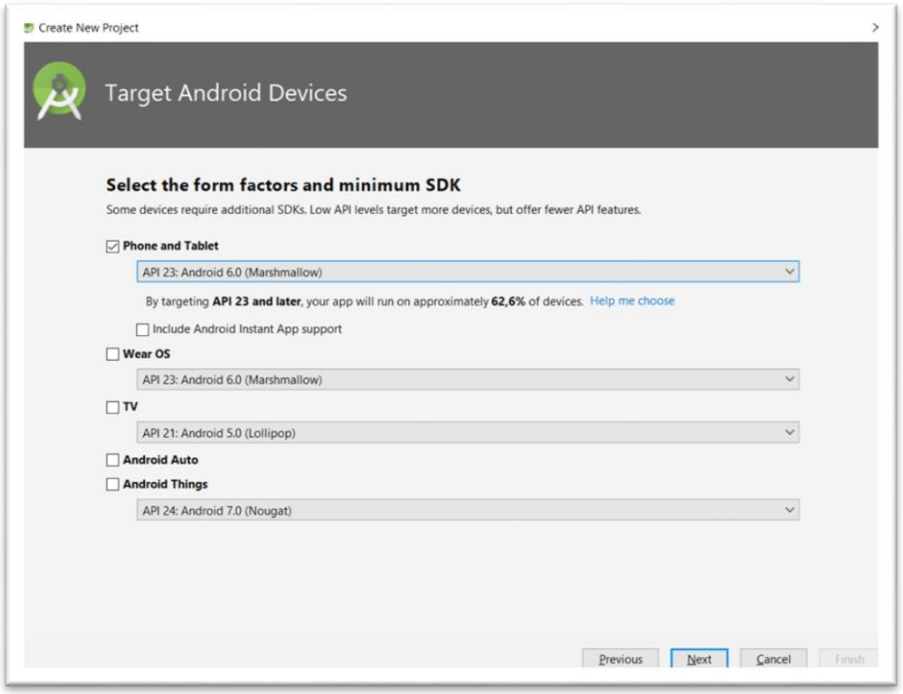


Figura 69. Select the devices

As soon as we create the project the wizard helps us to create an activity. We don't use any template.

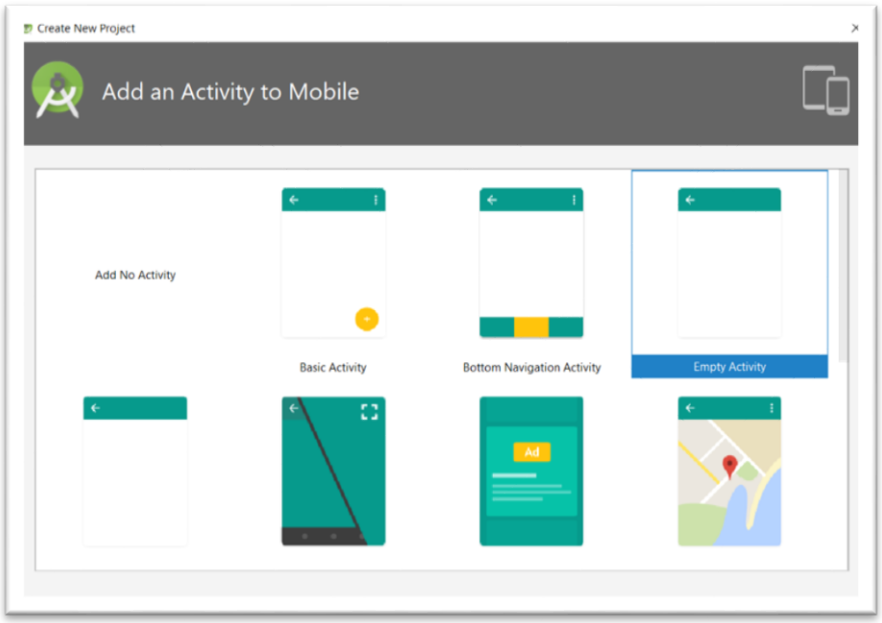


Figura 70. Activity Template

In the next section we choose the name of the activity and the name of the layout. This layout is an xml file which create the graphical components. (View)

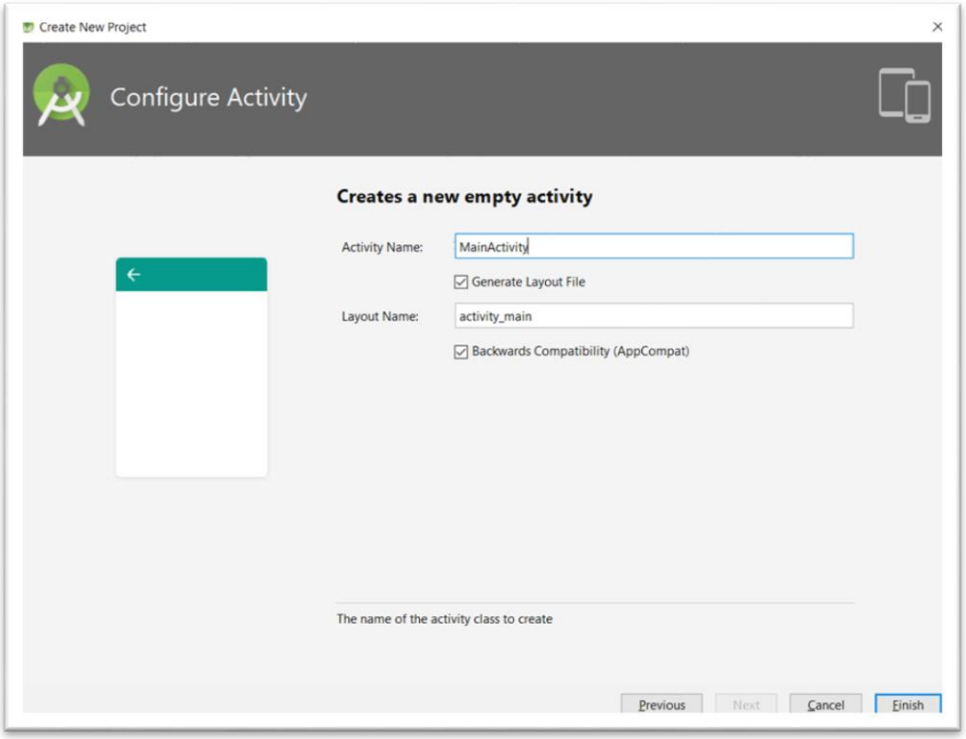


Figura 71. Name of activity and layout

When we finish the wizard, android studio runs Gradle scripts and shows us the environment where we will work.

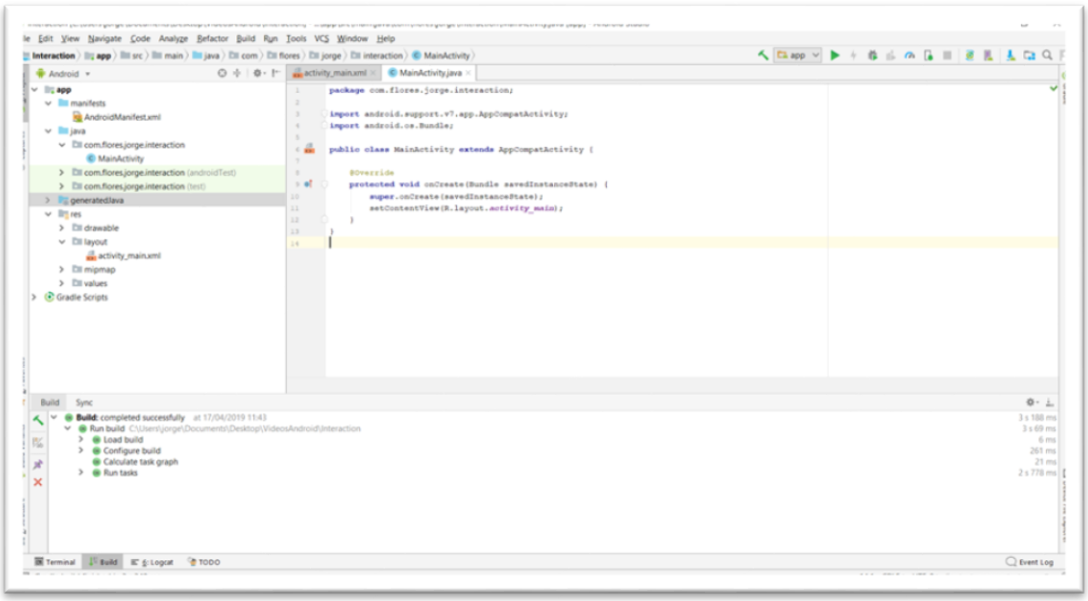


Figura 72. Environment

As we can see on the above Figura 72 we have a project structure at left of the screen.

These folders have all resources our application. For example, MainActivity.java include the logic of the activity. The file named activity_main.xml include the view of our activity. We could see in the next Figura 73.

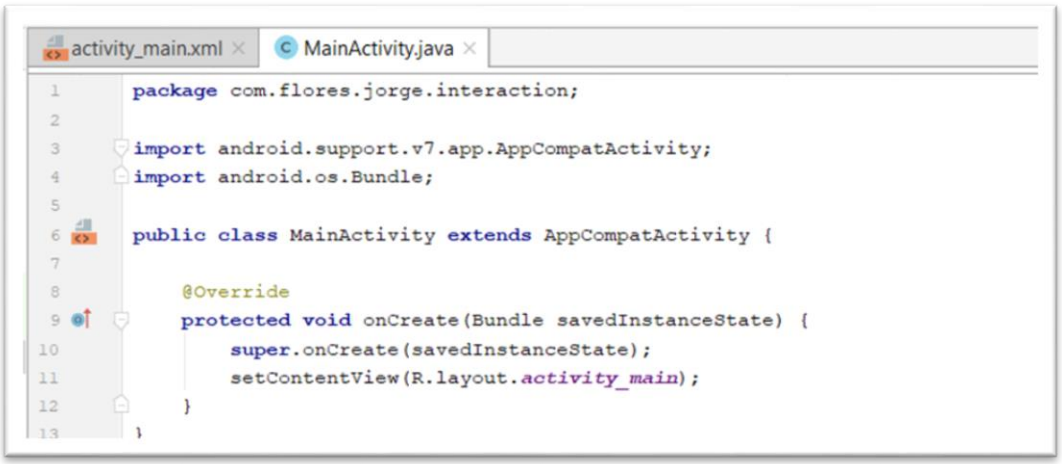


Figura 73. Activity Main

We should check API documentation to learn more about the different classes. AppCompatActivity extends of FrangmentActivity witch extends of Activity.

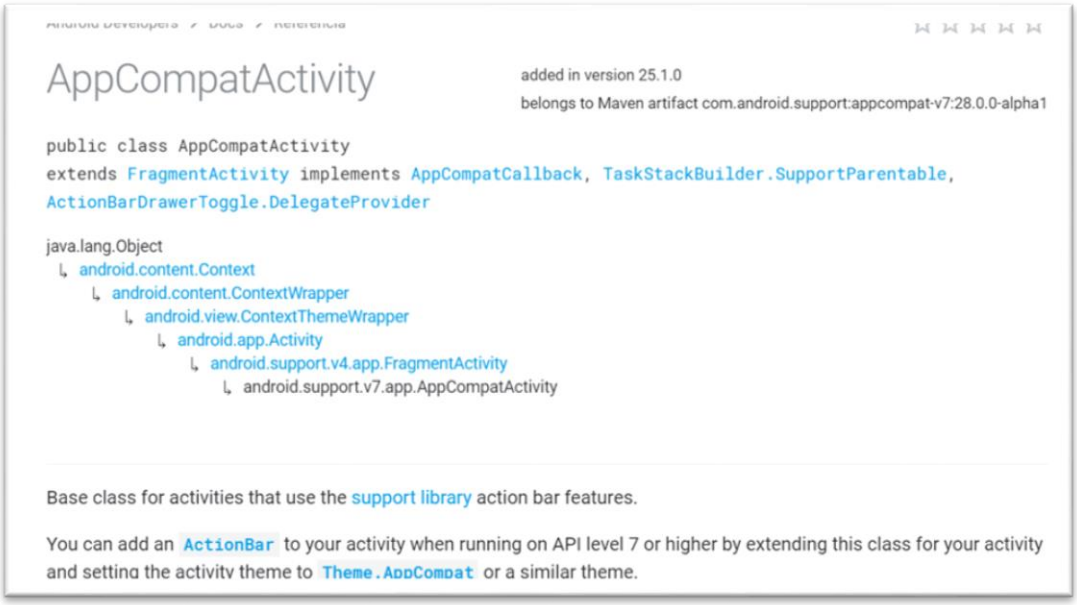


Figura 74. AppCompatActivity

Figura 75. Fragment Activity

We change AppCompatActivity for Activity as we could see in the next Figura 76.

```
1  package com.flores.jorge.interaction;
2
3  import android.app.Activity;
4  import android.os.Bundle;
5
6  public class MainActivity extends Activity {
7
8      @Override
9      protected void onCreate(Bundle savedInstanceState) {
10         super.onCreate(savedInstanceState);
11         setContentView(R.layout.activity_main);
12     }
13 }
```

Figura 76. Extends Activity

We could look in the API documentation for the class Activity and its method onCreate()

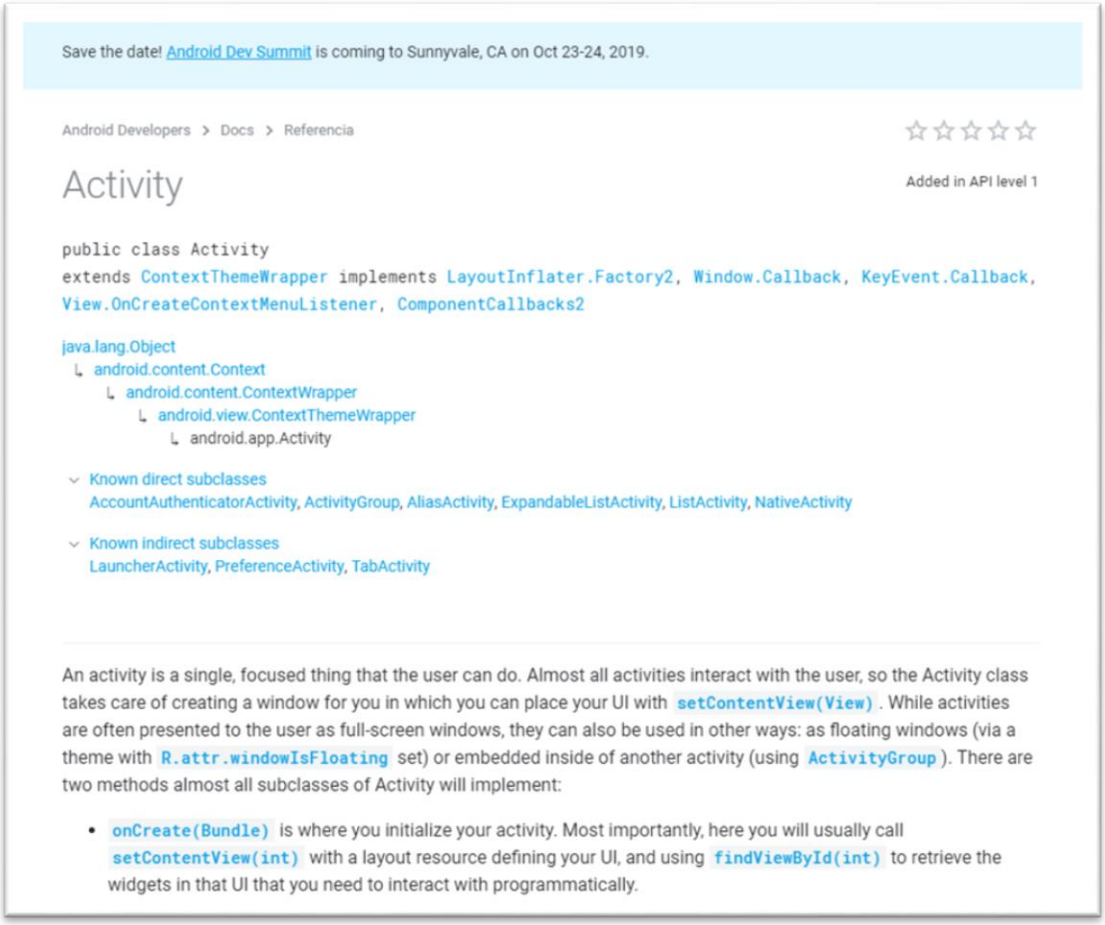


Figura 77. Activity Class

Witch this method shown below we can initialize the view. This view is accessible by file R.

```
setContentView(R.layout.activity_main);
```

Figura 78. setContentView

Another very important thing is the Activity LifeCycle. As a user navigates through, out of, and back to your app, the Activity instances in your app transition through different states in their lifecycle. The following Figura 79 shows the Activity LifeCycle.

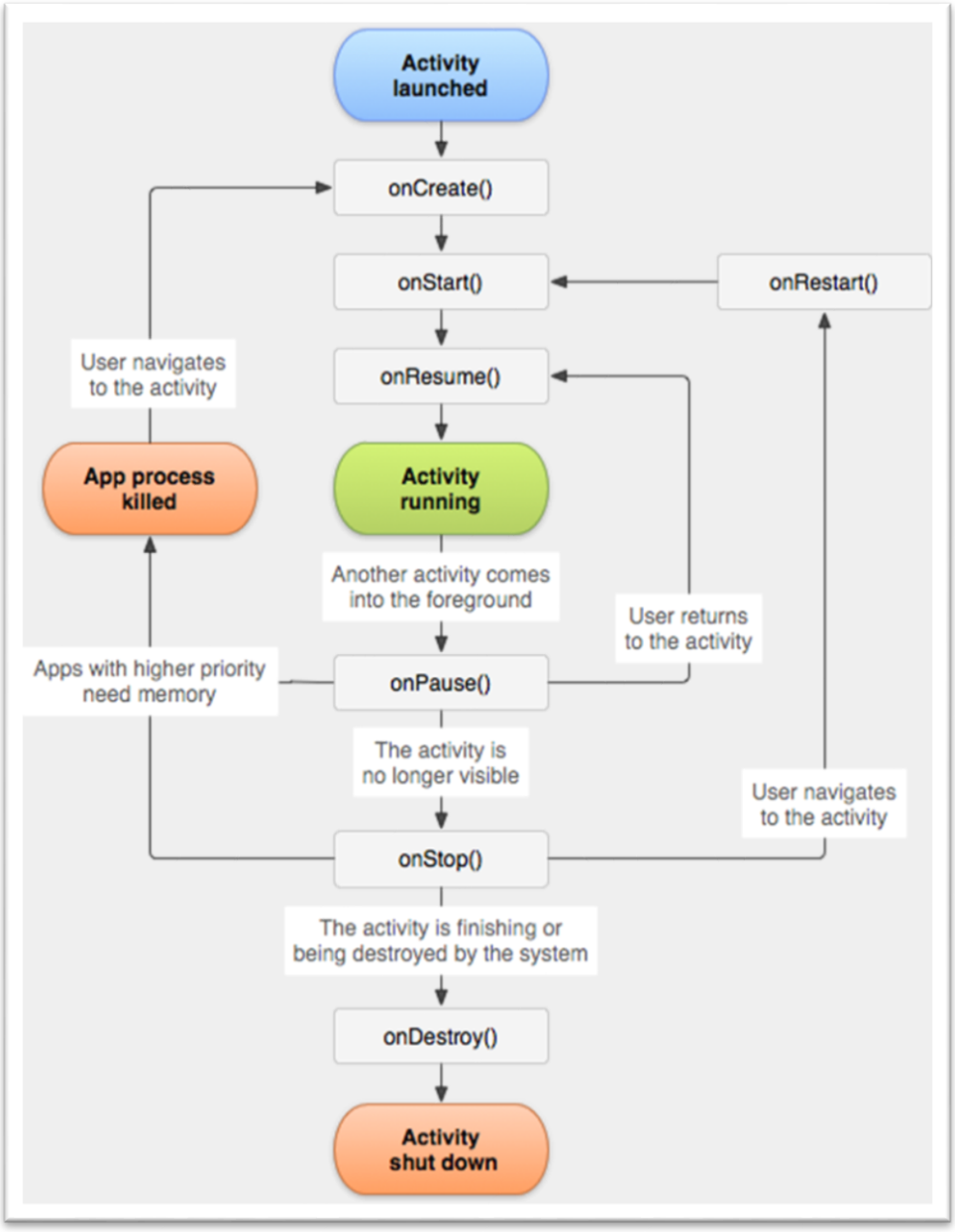


Figura 79. Activity LifeCycle. Ref (7.40)

In the file activity_main.xml we could see the graphical interface of this activity.

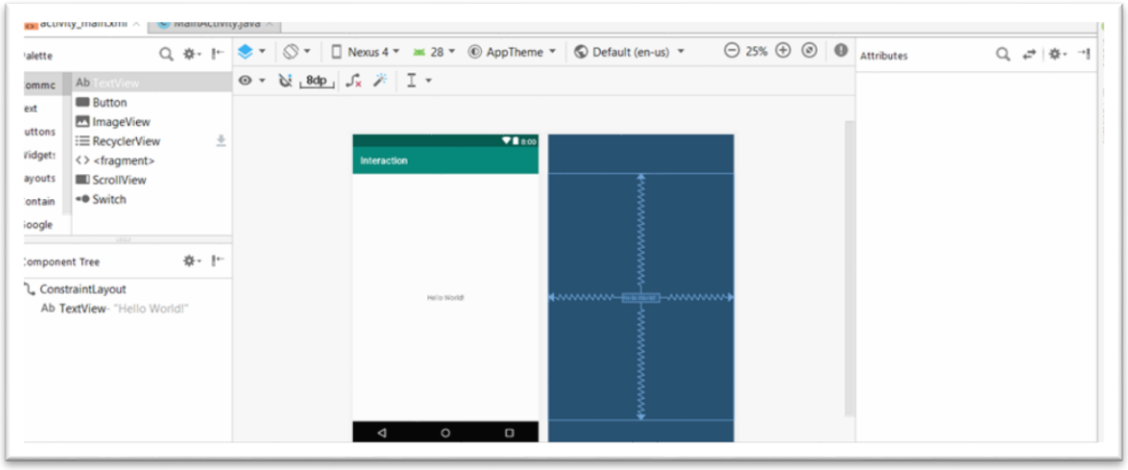


Figura 80. activity_main.xml

Furthermore, we could work with this view by xml syntax.

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</android.support.constraint.ConstraintLayout>
```

Figura 81. XML syntax

5.14 Test the Application

We test our application in several devices with the emulator or with our physical device.

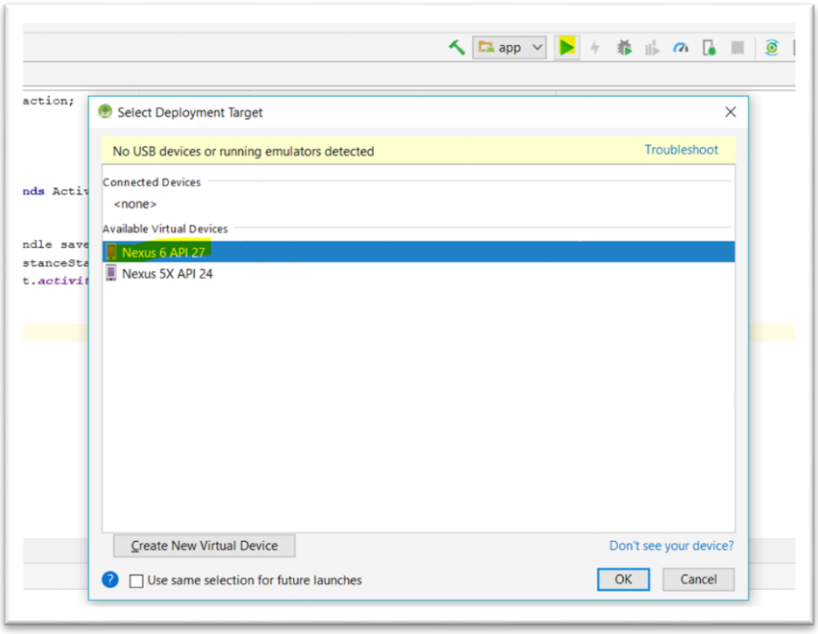


Figura 82. Test our app

As we could see in the next Figura 83 the application runs under an emulator.

Figura 83. Runs App

5.15 App speak us

To make the application talk to us we must to use TextToSpeech class which was implemented in API 4. This class synthesizes and convert into sound the text we introduce.

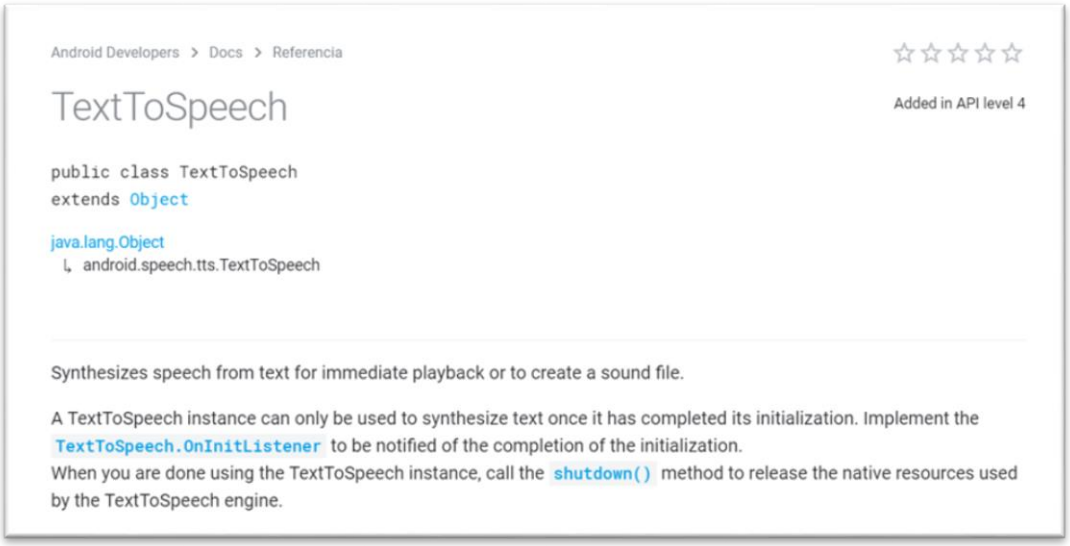


Figura 84. TextToSpeech Class

We need to implement OnInitListener interface like we could see as below image.

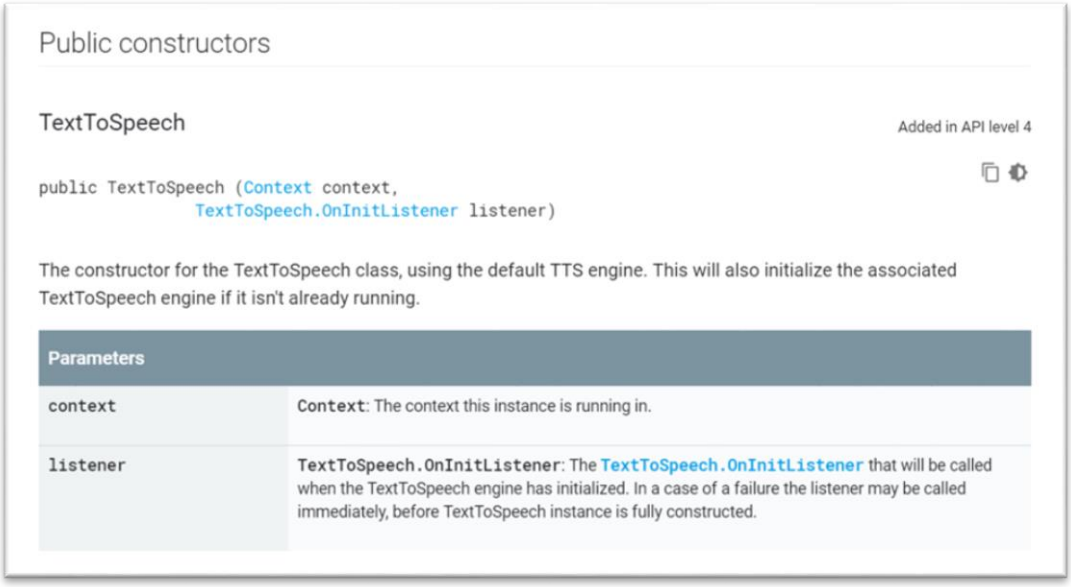
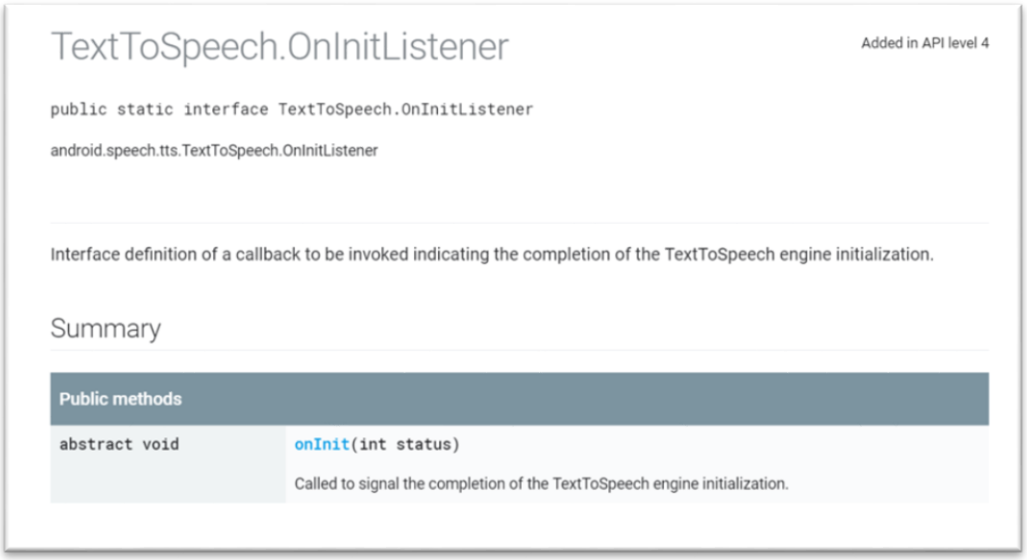


Figura 85. TextToSpeech Constructor

OnInitListener has to implement the method onInit() as we could see in the next Figura 86.



TextToSpeech.OnInitListener

Added in API level 4

```
public static interface TextToSpeech.OnInitListener
```

android.speech.tts.TextToSpeech.OnInitListener

Interface definition of a callback to be invoked indicating the completion of the TextToSpeech engine initialization.

Summary

Public methods	
abstract void	onInit(int status) Called to signal the completion of the TextToSpeech engine initialization.

Figura 86. onInit() Method

Finally, we have called a speak() method for creating an instance to begin to speak.

speak

Added in API level 21

```
public int speak (CharSequence text,
                  int queueMode,
                  Bundle params,
                  String utteranceId)
```

Speaks the text using the specified queuing strategy and speech parameters, the text may be spanned with TtsSpans. This method is asynchronous, i.e. the method just adds the request to the queue of TTS requests and then returns. The synthesis might not have finished (or even started!) at the time when this method returns. In order to reliably detect errors during synthesis, we recommend setting an utterance progress listener (see `setOnUtteranceProgressListener(UtteranceProgressListener)`) and using the `Engine#KEY_PARAM_UTTERANCE_ID` parameter.

Parameters	
text	**CharSequence**: The string of text to be spoken. No longer than `getMaxSpeechInputLength()` characters.
queueMode	**int**: The queuing strategy to use, `QUEUE_ADD` or `QUEUE_FLUSH`.
params	**Bundle**: Parameters for the request. Can be null. Supported parameter names: `Engine#KEY_PARAM_STREAM`, `Engine#KEY_PARAM_VOLUME`, `Engine#KEY_PARAM_PAN`. Engine specific parameters may be passed in but the parameter keys must be prefixed by the name of the engine they are intended for. For example the keys "com.svox.pico_foo" and "com.svox.pico:bar" will be passed to the engine named "com.svox.pico" if it is being used.
utteranceId	**String**: An unique identifier for this request.

Figura 87. Speak() method

We test this with a button to interact with the screen and, in this way, the application talk to us when we press the button.

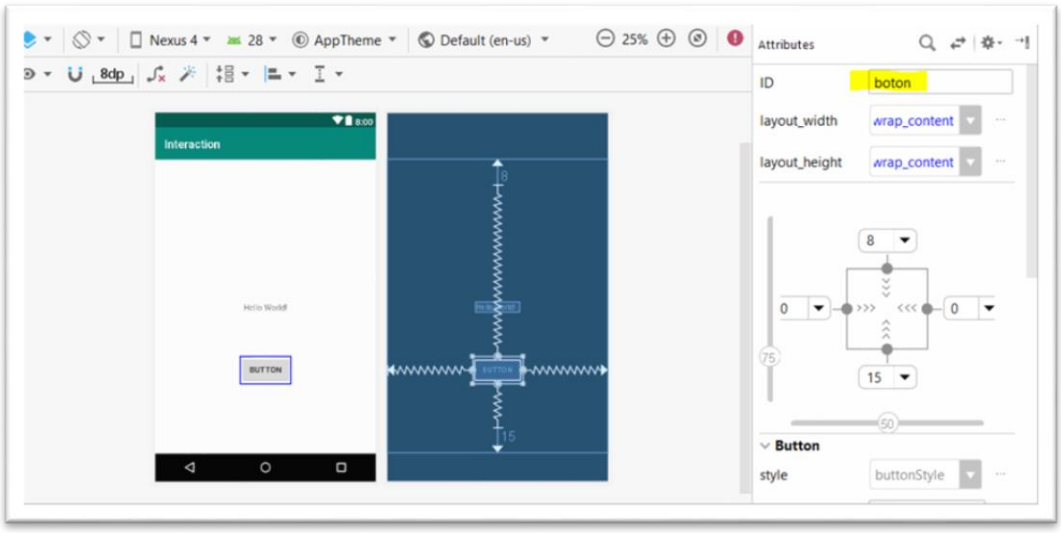


Figura 88. Insert Button

The code that we have created for this purpose is shown in the next Figura 89.

```
package com.flores.jorge.interaction;

import android.app.Activity;
import android.os.Bundle;
import android.speech.tts.TextToSpeech;
import android.view.View;

import java.util.Locale;

public class MainActivity extends Activity {

    TextToSpeech tl;


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);


        tl=new TextToSpeech(getApplicationContext(), new TextToSpeech.OnInitListener() {
            @Override
            public void onInit(int status) {
                if(status != TextToSpeech.ERROR) {
                    tl.setLanguage(Locale.getDefault());
                }
            }
        });

    }

    public void hablar(View boton){
        tl.speak( text: "Hola Aplicación", TextToSpeech.QUEUE_FLUSH,  params: null, utteranceId: null);
    }
}
```

Figura 89. Talk Code

Furthermore, we should configure the click event to manage the interaction.

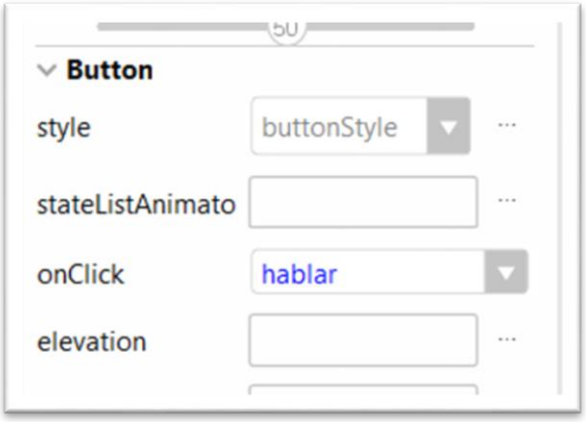


Figura 90. Click Event

In this way we relate the talking method to view (by the button view) when we click on the Button. Once we try to launch the application and press the button the application speaks to us.

Now, we want to make the application talk to us without having to press any button. Let it speak to us by default. The first thing we could think is that if we invoke the speak() method when we create the application it will work for us. To make it work we are going to create a thread and a handler of that thread (Handler Class) so that two seconds after starting the application automatically speaks to us. For it, we are going to remove the button too.

```
public class MainActivity extends Activity {

    TextToSpeech t1;
    private Handler temporizador;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);


        t1=new TextToSpeech(getApplicationContext(), new TextToSpeech.OnInitListener() {
            @Override
            public void onInit(int status) {
                if(status != TextToSpeech.ERROR) {
                    t1.setLanguage(Locale.getDefault());
                }
            }
        });

        temporizador=new Handler();
        temporizador.postDelayed(elhilo, delayMillis: 2000);

    }

    private  Runnable elhilo= new Runnable() {
        @Override
        public void run() {
            String toSpeak1 = "¿En que puedo ayudarte?";
            t1.speak(toSpeak1, TextToSpeech.QUEUE_FLUSH,  params: null, utteranceId: null);
        }
    };

}
```

Figura 91. Speak by default

We use postDelayed() to start the thread two seconds after the applications runs. To test the application we put our mobile device in USB debugging mode. (This is found in the Android development options, which are accessed differently depending on the device).

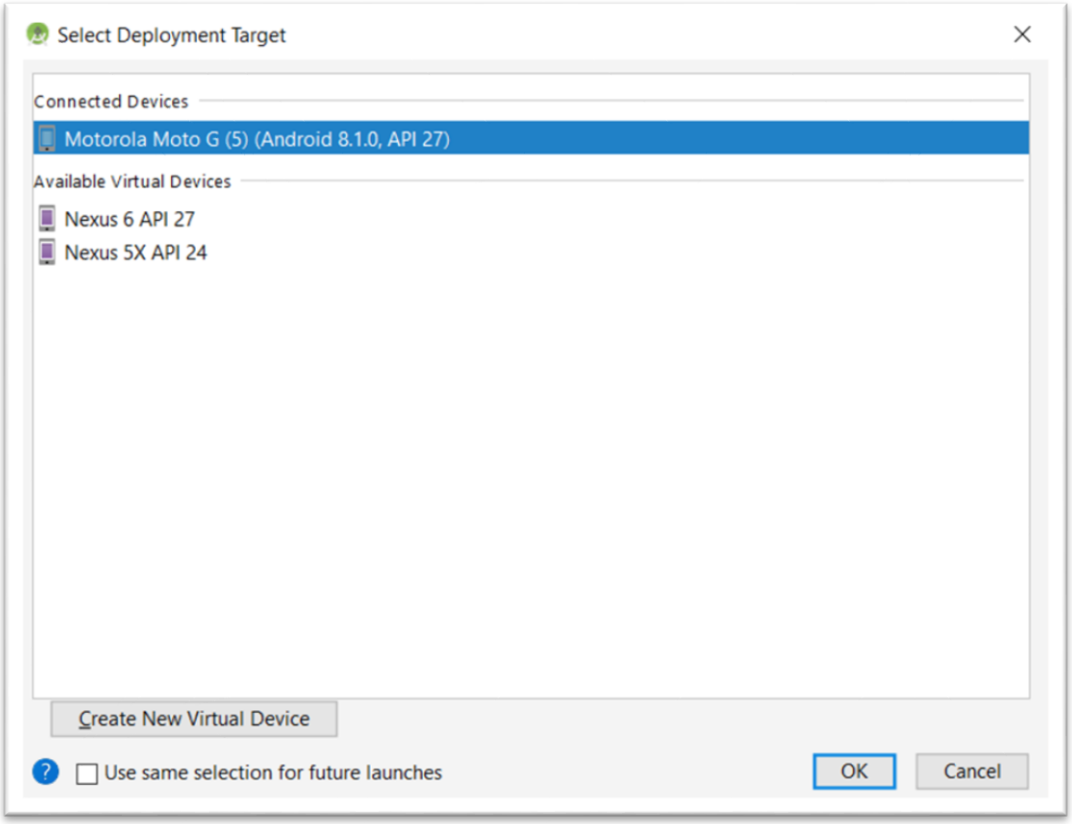
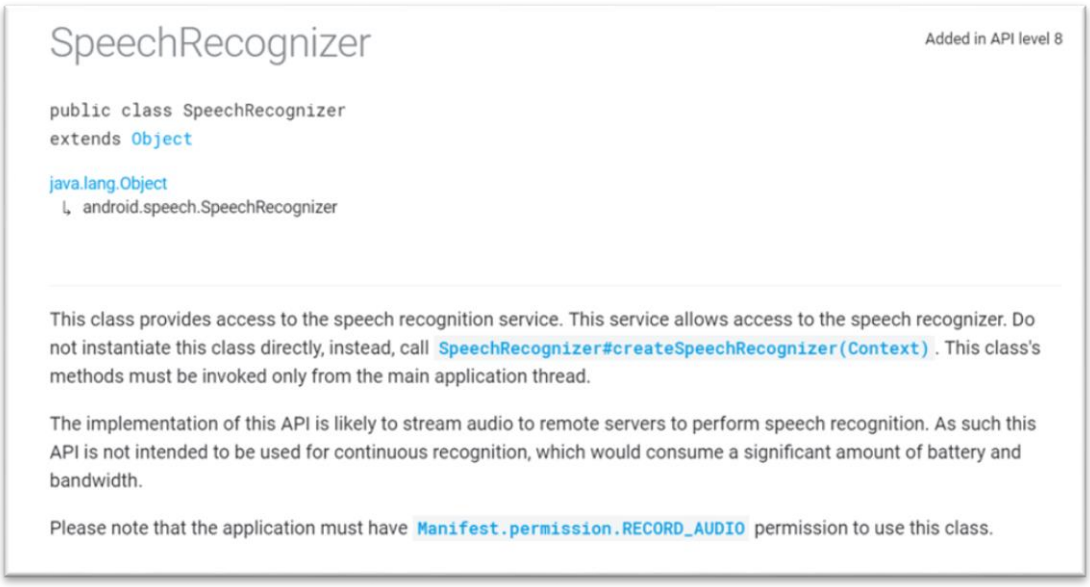


Figura 92. Test in our device

5.16 App listens to us

Our application must listen what we say. For this purpose we are using SpeechRecognizer class. This class was added in API level 8. The application forces us to give permissions to record Audio. We indicate this permission in the Manifest file. Also, API show us that shouldn't create the instance directly by the class. We should call the method createSpechRecognizer().



SpeechRecognizer

Added in API level 8

```
public class SpeechRecognizer
extends Object
```

java.lang.Object
↳ android.speech.SpeechRecognizer

This class provides access to the speech recognition service. This service allows access to the speech recognizer. Do not instantiate this class directly, instead, call `SpeechRecognizer#createSpeechRecognizer(Context)`. This class's methods must be invoked only from the main application thread.

The implementation of this API is likely to stream audio to remote servers to perform speech recognition. As such this API is not intended to be used for continuous recognition, which would consume a significant amount of battery and bandwidth.

Please note that the application must have `Manifest.permission.RECORD_AUDIO` permission to use this class.

Figura 93. SpeechRecognizer

We also need to implement a Listener that listens. We're going to do it with setRecognitionListener().

setRecognitionListener

Added in API level 8

```
public void setRecognitionListener (RecognitionListener listener)
```

Sets the listener that will receive all the callbacks. The previous unfinished commands will be executed with the old listener, while any following command will be executed with the new listener.

Parameters	
`listener`	**`RecognitionListener`**: listener that will receive all the callbacks from the created `SpeechRecognizer`, this must not be null.

Figura 94. setRecognitionListener

We are going to instantiate a new class and we are going to implement the RecognitionListerner interface where we will overwrite all the methods that we need.

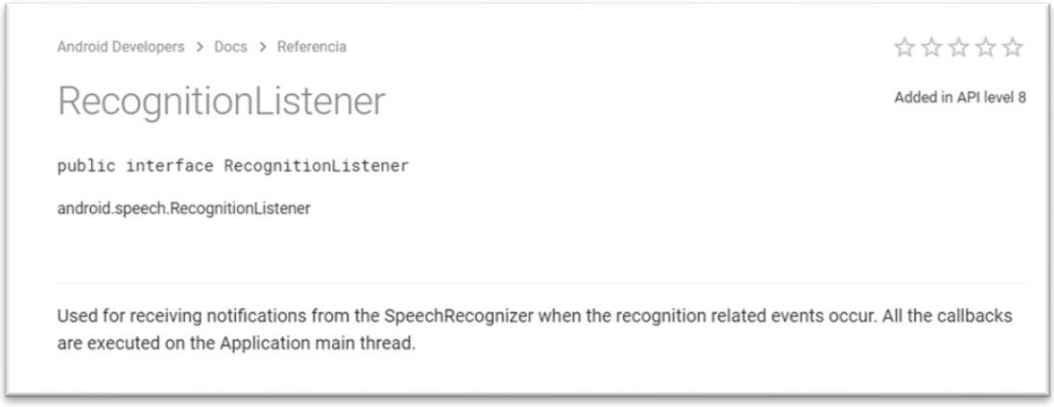


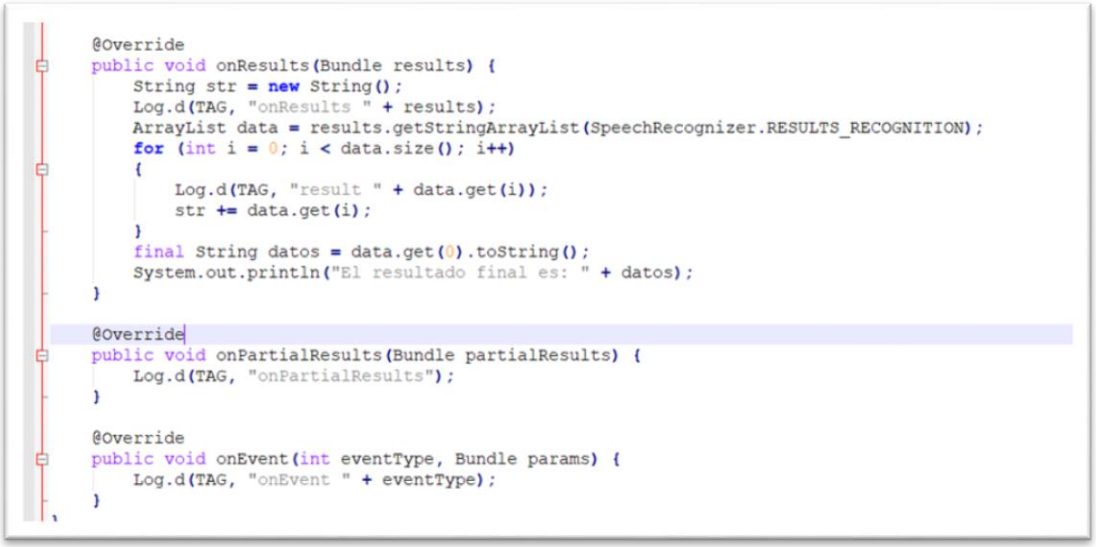
Figura 95. RecognitionListerner interface

We implemented all these methods because, in this way, we could get the control of the result set (The information that we speak and the application listened).

```
class ClaseEscucha implements RecognitionListener {
    private static final String TAG = "MyActivity";
    Context myContext;

    public ClaseEscucha(Context myContext){
        this.myContext = myContext;
    }

    @Override
    public void onReadyForSpeech(Bundle params) {
        Log.d(TAG, "onReadyForSpeech");
        Toast.makeText(myContext, "EMPIEZA A HABLAR",Toast.LENGTH_SHORT).show();
    }

    @Override
    public void onBeginningOfSpeech() {
        Log.d(TAG, "onBeginningOfSpeech");
    }

    @Override
    public void onRmsChanged(float rmsdB) {
        Log.d(TAG, "onRmsChanged");
    }

    @Override
    public void onBufferReceived(byte[] buffer) {
        Log.d(TAG, "onBufferReceived");
    }

    @Override
    public void onEndOfSpeech() {
        Log.d(TAG, "onEndofSpeech");
        Toast.makeText(myContext, "SE ACABO",Toast.LENGTH_SHORT).show();
    }

    @Override
    public void onError(int error) {
        Log.d(TAG,  "error " +  error);
    }
```

Figura 96. RecongnitionListener Implements

We can see in the next Figura 97 and how get the data and processing it.



```
@Override
public void onResults(Bundle results) {
    String str = new String();
    Log.d(TAG, "onResults " + results);
    ArrayList data = results.getStringArrayList(SpeechRecognizer.RESULTS_RECOGNITION);
    for (int i = 0; i < data.size(); i++)
    {
        Log.d(TAG, "result " + data.get(i));
        str += data.get(i);
    }
    final String datos = data.get(0).toString();
    System.out.println("El resultado final es: " + datos);
}

@Override
public void onPartialResults(Bundle partialResults) {
    Log.d(TAG, "onPartialResults");
}

@Override
public void onEvent(int eventType, Bundle params) {
    Log.d(TAG, "onEvent " + eventType);
}
```

Figura 97. onResult Method

We also created the instance of the Handler for listening and started it with the postDelayed() method 3 seconds later. We do to avoid getting mixed up with the question we ask at the beginning of the application.

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    sr = SpeechRecognizer.createSpeechRecognizer(this);
    sr.setRecognitionListener(new ClaseEscucha(getApplicationContext()));
    temporizadorHablar=new Handler();
    temporizadorEscucha=new Handler();
    temporizadorHablar.postDelayed(hiloHabla, delayMillis: 1000);
    temporizadorEscucha.postDelayed(hiloEscucha, delayMillis: 3000);
}
```

Figura 98. Listen Thread

Inside the listen Thread we implemented run() method for start the method called startVoiceInput() as show as below:

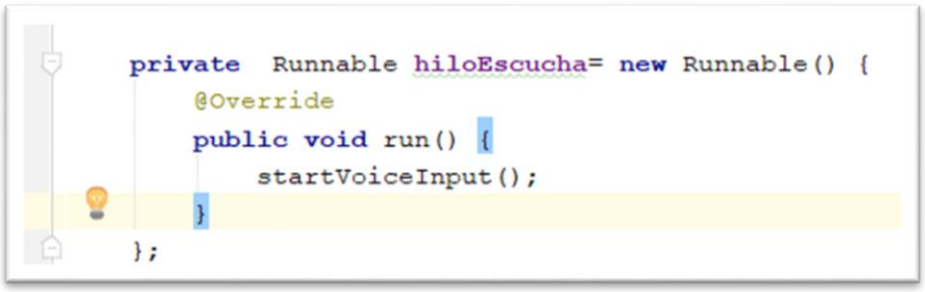


Figura 99. Implements run() method

We could see in the next Figura 100 the startVoiceInput() and how to create an Intent class to begin the recognition of the user voice.

Figura 100. startVoiceInput() method

Finally, we override onDestroy() method to destroy all instances.

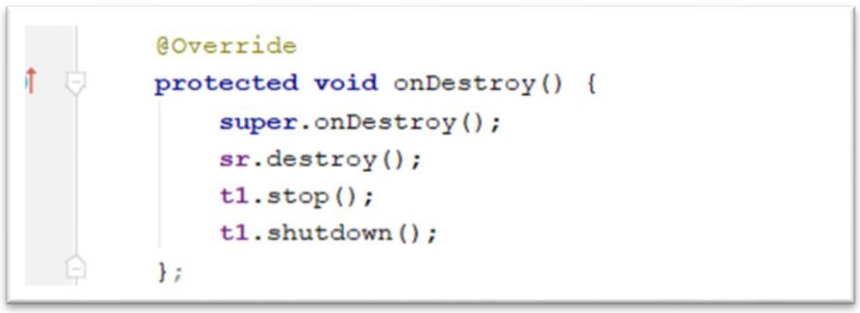


Figura 101. onDestroy() method

Getting all permissions is very important. For this, we introduce the next code in the Android manifest.xml as we can see in the next Figura 102.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.flores.jorge.interaction">
    <!--parra dar permisos de audio si se implementa la interfaz-->
    <uses-permission android:name="android.permission.RECORD_AUDIO" />
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="Interaction"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Figura 102. Android Permission

We notice how, although we get permissions in the Android manifest file the application doesn't work. To fix that, we change the permissions on the device.

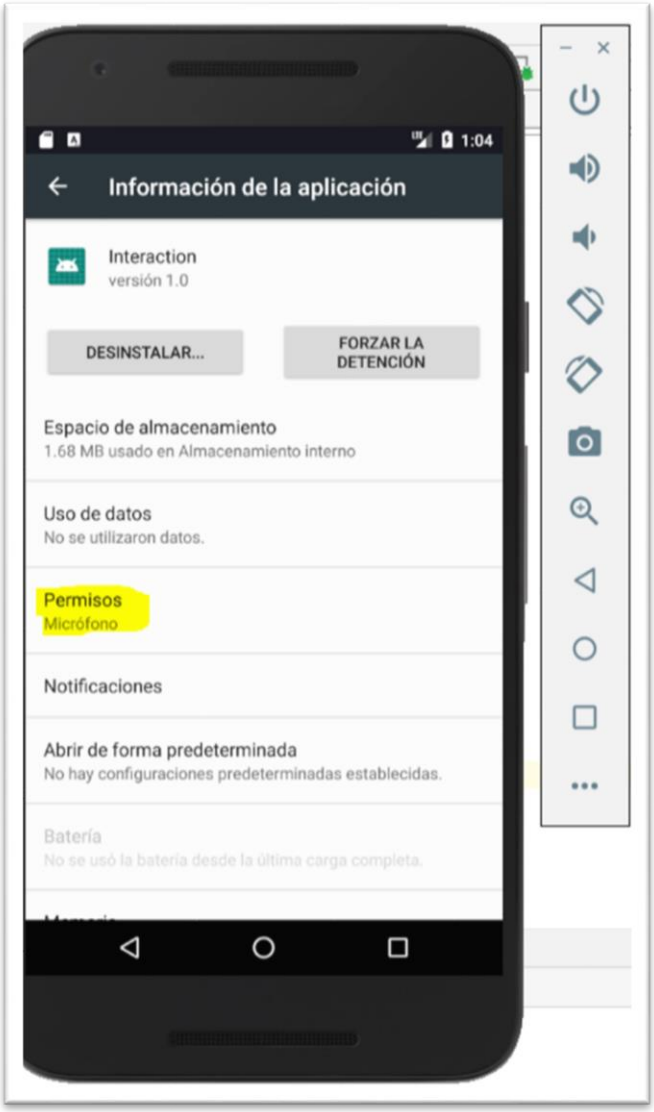


Figura 103. Device configuration

Once we test the application and run it we can see the Log and see exactly the words that we want to say to the application.

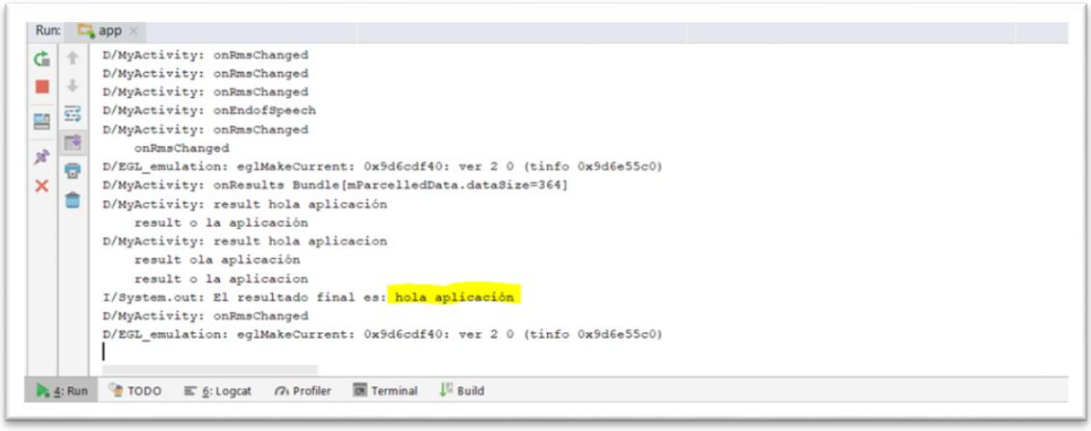


Figura 104. Final Result

We already have the basic flow. We could use finish() method when the user says "salir" to exit the application. We close the applications always, in order to avoid problems with his devices.

```
1    public void elegir(String datos){
2        if(datos.equalsIgnoreCase( anotherString: "salir")){
3            myAcivity.finish();
4        }
5    }
```

Figura 105. Finish() method

We put in the drawable folder the icons that we have used in the Java Application. We show that icons in the Android Application too.

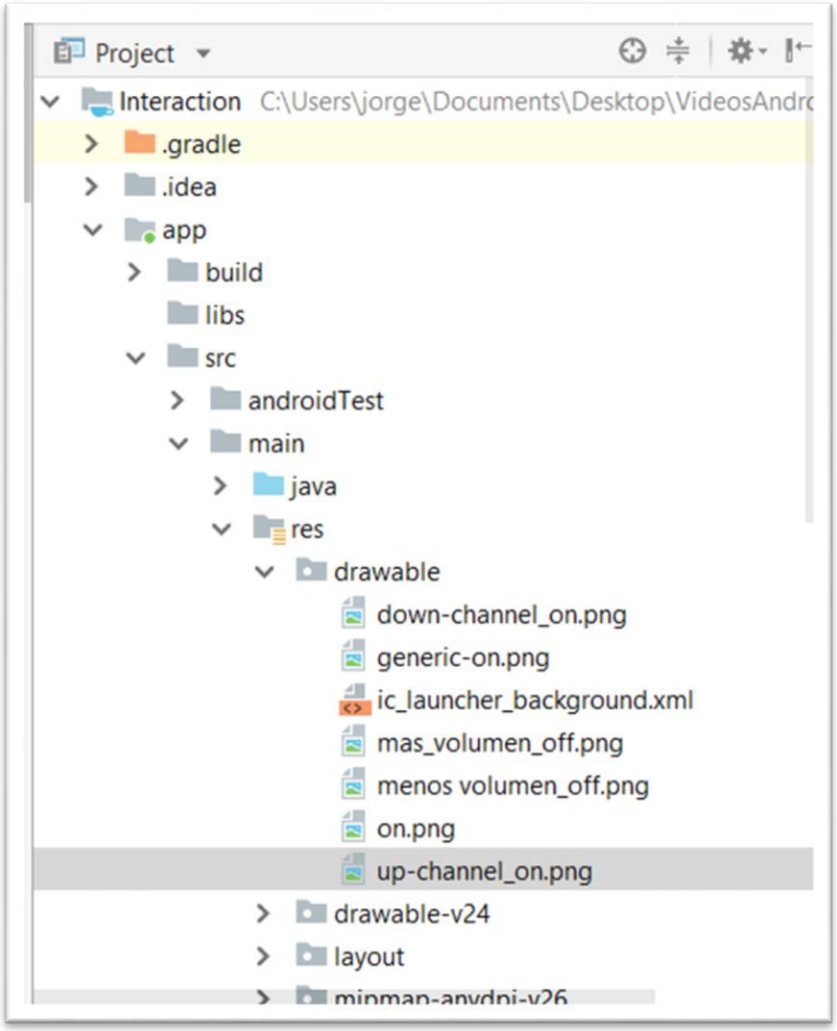


Figura 106. Icons

In the view we put the property visibility=”invisible” so that we don´t show icons when the application starts.

```
<ImageView
    android:id="@+id/imageView"
    android:layout_width="216dp"
    android:layout_height="218dp"
    android:layout_marginStart="146dp"
    android:layout_marginTop="179dp"
    android:layout_marginEnd="238dp"
    android:layout_marginBottom="331dp"
    android:src="@drawable/on"
    app:layout_constraintBottom_toBottomOf="paren
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.287"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.221"
    android:visibility="invisible"
    />
```

Figura 107. Visibility

We mapped the view with the listen class with findViewByID() method which allow us to use the view objects in the java code.

```
private static final String TAG = "MyActivity";
private Context myContext;
private Activity myAcivity;
ImageView image;
public ClaseEscucha(Context myContext, Activity myActivity){
    this.myContext = myContext;
    this.myAcivity =myActivity;
    image = (ImageView)myAcivity.findViewById(R.id.imageView) ;
}
```

Figura 108. Map the object view

In the elegirDatos() method we change the image according with our voice and we put the property visibility at visible.

```
public void elegir(String datos){
    if(datos.equalsIgnoreCase( anotherString: "salir")){
        myAcivity.finish();
    }
    if(datos.equalsIgnoreCase( anotherString: "encender televisión")){
        System.out.println("TELEVISION ENCENDIDA");
        image.setImageResource(R.drawable.on);
        image.setVisibility(View.VISIBLE);
    }
    if(datos.equalsIgnoreCase( anotherString: "apagar televisión")){
        System.out.println("TELEVISION APAGADA");
        image.setImageResource(R.drawable.on);
        image.setVisibility(View.VISIBLE);
    }
    if(datos.equalsIgnoreCase( anotherString: "subir volumen")){
        System.out.println("SUBIR VOLUMEN");
        image.setImageResource(R.drawable.channel_up);
        image.setVisibility(View.VISIBLE);
    }
    if(datos.equalsIgnoreCase( anotherString: "bajar volumen")){
        System.out.println("BAJAR VOLUMEN");
        image.setImageResource(R.drawable.volume_down);
        image.setVisibility(View.VISIBLE);
    }
    if(datos.equalsIgnoreCase( anotherString: "subir canal")){
        System.out.println("SUBIR CANAL");
        image.setImageResource(R.drawable.channel_up);
        image.setVisibility(View.VISIBLE);
    }
    if(datos.equalsIgnoreCase( anotherString: "bajar canal")){
        System.out.println("BAJAR CANAL");
        image.setImageResource(R.drawable.down_channel);
        image.setVisibility(View.VISIBLE);
    }
```

Figura 109. elegir() Method

We override onError() method and we display an error image to show him an alert if the application fails.

```
@Override
public void onError(int error) {
    Log.d(TAG,  msg: "error " +  error);
    image.setImageResource(R.drawable.error);
    image.setVisibility(View.VISIBLE);
}
```

Figura 110. onError() Method

Once we have the main Activity we need to create another Activity to save the IP address information and a port to communicate with the server.

To do this, we will create a new layout called configuration.xml and a new class that extends the activity called configurationApp.

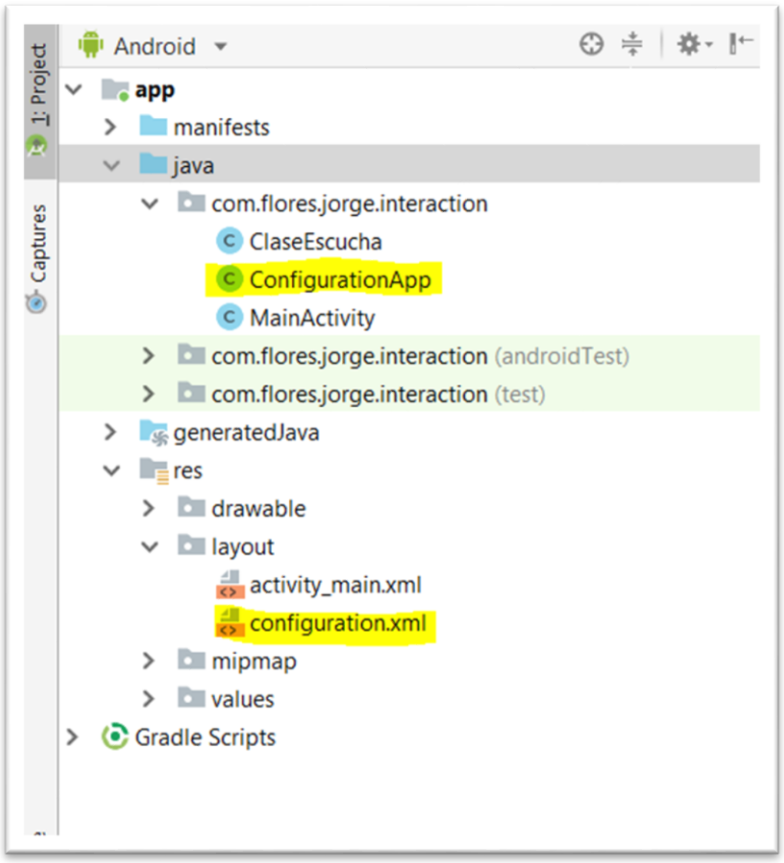


Figura 111. Configuation Activity

We have designed the following interface as we can see in the next Figura 112.

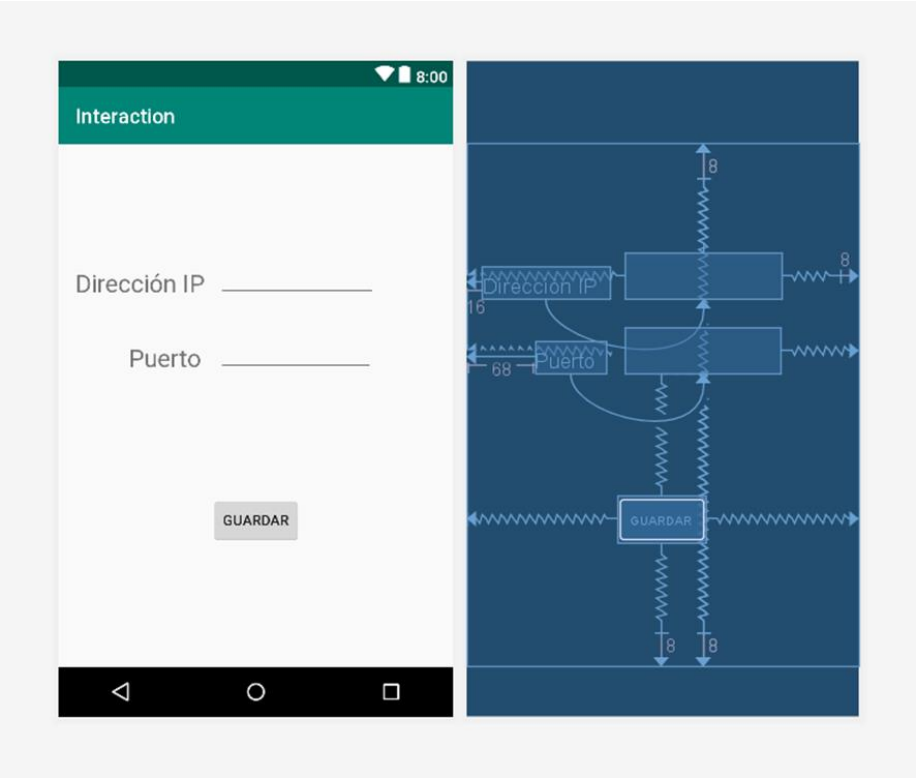


Figura 112. Configuration Interface

The fields will be set as text for the IP direction and as numbers for the port.

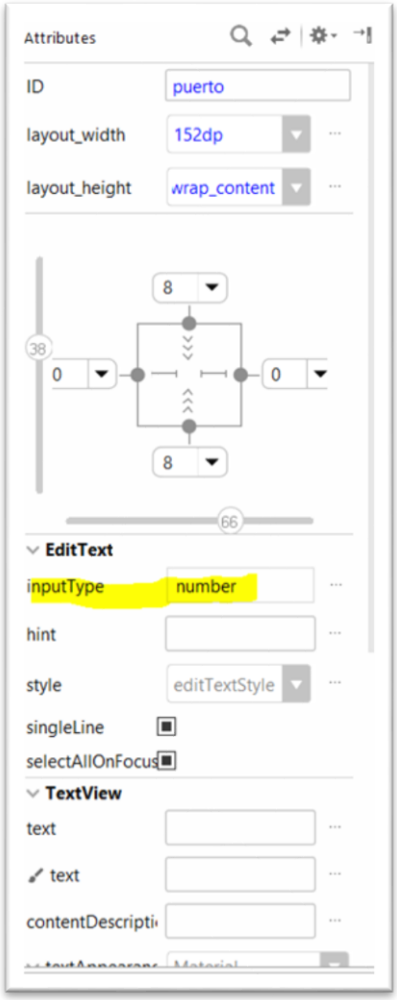


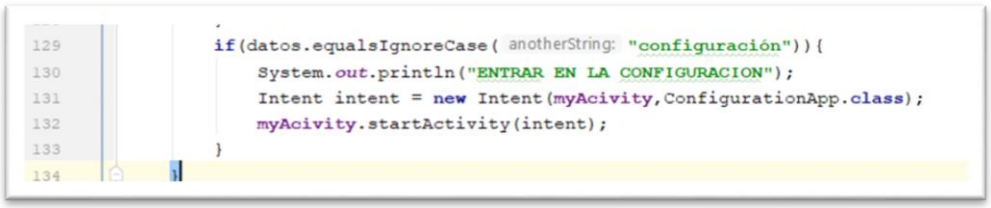
Figura 113. Port as a number

We extend configurationApp.java with the class Activity and set the view with onCreate() method.

```
package com.flores.jorge.interaction;

import android.app.Activity;
import android.os.Bundle;

public class ConfigurationApp extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.configuration);
    }

}
```

Figura 114. configurationApp Class

We have created one Intent Class to change the Activity when the user says the word configuration. We use startActivity() method to start a new activity.



```
        if(datos.equalsIgnoreCase( anotherString: "configuración")){
            System.out.println("ENTRAR EN LA CONFIGURACION");
            Intent intent = new Intent(myAcivity,ConfigurationApp.class);
            myAcivity.startActivity(intent);
        }
    }
```

Figura 115. New Activity

We must put the activity in the manifest.xml

```
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    <activity android:name=".MainActivity">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
    <activity android:name=".ConfigurationApp"></activity>
</application>
```

Figura 116. Manifest with new activity

Now, when we launch the application from the emulator and say the word "configuración" we see the new activity.

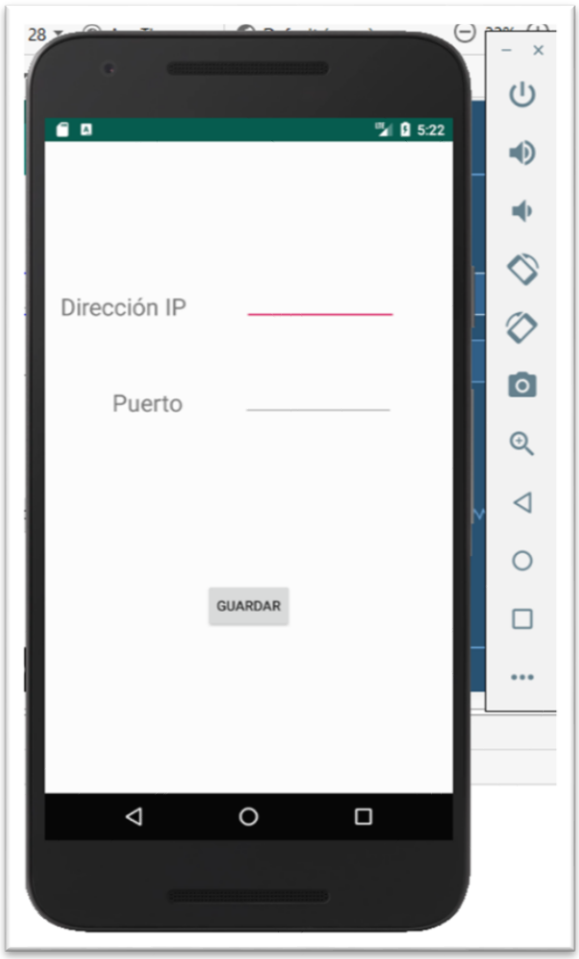
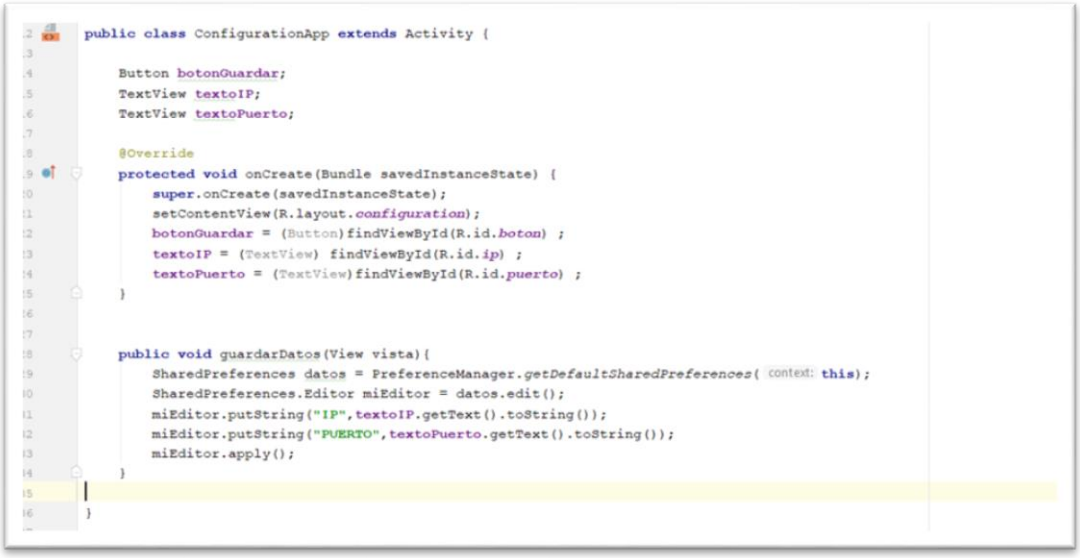


Figura 117. Configuration Activity

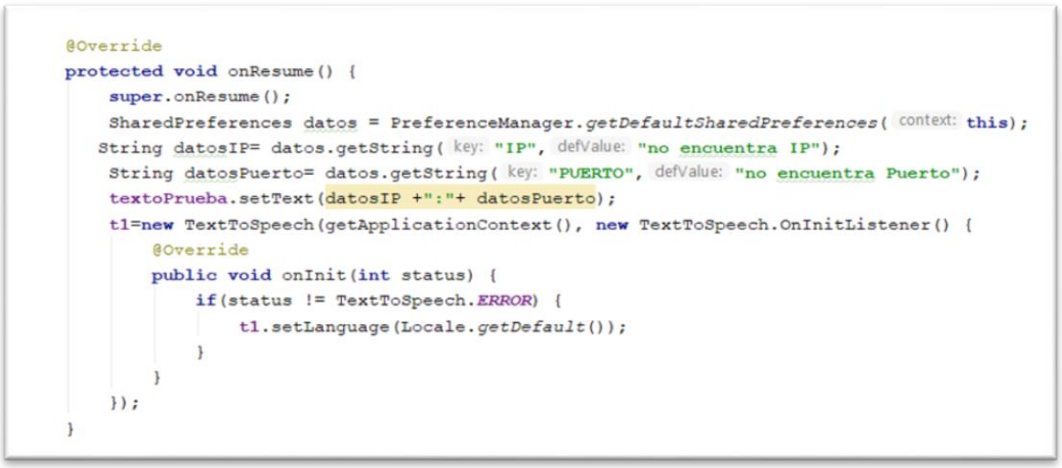
In addition, we are going to save the information sent to the microcontroller. We are going to do that with SharedPreferences. This way we will be able to save data permanently in the application without using BBDD or external files.



```
public class ConfigurationApp extends Activity {

    Button botonGuardar;
    TextView textoIP;
    TextView textoPuerto;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.configuration);
        botonGuardar = (Button)findViewById(R.id.boton) ;
        textoIP = (TextView) findViewById(R.id.ip) ;
        textoPuerto = (TextView)findViewById(R.id.puerto) ;
    }


    public void guardarDatos(View vista){
        SharedPreferences datos = PreferenceManager.getDefaultSharedPreferences( context: this);
        SharedPreferences.Editor miEditor = datos.edit();
        miEditor.putString("IP",textoIP.getText().toString());
        miEditor.putString("PUERTO",textoPuerto.getText().toString());
        miEditor.apply();
    }

}
```

Figura 118. Save Shared Preferences

In our MainActivity we put a Text View to be able to see what we are saving. In the onResume() method we obtain the preferences shared by default and we save the data in variables, which we will later show in a Text View.



```
@Override
protected void onResume() {
    super.onResume();
    SharedPreferences datos = PreferenceManager.getDefaultSharedPreferences( context: this);
   String datosIP= datos.getString( key: "IP", defValue: "no encuentra IP");
    String datosPuerto= datos.getString( key: "PUERTO", defValue: "no encuentra Puerto");
    textoPrueba.setText(datosIP +":"+ datosPuerto);
    t1=new TextToSpeech(getApplicationContext(), new TextToSpeech.OnInitListener() {
        @Override
        public void onInit(int status) {
            if(status != TextToSpeech.ERROR) {
                t1.setLanguage(Locale.getDefault());
            }
        }
    });
}
```

Figura 119. Get SharedPreferences

We could see the IP a port at the screen.

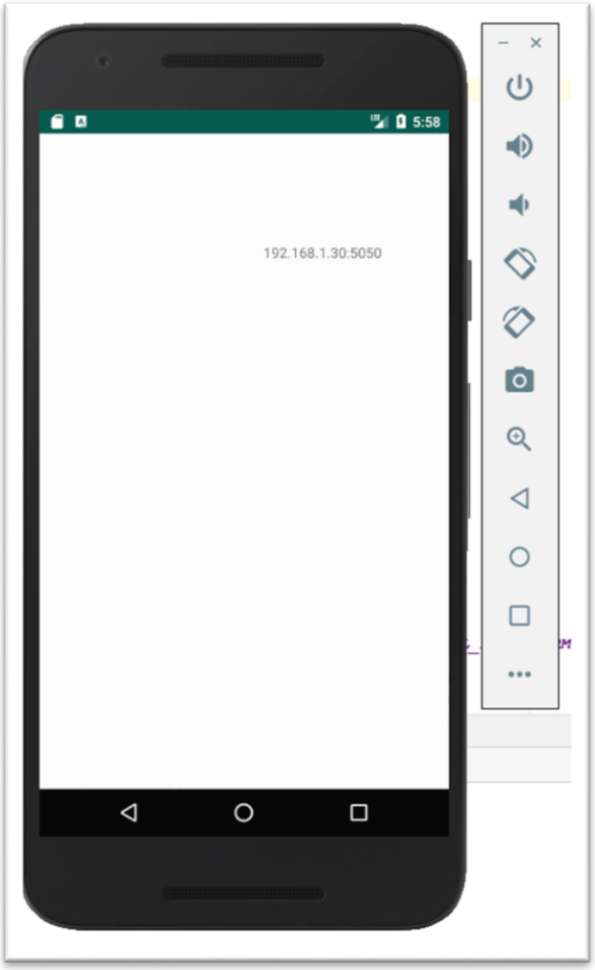


Figura 120. Ip and Port

5.17 Send data to the server

We are going to reuse the Client class. This class is the same class that we used for the Java application. We are going to make the connection with another wire to establish the connection and for now, we are going to test if it works. To do that, we send the instruction turn on television.

We create a new Handler which initialize the constructor of the class named ClassListen. We create a new class of client and we try to make the connection with the Server.

```
public void elegir(String datos){
    if(datos.equalsIgnoreCase( anotherString: "salir")){
        myAcivity.finish();
    }
    if(datos.equalsIgnoreCase( anotherString: "encender televisión")){
        System.out.println("TELEVISION ENCENDIDA");
        image.setImageResource(R.drawable.on);
        image.setVisibility(View.VISIBLE);
       final Client newClient = new Client();
        Runnable elhilo3= new Runnable() {
            @Override
            public void run() {
                try {
                    Thread.sleep( millis: 2000);
                    newClient.openConnection( ip: "192.168.1.37", puerto: 5050);
                    newClient.openStream();
                    newClient.send( s: "hola envio");
                    newClient.closeConnection();
                }catch (Exception e){
                    Log.d(TAG,   msg: "error " +  newClient.toString());
                }
            }
        };
        temporizador3.postDelayed(elhilo3, delayMillis: 1000);
    }
```

Figura 121. Cient Class

When we run the application and say “encender television”, we have a permission error.

```
n:  app
I/System.out: El resultado final es: encender television
    TELEVISION ENCENDIDA
D/EGL_emulation: eglMakeCurrent: 0x9ff852a0: ver 2 0 (tinfo 0x9ff83630)
D/MyActivity: onRmsChanged
    onRmsChanged
I/System.out: Excepción al levantar conexión: Permission denied
D/MyActivity: error com.flores.jorge.interaction.Client@e740944
```

Figura 122. Error permission

In the manifest.xml file we are going to include the permissions to connect us via wifi.

```
<uses-permission android:name="android.permission.INTERNET" >
</uses-permission>

<!--<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" >-->
<!--</uses-permission>-->
<uses-feature android:name="android.hardware.wifi" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"></uses-permission
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"></uses-permission>
```

Figura 123. Wifi and network permissions

But it doesn't work and no error log appears when trying to capture it.

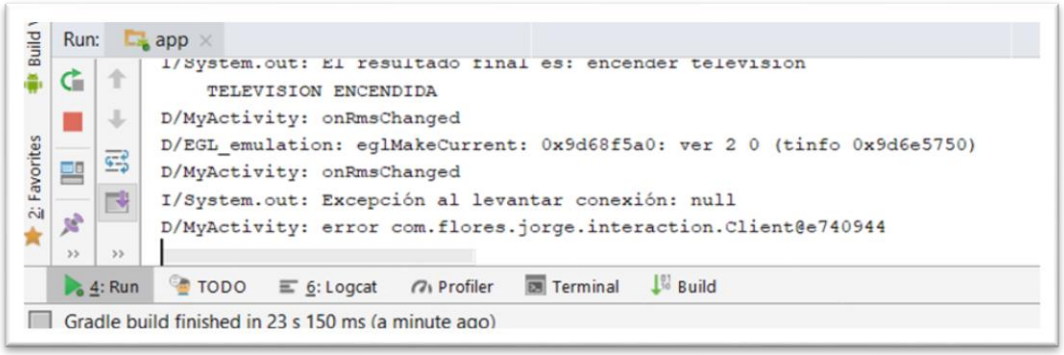


Figura 124. Terminal for debugging

Try to find a solution we see in StackOverflow two possible fixes and the reason why this happens. We are trying to make the connection to the network from the main thread. We have to change the thread policies or use AsyncTask Class.

Figura 125. Error solution StackOverflow

When we use AsyncTask Class it works.

```
218     class envioDatos extends AsyncTask<String,Void,Void>{
219         Client newClient = new Client();
220         public envioDatos() {
223
224         @Override
225         protected void onPreExecute() {
228
229         @Override
230         protected void onPostExecute(Void aVoid) {
234
235         @Override
236         protected void onProgressUpdate(Void... values) {
239
240         @Override
241         protected void onCancelled(Void aVoid) {
244
245         @Override
246         protected void onCancelled() {
249
250         @Override
251         protected Void doInBackground(String... datoEnviar) {
252             String cadenaTexto = datoEnviar[0];
253             try {
254                 Thread.sleep(2000);
255                 newClient.openConnection("192.168.0.25",5050);
256                 newClient.openStream();
257                 newClient.send(cadenaTexto);
258                 newClient.closeConnection();
259             }catch (Exception e){
260                 Log.d(TAG,  "error vista1" +  newClient.toString());
261             }
262             return null;
263         }
264     }
```

Figura 126. AsyncTask

We are also going to change the way we connect to the server by modifying the response time. If this does not give us an answer in 3 seconds we finish the connection.

```
6       public void openConnection(String ip, int puerto)  throws Exception{
7           try {
8               socket = new Socket();
9               socket.connect(new InetSocketAddress(ip,puerto), timeout: 300);
0               System.out.println("Conectado a :" + socket.getInetAddress().getHostName());
1           } catch (Exception e) {
2               System.out.println("Excepción al levantar conexión: " + e.getMessage());
3           }
4       }
```

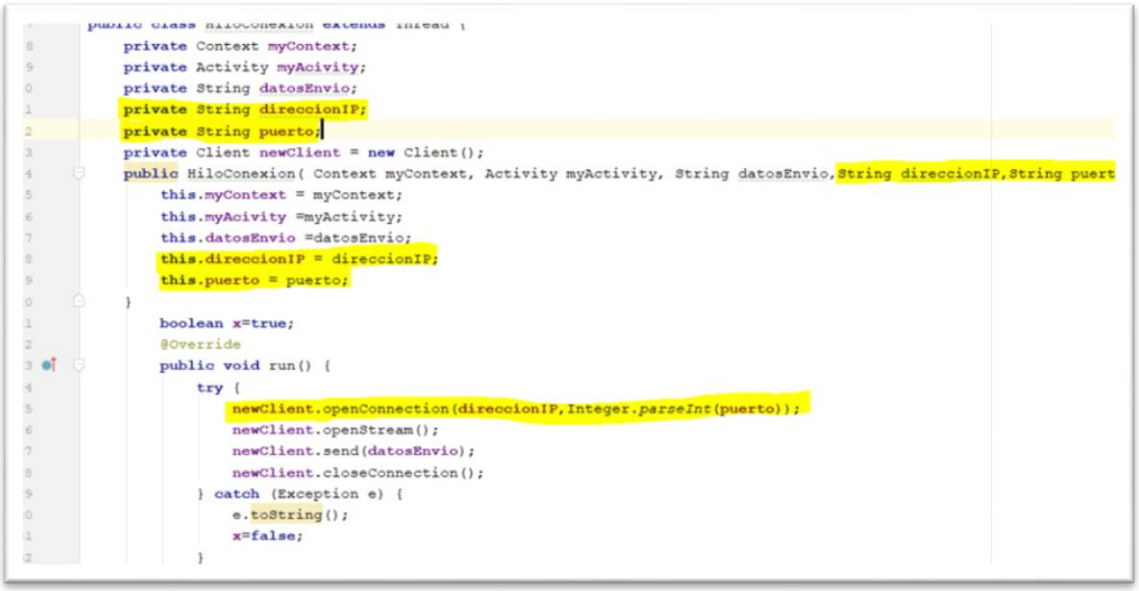
Figura 127. Timeout connection

Finally, we structure better the code and we put all the corresponding calls to launch the corresponding thread in case we want to turn on the TV, increase volume, etc

```
public void elegir(String datos){
    if(datos.equalsIgnoreCase( anotherString: "salir")){
        myAcivity.finish();
    }
    if(datos.equalsIgnoreCase( anotherString: "encender televisión")){
        System.out.println("TELEVISION ENCENDIDA");
        image.setImageResource(R.drawable.on);
        image.setVisibility(View.VISIBLE);
        new HiloConexion(myContext,myAcivity, datosEnvio: "encenderTV").start();
    }
    if(datos.equalsIgnoreCase( anotherString: "apagar televisión")){
        System.out.println("TELEVISION APAGADA");
        image.setImageResource(R.drawable.on);
        image.setVisibility(View.VISIBLE);
        new HiloConexion(myContext,myAcivity, datosEnvio: "apagarTV").start();
    }
    if(datos.equalsIgnoreCase( anotherString: "subir volumen")){
        System.out.println("SUBIR VOLUMEN");
        image.setImageResource(R.drawable.channel_up);
        image.setVisibility(View.VISIBLE);
        new HiloConexion(myContext,myAcivity, datosEnvio: "subirVolumen").start();
    }
    if(datos.equalsIgnoreCase( anotherString: "bajar volumen")){
        System.out.println("BAJAR VOLUMEN");
        image.setImageResource(R.drawable.volume_down);
        image.setVisibility(View.VISIBLE);
        new HiloConexion(myContext,myAcivity, datosEnvio: "bajarVolumen").start();
```

Figura 128. elegir() method

Next thing to do is connect to the IP address and port that we saved in SharedPreferences.



```
public class HiloConexion extends Thread {
    private Context myContext;
    private Activity myAcivity;
    private String datosEnvio;
    private String direccionIP;
    private String puerto;
    private Client newClient = new Client();
    public HiloConexion( Context myContext, Activity myActivity, String datosEnvio,String direccionIP,String puert
        this.myContext = myContext;
        this.myAcivity =myActivity;
        this.datosEnvio =datosEnvio;
        this.direccionIP = direccionIP;
        this.puerto = puerto;
    }
        boolean x=true;
        @Override
        public void run() {
            try {
                newClient.openConnection(direccionIP,Integer.parseInt(puerto));
                newClient.openStream();
                newClient.send(datosEnvio);
                newClient.closeConnection();
            } catch (Exception e) {
                e.toString();
                x=false;
            }
```

Figura 129. Give the ip and port

Also, we pass the IP address and port from the main activity to the Connection Thread. We are doing this in the constructors. Now, we recover the IP and port in that place and pass it in the ClassListen as we see below.

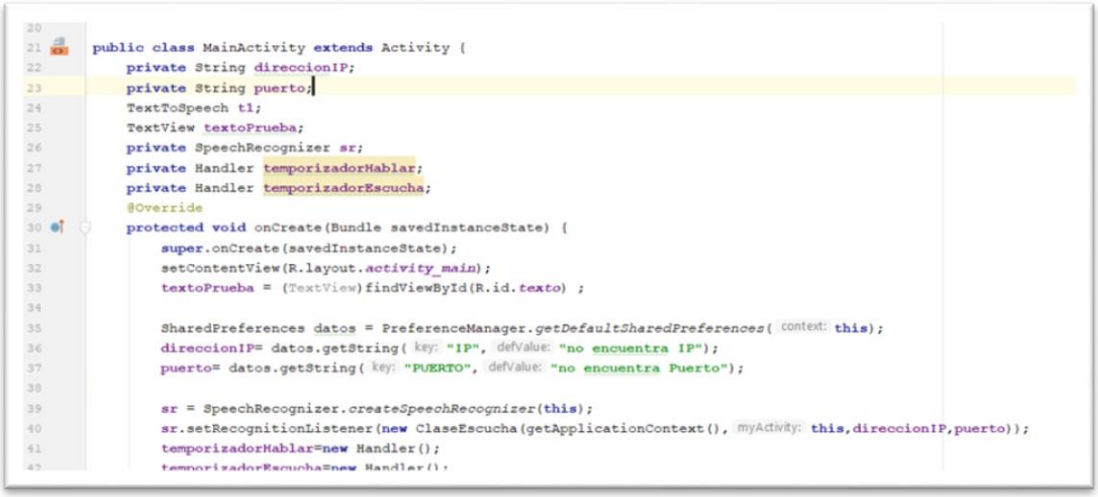


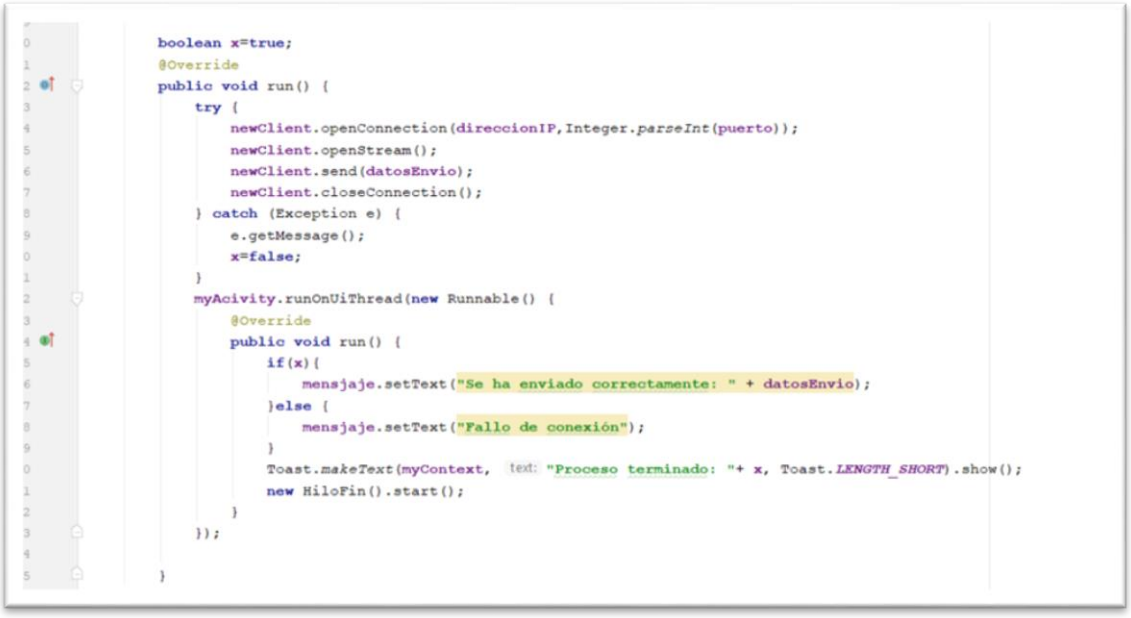
Figura 130. Pass ip and port to Clase Escucha

5.18 Errors

We finish the application every time that we launch that. If there any errors we show the error and we finish the application.

Figura 131. Finish the app

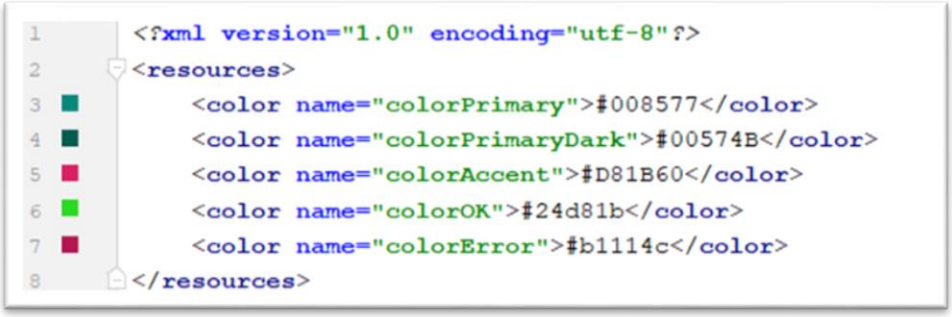
In addition, we have included a TextView that show to the user if the connection was ok or not.



```
boolean x=true;
@Override
public void run() {
    try {
        newClient.openConnection(direccionIP,Integer.parseInt(puerto));
        newClient.openStream();
        newClient.send(datosEnvio);
        newClient.closeConnection();
    } catch (Exception e) {
        e.getMessage();
        x=false;
    }
    myAcivity.runOnUiThread(new Runnable() {
        @Override
        public void run() {
            if(x){
                mensjaje.setText("Se ha enviado correctamente: " + datosEnvio);
            }else {
                mensjaje.setText("Fallo de conexión");
            }
            Toast.makeText(myContext,  text: "Proceso terminado: "+ x, Toast.LENGTH_SHORT).show();
            new HiloFin().start();
        }
    });

}
```

Figura 132. Is ok or not

We choose the colour whether it went ok or not. Green if it was correctly or red if it failed the connection. The colours are configured in the resources file.



```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">#008577</color>
    <color name="colorPrimaryDark">#00574B</color>
    <color name="colorAccent">#D81B60</color>
    <color name="colorOK">#24d81b</color>
    <color name="colorError">#b1114c</color>
</resources>
```

Figura 133. Colour File

We set the correct colour in the java code in each case.

Figura 134. Set text colour

When we save the configuration, we go out to the application with the finish() method.

Figura 135. Go out of the application

If we only do this, we will return to the main activity, so we need to pass in the Intent Class instance the following code.

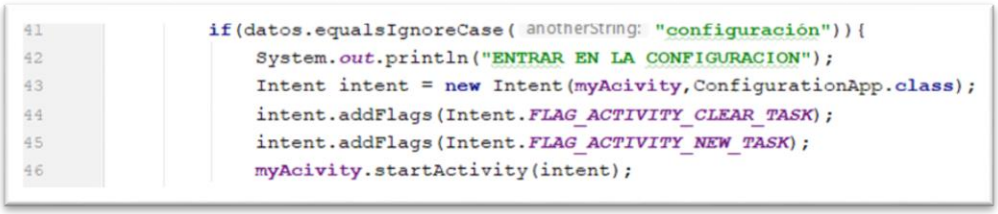


Figura 136. Close the previous activity

Also, we need to create a new thread to execute AsycnTask.

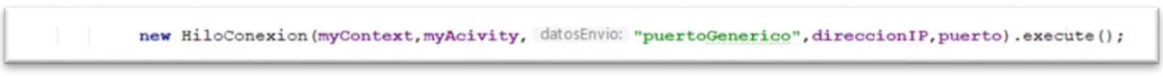


Figura 137. New Thread

In the HiloConexion class we see the doInBackground() method which helps us to connect with the server.

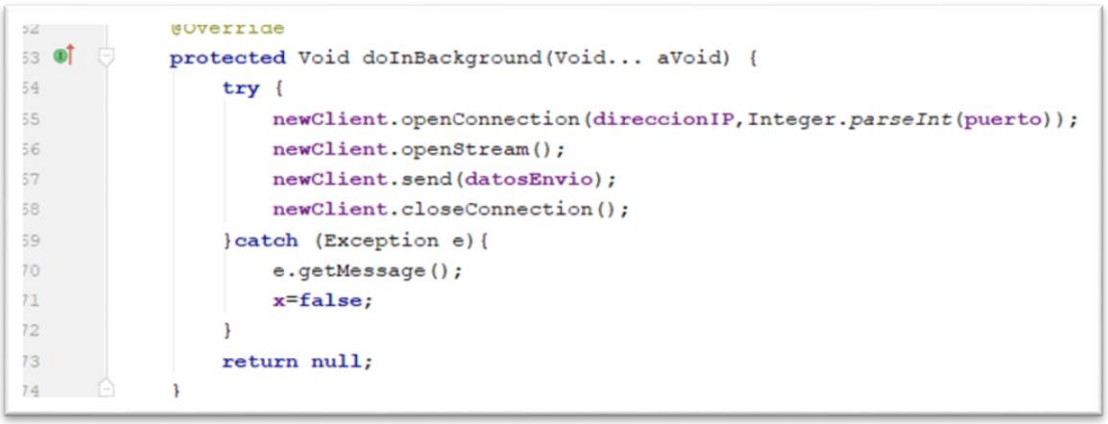


Figura 138. doInBackground() method

Once we finish running the background process we call onPostExecute() method as we see below to update the graphical interface.

Figura 139. onPostExecute() method

At the end, we call HiloFin class that is responsible for waiting a few seconds and closing the application.

```
class HiloFin extends Thread {
    @Override
    public void run() {

        myAcivity.runOnUiThread(new Runnable() {
            @Override
            public void run() {
                try{
                    Thread.sleep( millis: 3000);
                    myAcivity.finish();
                }catch (Exception e){
                    System.out.println("Fallo: " + e.getMessage());
                    myAcivity.finish();
                }
            }
        });
    }
}
```

Figura 140. HiloFin Class

To finish, we make some graphic improvements such as placing the labels better, give a more appropriate size, force the application to not be able to change orientation the device (in the Android manifest as we show below).

```
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="Interaction"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    <activity android:name=".MainActivity"
        android:configChanges="orientation"
        android:screenOrientation="portrait">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
    <activity android:name=".ConfigurationApp"
        android:configChanges="orientation"
        android:screenOrientation="portrait">
        ></activity>
</application>
```

Figura 141. Portrait view

We also put a background gradient and a title in the two layouts. To do that, we create a file called gradient.xml which we will incorporate into the drawable folder.

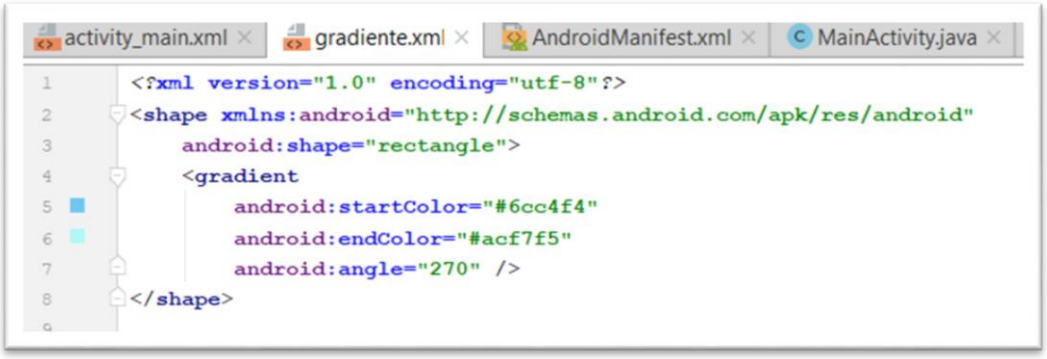


Figura 142. Gradient BackGround

Also, we put a title as we see in the next Figura 143.

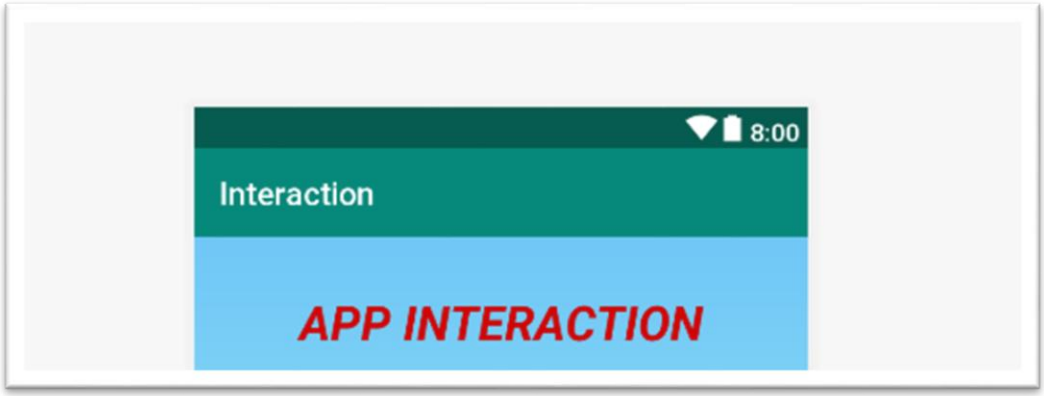


Figura 143. Title

In addition, we have established an image as a ImageView with the instructions set. In this way, user can talk to the application and we show this instructions set at the beginning.

Figura 144. Instructions set

We could see in the next Figura 145 and Figura 146 the user interface of the application.

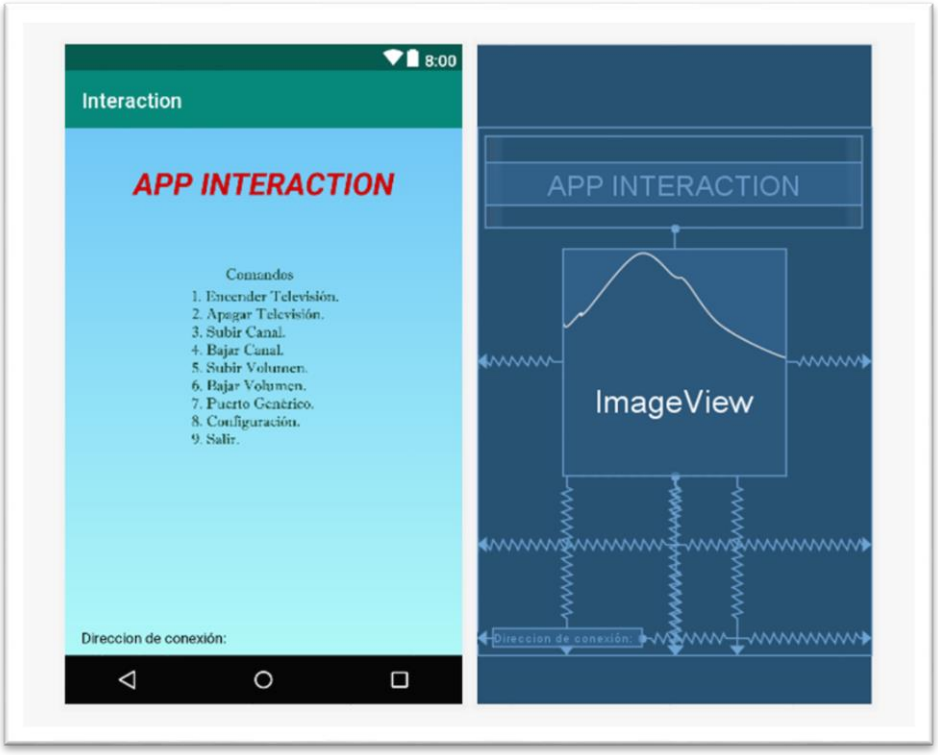


Figura 145. User interface I

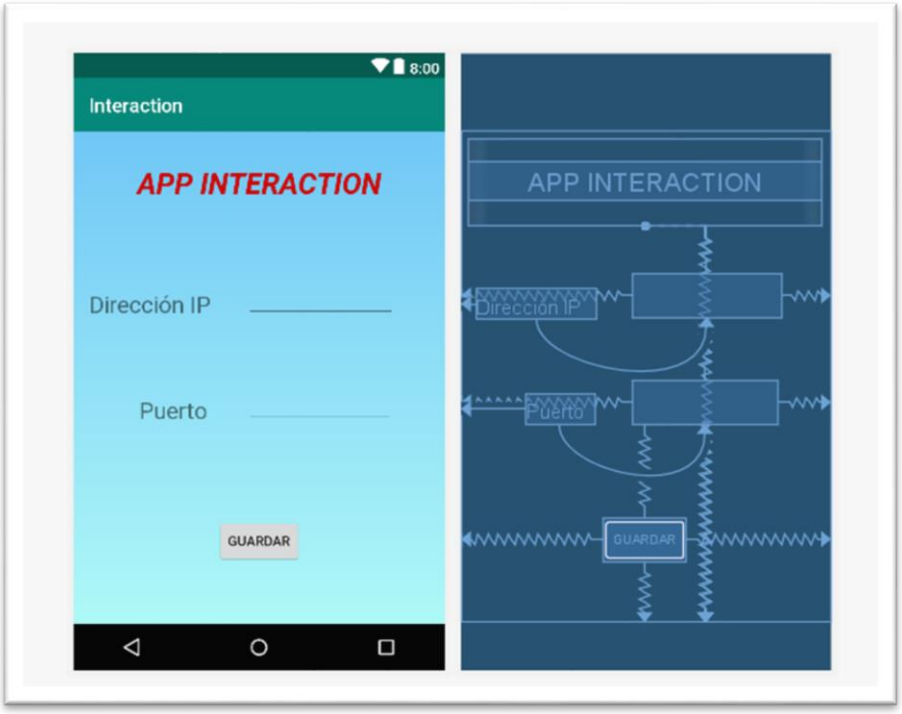


Figura 146. User Interface II

5.19 Test

To finish, all the instructions were tested on both the Android simulator and the Smartphone.

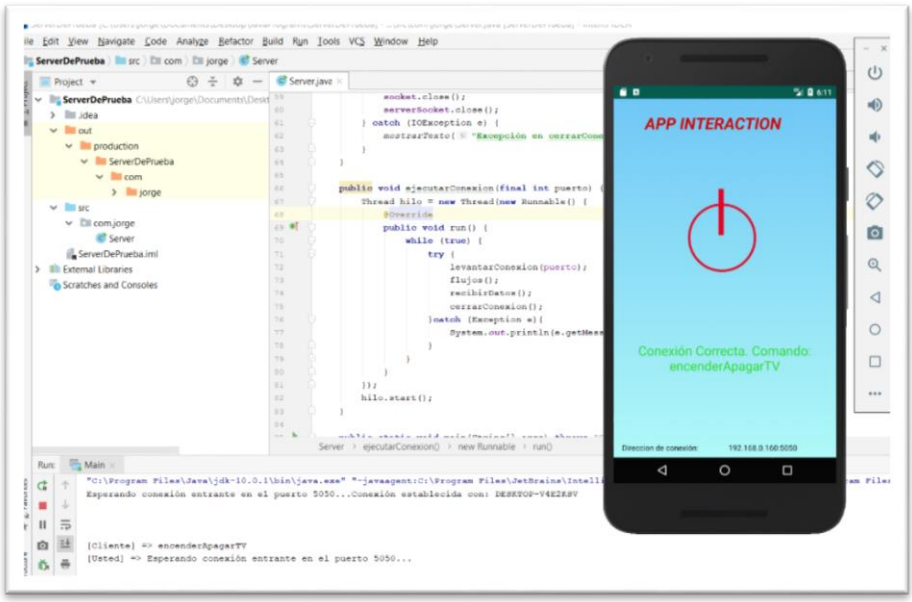


Figura 147. Test Android

Everything was fine. At the moment the server is only in a local network but it could support any internet connection.

5.19 Líneas futuras

Las posibilidades de mejora del proyecto global son ilimitadas, tanto en la incorporación de periféricos como en la simplificación de operaciones. Se deberá estar pendiente de la evolución del 5G y la pléyade de dispositivos que van a aparecer. No obstante, en el futuro más inmediato y en lo que al desarrollo de nuestra aplicación concierne, podemos destacar lo siguiente:

- Implementar unos comandos mejorados: Subir dos niveles de volumen, consultar la guía de programación, etc.
- Mejoras de seguridad. Codificar en base 64 y encriptar los comandos por si interceptan la comunicación con el fin de evitar que otras personas sepan que pueden controlar componentes electrónicos del domicilio si consiguen acceder al sistema.
- Configuración por voz para la parte de configuración IP de la aplicación móvil.
- Mejorar el responsive para Tablet y otros móviles.
- Capacidad para configurar la aplicación en multiidioma.

6. Final Conclusions

It is difficult to draw a brief conclusion from a project that has taken more than 8 months to develop it. However, we can sum up as follows:

- We have been able to achieve a project based on some kind of project management using online scheduling tools, by means of periodic milestones.
- We have developed a deeper knowledge on people disabilities and limitations, derived from a traditional concept of product design more than such people condition themselves.
- We have learnt a lot about Android and Java Fx and different ways of interacting between several platforms.

- These Apps will help some people with disabilities to do many things by themselves.
- Other student can follow the project adding new functionalities that improve some people environments.

7. REFERENCIAS BIBLIOGRÁFICAS

7.1 OMS, Discapacidades. [En línea] [Consulta: 10-noviembre-2018]. Disponible en: https://www.who.int/topics/disabilities/es/

7.2 Disiswork, Tipos de discapacidad. [En línea] [Consulta: 11-noviembre-2018]. Disponible en: https://disiswork.com/blog/tipos-de-discapacidad/

7.3 J. Antonio Giraldo, Tecnología asistiva… Ingeniería aplicada a la discapacidad. [En línea] [Consulta: 12-noviembre-2018]. Disponible en: https://perkinshumanmedia.net/brand/tecnologia-asistiva/

7.4 Más allá del ELA, IMPORTANCIA DE LAS NUEVAS TECNOLOGÍAS EN LA ELA. [En línea] [Consulta: 12-noviembre-2018]. Disponible en: http://masalladelaela.blogspot.com/2015/04/importancia-de-las-nuevas-tecnologias.html

7.5 Cruz Roja, EL DISCAPACITADO Y LA SOCIEDAD. [En línea] [Consulta: 15-noviembre-2018]. Disponible en: http://www.cruzroja.es/portal/page?_pageid=418,12398016&_dad=portal30&_schema=PORTAL30

7.6 W3C, Estándar Web del W3C define la accesibilidad para la Web de nueva generación. [En línea] [Consulta: 19-noviembre-2018]. Disponible en: https://www.w3c.es/Prensa/2008/nota081211_wcag20

7.7 Fundación Adecco, Tecnología y discapacidad. [En línea] [Consulta: 20-diciembre-2018]. Disponible en: https://fundacionadecco.org/wp-content/uploads/2016/07/Informe-Tecnolog%C3%ADa-y-Discapacidad.-Fundaci%C3%B3n-Adecco-y-Keysight2017.pdf

7.8 Alejandro Frechina, Metodología Scrum ¿Que es?. [En línea] [Consulta: 23-enero-2019]. Disponible en: https://winred.es/management/metodologia-scrum-que-es/gmx-niv116-con24594.htm

7.9 OBS Business School, Roles, Eventos y Artefactos en la metodología Scrum. [En línea] [Consulta: 23-enero-2019]. Disponible en: https://www.obs-edu.com/es/blog-investigacion/project-management/roles-eventos-y-artefactos-en-la-metodologia-scrum

7.10 José Marcos, El Congreso reconoce el derecho a votar de 100.000 discapacitados intelectuales. [En línea] [Consulta: 20-enero-2019]. Disponible en: https://elpais.com/politica/2018/10/17/actualidad/1539804297_438797.html

7.11 Javier Del Castillo, La tecnología rompe las barreras de la discapacidad. [En línea] [Consulta: 21-enero-2019]. Disponible en: https://blogthinkbig.com/la-tecnologia-rompe-las-barreras-de-la-discapacidad

7.12 Manfred Kube, De qué forma el IoT está ayudando a las personas con discapacidad. [En línea] [Consulta: 22-enero-2019]. Disponible en: https://itusers.today/forma-iot-esta-ayudando-las-personas-discapacidad/

7.13 CasaDomo, Proyecto Smart Assist con IoT para personas con discapacidad. [En línea] [Consulta: 22-enero-2019]. Disponible en: https://www.casadomo.com/2016/11/11/proyecto-smart-assist-con-iot-para-personas-con-discapacidad

7.14 Proyectos Agiles.org, Qué es SCRUM. [En línea] [Consulta: 23-enero-2019]. Disponible en: https://proyectosagiles.org/que-es-scrum/

7.15 Agilemanifesto.org, Principios del Manifiesto Ágil. [En línea] [Consulta: 23-enero-2019]. Disponible en: http://agilemanifesto.org/iso/es/principles.html

7.16 Alexander Menzinsky, ¿Qué tiene que ver Scrum con la melé de rugby?. [En línea] [Consulta: 24-enero-201]. Disponible en: http://scrum.menzinsky.com/2014/08/que-tiene-que-ver-scrum-con-la-mele-de.html

7.17 Roger Dudler, git - la guía sencilla [En línea] [Consulta: 12-Febrero-2019]. Disponible en: http://rogerdudler.github.io/git-guide/index.es.html

7.18 Facundo Goñi, 8 Razones por las que debes usar Git. [En línea] [Consulta: 25-Febrero-2019]. Disponible en: https://blog.coffeedevs.com/8-razones-para-usar-git/

7.19 INE, Panorámica de la discapacidad en España. [En línea] [Consulta: 26-Febrero-2019]. Disponible en: https://www.ine.es/revistas/cifraine/1009.pdf

7.20 Licdo. José I. Isaza G., Domótica y discapacidad. [En línea] [Consulta: 26-Febrero-2019]. Disponible en: https://revistas.utp.ac.pa/index.php/el-tecnologico/article/view/224/html

7.21 Miriam García, MVC (Modelo-Vista-Controlador): ¿qué es y para qué sirve?. [En línea] [Consulta: 13-Marzo-2019]. Disponible en: https://codingornot.com/mvc-modelo-vista-controlador-que-es-y-para-que-sirve

7.22 Código Facilito, MVC (Model, View, Controller) Explicado.. [En línea] [Consulta: 13-Marzo-2019]. Disponible en: https://codigofacilito.com/articulos/mvc-model-view-controller-explicado

7.23 Andres Almiray, Definiendo comportamiento personalizado en FXML con FXMLLoader. [En línea] [Consulta: 17-octubre-2018]. Disponible en: http://andresalmiray.com/articles/definiendo-comportamiento-personalizado-en-fxml-con-fxmlloader/

7.24 JavaTPoint, JavaFX Overview. [En línea] [Consulta: 18-octubre-2018]. Disponible en: https://www.javatpoint.com/javafx-overview

7.25 Tutor de Programacion, JavaFX Tutorial de Introducción. [En línea] [Consulta: 15-octubre-2018]. Disponible en: http://acodigo.blogspot.com/2017/02/javafx-tutorial-de-introduccion.html

7.26 Ramón Invarato, AsyncTask en Android [En línea] [Consulta: 12-Marzo-2019]. Disponible en: https://jarroba.com/asynctask-en-android/

7.27 Stack OverFlow, android.os.NetworkOnMainThreadException at android.os.StrictMode$AndroidBlockGuardPolicy.onNetwork(StrictMode.java:1145) [En línea] [Consulta: 18-Enerro-2019]. Disponible en: https://stackoverflow.com/questions/25093546/android-os-networkonmainthreadexception-at-android-os-strictmodeandroidblockgua/25093599#25093599

7.28 Ramón Invarato, Context de Android. [En línea] [Consulta: 18-Febrero-2019]. Disponible en: https://jarroba.com/context-de-android/

7.29 IntelliJ IDEA 2019.1 Help, Packaging a Module into a JAR File. [En línea] [Consulta: 19-Febrero-2019]. Disponible en: https://www.jetbrains.com/help/idea/packaging-a-module-into-a-jar-file.html

7.30 IntelliJ IDEA 2019.1 Help, Create your first Java application. [En línea] [Consulta: 19-Febrero-2019]. Disponible en: https://www.jetbrains.com/help/idea/creating-and-running-your-first-java-application.html

7.31 Oracle, Class FileReader. [En línea] [Consulta: 19-diciembre-2018]. Disponible en: https://docs.oracle.com/javase/7/docs/api/java/io/FileReader.html

7.32 Parzi Byte, Ejemplo de Sockets en Java: chat básico entre cliente y servidor. [En línea] [Consulta: 03-octubre-201]. Disponible en: https://parzibyte.me/blog/2018/02/09/sockets-java-chat-cliente-servidor/

7.33 Oracle, Getting Started with JavaFX. [En línea] [Consulta: 06-Noviembre-2018]. Disponible en: https://docs.oracle.com/javafx/2/get_started/css.htm

7.34 Ricardo Moya, Patrón Singleton en Java, con ejemplos. [En línea] [Consulta: 18-octubre-2018]. Disponible en: https://jarroba.com/patron-singleton-en-java-con-ejemplos/

7.35 Píldoras Informáticas, Curso Android. [En línea] [Consulta: 19-octubre-2018]. Disponible en: https://www.youtube.com/watch?v=pdYkmCcQFd8

7.36 Android Developer Documentation, RecognitionListener. [En línea] [Consulta: 12-Marzo-2019]. Disponible en: https://developer.android.com/reference/android/speech/RecognitionListener

7.37 Android Developer Documentation, SpeechRecognizer. [En línea] [Consulta: 13-Abril-2019]. Disponible en: https://developer.android.com/reference/android/speech/SpeechRecognizer

7.38 Android Developer Documentation, TextToSpeech. [En línea] [Consulta: 25-Abril-2019]. Disponible en: https://developer.android.com/reference/android/speech/tts/TextToSpeech

7.39 Android Developer Documentation, Understand the Activity Lifecycle. [En línea] [Consulta: 07-Marzo-2019]. Disponible en: https://developer.android.com/guide/components/activities/activity-lifecycle

7.40 Android Developer Documentation, Activity [En línea] [Consulta: 28-Marzo-2019]. Disponible en: https://developer.android.com/reference/android/app/Activity

7.41 Android Developer Documentation, AppCompatActivity. [En línea] [Consulta: 15-Febrero-2019]. Disponible en: https://developer.android.com/reference/android/support/v7/app/AppCompatActivity

7.42 Plena Inclusión, *El INE ofrecerá en 2019 nuevos datos sobre la discapacidad en España.* [En línea] [Consulta: 3-febrero-2019]. Disponible en: https://www.plenainclusion.org/informate/actualidad/noticias/2019/el-ine-ofrecera-en-2019-nuevos-datos-sobre-la-discapacidad-en

7.43 Imserso, *Base Estatal de datos de personas con discapacidad*. [En línea] [Consulta: 3-febrero-2019]. Disponible en: https://www.who.int/topics/disabilities/es/

7.44 ELA México, *ETRAN Presentación*. [En línea] [Consulta: 3-marzo-2019]. Disponible en: https://sites.google.com/site/vicortega2/etran

7.45 Aula Abierta Arasaac, *TABLEROS DE COMUNICACIÓN CON PICTOGRAMAS*. [En línea] [Consulta: 3-marzo-2019]. Disponible en: https://aulaabierta.arasaac.org/materiales-caa-tableros-de-comunicacion

7.46 MegaBEE, *Welcome to MegaBee*. [En línea][Consulta: 3-marzo-2019]. Disponible en: http://www.megabee.net/img/black.jpg

7.47 Network 1 consulting, *Smartphones & Tablets*.[En línea] [Consulta: 3-marzo-2019]. Disponible en: https://network1consulting.com/it-services/smartphones-tablets/

7.48 Irisbond, *Productos*. [En línea] [Consulta: 3-marzo-2019]. Disponible en: https://www.irisbond.com/productos/irisbond-duo

7.49 Tecnología Adaptada, *Head Wand*. [En línea] [Consulta: 3-marzo-2019]. Disponible en: http://tecnologiacaam.blogspot.com/p/head-wand.html

7.50 Ingenium, *LIFEWARE INTEGRA*. [En línea] [Consulta: 3-marzo-2019]. Disponible en: http://ingenium.usm.cl/noticias/lifewareintegra/

7.51 Comunidad IEBS, *Metodologías Ágiles*. [En línea] [Consulta: 3-marzo-2019]. Disponible en: https://comunidad.iebschool.com/metodologiasagiles/files/2015/05/metodologias-agiles.jpg

7.52 Alejandro Frechina, Metodología Scrum ¿Que es?. [Consulta: 3-marzo-2019]. Disponible en: https://winred.es/uploads/contenidos_usrs/originales/2566824_013_scrum_process_20180618180200.jpg

8. ANEXOS

8.1 Clase Data (Java Application)

```
package com.jflores.visualApp.data;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;


public class Data {
    //Referencia del único objeto que vamos a crear
    private static Data instance;
    //Nombre Fichero de configuración donde están los datos, esos
datos luego se almacenarán en item
    private static final String FILENAME =
"C:\\Users\\jorge\\Documents\\Desktop\\Trabajo fin master
Apps\\myActionApp-
java\\MyAppAction\\src\\com\\jflores\\visualApp\\conf\\properties.co
nf";
    //Lista de Item(shortDescription,details) para guardarlos en una
lista.
    private static ArrayList<Item> listOfItem = new ArrayList();

    private static boolean flag = false;

    public static Data getInstance() {
        if (instance==null){
            // Se crea la instancia si no existe y además se va a
buscar sólo una vez  los datos al fichero
            instance = new Data();
```

```
            try {
                BufferedReader br = new BufferedReader(new
FileReader(FILENAME));
                String input= br.readLine();
                while (input != null) {
                    //si podemos, deberíamos cambiar por otro
carácter
                    String[] itemPieces = input.split("=");
                    //capturamos cada dato en un array
                    String shortDescription = itemPieces[0];
                    String details = itemPieces[1];
                    //Creamos el Ítem y lo agregamos a una lista de
Ítems
                    Item newItem = new Item(shortDescription,
details);
                    listOfItem.add(newItem);
                    input= br.readLine();
                }

            } catch (IOException e){
                //En el caso que hubiera un error de lectura avisa
con una bandera
                System.out.println(e.getMessage());
                flag = true;
            }

        }
        //si este método se llama por segunda vez devuelve la
primera instancia y no crea otro objeto
        return instance;
    }
// Singleton, constructor privado para que no se pueda acceder a él
(clase con una única instancia)
    //aunque no se pueda acceder cuando se crea la instancia aparece
inizializado.
    private Data() {
        System.out.println("Inicializado");
    }

    //método para devolver los Ítems (datos del archivo de conf y
llevamos a un arrayList)
    public ArrayList<Item> getItems() {
        return listOfItem;
    }

    //Comprueba si la bandera es falsa o verdad para controlar el
error de lectura del fichero
    public static boolean isFlag() {
        return flag;
    }
}
```

8.2 Clase Item (Java Application)

```
package com.jflores.visualApp.data;


public class Item {
```

```
    private String shortDescription;
    private String details;

    //Constructor de la clase
    public Item(String shortDescription, String details) {
        this.shortDescription = shortDescription;
        this.details = details;
    }

    //Getters y Setters
    public String getShortDescription() {
        return shortDescription;
    }

    public void setShortDescription(String shortDescription) {
        this.shortDescription = shortDescription;
    }

    public String getDetails() {
        return details;
    }

    public void setDetails(String details) {
        this.details = details;
    }

    //Sobreescribe el método toString para que si imprimimos un Ítem
devuelva la descripción.
    @Override
    public String toString() {
        return shortDescription;
    }
}
```

8.3 Clase Client (Java Application)

```
//esta clase no se usa solo para ver mejor la estructuración

package com.jflores.visualApp;


import java.io.DataOutputStream;
import java.io.IOException;
import java.net.Socket;

public class Client {

    private static Socket socket;
    private static DataOutputStream bufferOut = null;



    public void openConnection(String ip, int puerto)  throws
Exception{
        try {
            socket = new Socket(ip, puerto);
            System.out.println("Conectado a :" +
socket.getInetAddress().getHostName());
        } catch (Exception e) {
```

```
            System.out.println("Excepción al levantar conexión: " +
e.getMessage());
        }
    }


    public void openStream() throws Exception{
        try {
            bufferOut = new
DataOutputStream(socket.getOutputStream());
            bufferOut.flush();
        } catch (IOException e) {
            System.out.println("Error en la apertura de flujos");
        }
    }

    public void send(String s) throws Exception{
        try {
            bufferOut.writeUTF(s);
            bufferOut.flush();
        } catch (IOException e) {
            System.out.println("IOException a enviar");
        }
    }

    public void closeConnection() throws Exception{
        try {
            bufferOut.close();
            socket.close();
            //Lo ponemos a null para que vuelva a crearlo de nuevo
cuando se vuelva a llamar y controlar bien las excepciones
            socket=null;
            System.out.println("Conexión terminada");
        } catch (IOException e) {
            System.out.println("IOException on cerrarConexion()");
        }
    }

}
```

8.4 Clase Controller (Java Application)

```
package com.jflores.visualApp;

import com.jflores.visualApp.data.Data;
import com.jflores.visualApp.data.Item;
import javafx.concurrent.Task;
import javafx.fxml.FXML;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.paint.Color;

import java.util.ArrayList;

public class Controller {
```

```
    private static final String TV_OFF =
"com/jflores/visualApp/img/off.png";
    private static final String TV_ON =
"com/jflores/visualApp/img/on.png";
    private static final String UP_CHANNEL_ON =
"com/jflores/visualApp/img/up-channel_on.png";
    private static final String UP_CHANNEL_OFF =
"com/jflores/visualApp/img/up-channel_off.png";
    private static final String DOWN_CHANNEL_ON =
"com/jflores/visualApp/img/down-channel on.png";
    private static final String DOWN_CHANNEL_OFF =
"com/jflores/visualApp/img/down-channel_off.png";
    private static final String UP_VOLUME_ON =
"com/jflores/visualApp/img/up_volume_on.png";
    private static final String UP_VOLUME_OFF =
"com/jflores/visualApp/img/up_volume_off.png";
    private static final String DOWN_VOLUME_ON =
"com/jflores/visualApp/img/down-volume-on.png";
    private static final String DOWN_VOLUME_OFF =
"com/jflores/visualApp/img/down-volume-off.png";
    private static final String GENERIC_PORT_ON =
"com/jflores/visualApp/img/generic-on.png";
    private static final String GENERIC_PORT_OFF =
"com/jflores/visualApp/img/generic-off.png";

    private static final Integer LECTURE_IP = 0;
    private static final Integer LECTURE_PORT = 1;
    private static final Integer LECTURE_TV_ON_OFF = 2;
    private static final Integer LECTURE_TV_UP_VOLUME = 3;
    private static final Integer LECTURE_DOWN_VOLUME = 4;
    private static final Integer LECTURE_UP_CHANNEL = 5;
    private static final Integer LECTURE_DOWN_CHANNEL = 6;
    private static final Integer LECTURE_GEERIC_PORT = 7;

    private static final Integer WIDTH = 240;
    private static final Integer HEIGHT = 240;

    private static final Integer THREAD_SLEEP = 2000;

    //Variables de la clase Controller los cuales son elementos que
se usan para la lógica o clases para relacionar
    // Su parte lógica con la interfaz. Ej: botón. Para relacionarlo
se usa la notación @FXML = id de la view
    private ArrayList<Item> items;
    @FXML
    private Button buttonOnTv;
    @FXML
    private Button buttonUpChannel;
    @FXML
    private Button buttonDownChannel;
    @FXML
    private Button buttonUpVolume;
    @FXML
    private Button buttonDownVolume;
    @FXML
    private Button buttonGenericPort;
    @FXML
    private Label label;

    Client newClient = new Client();
```

```
    public void initialize() {

        //Llamamos a la instancia para cargar los datos cuando se
inicializa la pantalla
        Data.getInstance();

        //Control de errores en el caso de que no encuentre el
archivo de configuración properties.conf
        //Es simplemente un aviso en un Lbal, cambiar a otro lugar
cuando se ejecute
        if (Data.isFlag()){
            label.setText("Error a la hora de encontrar el archivo
de configuración");
        }
        setIcons();



        //Guardamos los datos e items
        items = Data.getInstance().getItems();
//Pintamos los objetos que ha encontrado para depurar.
        for (int i=0;i<items.size();i++){
            System.out.println(items.get(i).getDetails());
        }


    }

    public void setIcons(){
        //Crear una imagen de la carpeta donde está la imagen
        Image iconTVOff = new
Image(TV_OFF,WIDTH,HEIGHT,false,false);
        // se establece la imagen del botón con el método
setGraphic, se debe usar como parámetro una subclase de Node que es
ImageView
        //public class ImageView
        //extends Node
        //The ImageView is a Node used for painting images loaded
with Image class.
        buttonOnTv.setGraphic(new ImageView(iconTVOff));

        Image iconUpChannel = new
Image(UP_CHANNEL_OFF,WIDTH,HEIGHT,false,false);
        buttonUpChannel.setGraphic(new ImageView(iconUpChannel));

        Image iconDownChannel = new
Image(DOWN_CHANNEL_OFF,WIDTH,HEIGHT,false,false);
        buttonDownChannel.setGraphic(new
ImageView(iconDownChannel));

        Image iconDownVolume = new
Image(DOWN_VOLUME_OFF,WIDTH,HEIGHT,false,false);
        buttonDownVolume.setGraphic(new ImageView(iconDownVolume));

        Image iconUpVolume = new
Image(UP_VOLUME_OFF,WIDTH,HEIGHT,false,false);
        buttonUpVolume.setGraphic(new ImageView(iconUpVolume));
```

```
        Image iconGenericPort = new
Image(GENERIC_PORT_OFF,WIDTH,HEIGHT,false,false);
        buttonGenericPort.setGraphic(new
ImageView(iconGenericPort));

    }

//No usamos threads para no mandar más datos a la vez.



    //Método para manejar el encendido de la TV cuando se hace click
sobre el botón On/Off.
    @FXML
    public void handleClickButtonOn(){
        //Como no es posible mandar un método con parámetros a
través de la vista o al menos no lo encontré en la doc
        //si que vi la forma de encapsular un método y pasar hay
parámetros de forma muy sencilla
        //Con el fin de no repetir código y tenerlo todo más
organizado
        encapsuleMethod(
                TV_OFF,
                TV_ON,
                buttonOnTv,
                LECTURE_TV_ON_OFF
        );


    }


    @FXML
    public void handleClickButtonUpChannel(){
        encapsuleMethod(
                UP_CHANNEL_OFF,
                UP_CHANNEL_ON,
                buttonUpChannel,
                LECTURE_UP_CHANNEL
        );
    }

    @FXML
    public void handleClickButtonDownChannel(){
        encapsuleMethod(
                DOWN_CHANNEL_OFF,
                DOWN_CHANNEL_ON,
                buttonDownChannel,
                LECTURE_DOWN_CHANNEL
        );
    }

    @FXML
    public void handleClickButtonUpVolume(){
        encapsuleMethod(
                UP_VOLUME_OFF,
                UP_VOLUME_ON,
                buttonUpVolume,
                LECTURE_TV_UP_VOLUME
        );
```

```
    }

    @FXML
    public void handleClickButtonDownVolume(){
        encapsuleMethod(
                DOWN_VOLUME_OFF,
                DOWN_VOLUME_ON,
                buttonDownVolume,
                LECTURE_DOWN_VOLUME
        );
    }

    @FXML
    public void handleClickButtonGenericPort(){
        encapsuleMethod(
                GENERIC_PORT_OFF,
                GENERIC_PORT_ON,
                buttonGenericPort,
                LECTURE_GEERIC_PORT
        );
    }


public void encapsuleMethod(String imagenApagada,String
imagenEncendida, Button botonPulsado, Integer datoAEnviar){

//    label.setText("");

    //imágenes para hacer el cambio y mostrar que hay un cambio y se
esta enviando el mensaje.
    Image iconApagado = new
Image(imagenApagada,WIDTH,HEIGHT,false,false);
    Image iconEncendido = new
Image(imagenEncendida,WIDTH,HEIGHT,false,false);


//https://docs.oracle.com/javase/8/javafx/api/javafx/concurrent/Task
.html
    Task<Void> myTask = new Task<Void>() {
        @Override
        protected Void call() throws Exception {
            try {
                Thread.sleep(THREAD_SLEEP);
            } catch (Exception e) {}
            return null;

        }

        @Override
        protected void succeeded() {
            botonPulsado.setGraphic(new ImageView(iconApagado));
            botonPulsado.setDisable(false);
        }

        @Override
        protected void running() {
            botonPulsado.setGraphic(new ImageView(iconEncendido));
            botonPulsado.setDisable(true);
        }
```

```
        };


        Task<Void> myTask2 = new Task<Void>() {
            boolean x=false;
            @Override
            protected Void call() throws Exception {
                try { //subcestible de ser comentado

newClient.openConnection(items.get(LECTURE_IP).getDetails(),Integer.
parseInt(items.get(LECTURE_PORT).getDetails()));
                    newClient.openStream();
                    newClient.send(items.get(datoAEnviar).getDetails());
                    newClient.closeConnection();
                }catch (Exception e){
                    x=true;
                    System.out.println("Falla conexión"); //capturar el
throws
                }
                return null;

            }

            @Override
            protected void succeeded() {
                if(!x){
                    label.setText("Orden:  " +
items.get(datoAEnviar).getShortDescription() + "  enviada.");
                    label.setTextFill(Color.color(0.2,1,0.1));
                }else{
                    label.setText("Error al conectar con el uC");
                    label.setTextFill(Color.color(1,0,0));
                }

            }
        };


        new Thread(myTask).start();
        new Thread(myTask2).start();


    }

    }
```

8.5 Clase Main (Java Application)

```
package com.jflores.visualApp;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.image.Image;
import javafx.stage.Stage;

public class Main extends Application {
```

```
    private static final String VIEW = "view.fxml";
    private static final String TITLE = "Visual Action App";
    private static final String ICON =
"com/jflores/visualApp/img/inicio.png";
    private static final Integer WIDTH = 920;
    private static final Integer HEIGHT = 600;

    //sobrescribimos el método start para cargar la interfaz
    @Override
    public void start(Stage primaryStage) throws Exception{
        //Cargar el fxml de la vista con javafx
        Parent root = FXMLLoader.load(getClass().getResource(VIEW));
        //establecer el nombre de la app
        primaryStage.setTitle(TITLE);
        //Establecer icono de la interfaz y agregarlo
        primaryStage.getIcons().add(new Image(ICON));
        //Establecer el tamaño de la ventana
        primaryStage.setScene(new Scene(root, WIDTH, HEIGHT));
        //Mostrar  el escenario (ventana)
        primaryStage.show();

    }


//ejecutamos el método lunch , buscar en doc FX.
    public static void main(String[] args) {
        launch(args);
    }

}
```

8.6 View FXML (Java Application)

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.geometry.Insets?>
<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.layout.GridPane?>

<!--Comentarios,
Archivo FXML con esa notación para la interfaz gráfica, separando la
parte gráfica (View) de la parte de la lógica (Controller)
-->
<!--Usamos Grid Pane como layout de la app, modelo de rejilla-->

<GridPane alignment="TOP_LEFT" hgap="10" prefHeight="218.0"
prefWidth="306.0" vgap="10" xmlns:fx="http://javafx.com/fxml/1"
xmlns="http://javafx.com/javafx/8.0.121"
fx:controller="com.jflores.visualApp.Controller">
    <padding><Insets bottom="10" left="25" right="25" top="25"
/></padding>
    <!--Añadimos un boton con un id para relacionarlo con el
controller, al igual que el evento que le damos
    al boton (onMouse) para relazionar ese evento al metodo del
controller hanldeClick-->
```

```
    <Button fx:id="buttonOnTv" onMouseClicked="#handleClickButtonOn"
GridPane.columnIndex="1" GridPane.columnSpan="2"
GridPane.rowIndex="1" />
    <Button fx:id="buttonUpChannel"
onMouseClicked="#handleClickButtonUpChannel"
GridPane.columnIndex="5" GridPane.columnSpan="2"
GridPane.rowIndex="1" />
    <Button fx:id="buttonDownChannel"
onMouseClicked="#handleClickButtonDownChannel"
GridPane.columnIndex="10" GridPane.columnSpan="2"
GridPane.rowIndex="1" />
    <Button fx:id="buttonUpVolume"
onMouseClicked="#handleClickButtonUpVolume"
GridPane.columnIndex="1" GridPane.columnSpan="2"
GridPane.rowIndex="2" />
    <Button fx:id="buttonDownVolume"
onMouseClicked="#handleClickButtonDownVolume"
GridPane.columnIndex="5" GridPane.columnSpan="2"
GridPane.rowIndex="2" />
    <Button fx:id="buttonGenericPort"
onMouseClicked="#handleClickButtonGenericPort"
GridPane.columnIndex="10" GridPane.columnSpan="2"
GridPane.rowIndex="2" />
    <!--Usamos d emomento labels para depurar-->

    <Label text="Datos Salida: " GridPane.columnIndex="1"
GridPane.columnSpan="2" GridPane.rowIndex="3">
    </Label>
    <Label fx:id="label" GridPane.columnIndex="1"
GridPane.columnSpan="2" GridPane.rowIndex="4">
    </Label>

</GridPane>
```

8.7 Properties.conf (Java Application)

```
Ip del uControlador primario=192.168.0.161
Puerto de Conexion=5050
1.Encender-Apagar Tv=encenderApagarTV
2.Subir Volumen Tv=subirVolumenTV
3.Bajar Volumen Tv=bajarVolumenTV
4.Subir Canal Tv=subirCanalTV
5.Bajar Canal Tv=bajarCanalTV
6.Puerto Generico=puertoGenerico
```

8.8 Clase Server (Java Server Test)

```
package com.jorge;

import java.io.DataInputStream;
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;


public class Server {
```

```
    private Socket socket;
    private ServerSocket serverSocket;
    private DataInputStream bufferDeEntrada = null;



    public void levantarConexion(int puerto) {
        try {
            serverSocket = new ServerSocket(puerto);
            mostrarTexto("Esperando conexión entrante en el puerto "
+ Integer.toString(puerto) + "...");
            socket = serverSocket.accept();
            mostrarTexto("Conexión establecida con: " +
socket.getInetAddress().getHostName() + "\n\n\n");
        } catch (Exception e) {
            mostrarTexto("Error en levantarConexion(): " +
e.toString());
            System.exit(0);
        }
    }
    public void flujos() {
        try {
            bufferDeEntrada = new
DataInputStream(socket.getInputStream());
        } catch (IOException e) {
            mostrarTexto("Error en la apertura de flujos");
        }
    }

    public void recibirDatos() {
        String st = "";
        try {
            do {
                st = (String) bufferDeEntrada.readUTF();
                mostrarTexto("\n[Cliente] => " + st);
                System.out.print("\n[Usted] => ");
            } while (!st.equals("SALIR"));
        } catch (IOException e) {
            cerrarConexion();
        }
    }


    public static void mostrarTexto(String s) {
        System.out.print(s);
    }


    public void cerrarConexion() {
        try {
            bufferDeEntrada.close();
            socket.close();
            serverSocket.close();
        } catch (IOException e) {
            mostrarTexto("Excepción en cerrarConexion(): " +
e.getMessage());
        }
    }
```

```
    public void ejecutarConexion(final int puerto) {
        Thread hilo = new Thread(new Runnable() {
            @Override
            public void run() {
                while (true) {
                    try {
                        levantarConexion(puerto);
                        flujos();
                        recibirDatos();
                        cerrarConexion();
                    }catch (Exception e){
                        System.out.println(e.getMessage());
                    }
                }
            }
        });
        hilo.start();
    }

    public static void main(String[] args) throws IOException {
        Server s = new Server();
        String puerto = "5050";
        s.ejecutarConexion(Integer.parseInt(puerto));
    }
}
```

8.9 Android Manifest (Android App)

```
<?xml version="1.0" encoding="utf-8"?>
<!--Archivo manifest el cual
proporciona información esencial sobre la aplicación al sistema
Android,
 información que el sistema debe tener para poder ejecutar el código
de la app.
 Entre otras cosas, el archivo de manifiesto hace lo siguiente:
 -  Nombra el paquete de Java para la aplicación.
 -  Describe los componentes de la aplicación, como las actividades,
    los servicios, los receptores de mensajes y los proveedores de
contenido que la integran.
 -  Determina los procesos que alojan a los componentes de la
aplicación.
 -  Declara los permisos debe tener la aplicación para acceder a las
partes
    Protegidas de una API e interactuar con otras aplicaciones.
 -  etc..
-->
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.flores.jorge.interaction">
    <!--para dar permisos de audio y de internet-->
    <uses-permission android:name="android.permission.RECORD_AUDIO"
/>
    <uses-permission android:name="android.permission.INTERNET" >
    </uses-permission>

    <uses-feature android:name="android.hardware.wifi" />
```

```
    <uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE"></uses-
permission>
    <uses-permission
android:name="android.permission.ACCESS_WIFI_STATE"></uses-
permission>

    <!--Indica nombre de la app, icono, la orientación con las
actividades -->
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity"
            android:configChanges="orientation"
            android:screenOrientation="portrait">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category
android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".ConfigurationApp"
            android:configChanges="orientation"
            android:screenOrientation="portrait">
            ></activity>
    </application>

</manifest>
```

8.10 Clase Escucha (Android App)

```
package com.flores.jorge.interaction;

import android.app.Activity;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.speech.RecognitionListener;
import android.speech.SpeechRecognizer;
import android.util.Log;
import android.view.View;
import android.widget.ImageView;
import android.widget.Toast;

import java.util.ArrayList;

//Clase usada como listener para hablar
class ClaseEscucha implements RecognitionListener {
    //Variables de la clase
    private static final String TAG = "MyActivity";
    private Context myContext;
    private Activity myAcivity;
    ImageView image;
```

```
    private String direccionIP;
    private String puerto;

    //Costructor de la clase al que le pasamos un contexto, una
activiad una ddireccion y un puerto
    public ClaseEscucha(Context myContext, Activity
myActivity,String direccionIP,String puerto){
        this.myContext = myContext;
        this.myAcivity =myActivity;
        this.direccionIP = direccionIP;
        this.puerto = puerto;
        image = (ImageView)myAcivity.findViewById(R.id.imageView);
    }

    //Metodos pertenecientes a la interfaz RecognitionListener
    //Ests metodos los dejamos con un log para depuración
    //Sobreescribimos este metodo para que nos muestre un Toast al
empezar a hablar.
    @Override
    public void onReadyForSpeech(Bundle params) {
        Log.d(TAG, "onReadyForSpeech");
        Toast.makeText(myContext, "EMPIEZA A
HABLAR",Toast.LENGTH_SHORT).show();
    }

    @Override
    public void onBeginningOfSpeech() {
        Log.d(TAG, "onBeginningOfSpeech");
    }

    @Override
    public void onRmsChanged(float rmsdB) {
        Log.d(TAG, "onRmsChanged");
    }

    @Override
    public void onBufferReceived(byte[] buffer) {
        Log.d(TAG, "onBufferReceived");
    }

    //Cuando deja de escuchar mostramos un Toast
    @Override
    public void onEndOfSpeech() {
        Log.d(TAG, "onEndofSpeech");
        Toast.makeText(myContext, "FIN
ESCUCHA",Toast.LENGTH_SHORT).show();
    }

    //En el caso de que ocurra un error  mostramos una imagen de
error.
    //Tambien creamos un nuevo hilo de conexion aprovechando para
que se cierre la app.
    @Override
    public void onError(int error) {
        Log.d(TAG,  "error " +  error);
        image.setImageResource(R.drawable.error);
        image.setVisibility(View.VISIBLE);
        new HiloConexion(myContext,myAcivity,"Not Voice
Detection",null,null).execute();
    }
```

```
    //Mostramos la elección de lo que se dice. Al ser un array en el
cual esta ordenador por probabilidad
    // Los resultados elegimos la opcion 0 la cual es la que mas se
parece.
    //En este metodo ejecutamos con elegirDatos(datos) que hacer con
lo que que hemos hablado.
    @Override
    public void onResults(Bundle results) {
        String str = new String();
        Log.d(TAG, "onResults " + results);
        ArrayList data =
results.getStringArrayList(SpeechRecognizer.RESULTS_RECOGNITION);
        for (int i = 0; i < data.size(); i++)
        {
            Log.d(TAG, "result " + data.get(i));
            str += data.get(i);
        }
        final String datos = data.get(0).toString();
        System.out.println("El resultado final es: " + datos);
        elegir(datos);

    }

    @Override
    public void onPartialResults(Bundle partialResults) {
        Log.d(TAG, "onPartialResults");
    }

    @Override
    public void onEvent(int eventType, Bundle params) {
        Log.d(TAG, "onEvent " + eventType);
    }

    //Metodo de elección seguín lo que digamos
    public void elegir(String datos){
        //Si decimos salir salimos de la app.
        if(datos.equalsIgnoreCase("salir")){
            myAcivity.finish();
        }
        //Si elegimos encender la tv cambiamos el icono al de
encender tv.
        //Creamos una nueva instancia del hilo de conexion donde
pasamos los datos necesarios
        // Para realizar la conexion, los datos a enviar y a donde.
        //Por ultimo salimos de la app desde el hilo de conexión
anque fuera bien.
        if(datos.equalsIgnoreCase("encender televisión")){
            System.out.println("TELEVISION ENCENDIDA");
            image.setImageResource(R.drawable.on);
            image.setVisibility(View.VISIBLE);
            new
HiloConexion(myContext,myAcivity,"encenderApagarTV",direccionIP,puer
to).execute();
        }
        else if(datos.equalsIgnoreCase("apagar televisión")){
            System.out.println("TELEVISION APAGADA");
            image.setImageResource(R.drawable.on);
            image.setVisibility(View.VISIBLE);
```

```
            new
HiloConexion(myContext,myAcivity,"encenderApagarTV",direccionIP,puer
to).execute();
        }
        else if(datos.equalsIgnoreCase("subir volumen")){
            System.out.println("SUBIR VOLUMEN");
            image.setImageResource(R.drawable.channel_up);
            image.setVisibility(View.VISIBLE);
            new
HiloConexion(myContext,myAcivity,"subirVolumenTV",direccionIP,puerto
).execute();
        }
        else if(datos.equalsIgnoreCase("bajar volumen")){
            System.out.println("BAJAR VOLUMEN");
            image.setImageResource(R.drawable.volume_down);
            image.setVisibility(View.VISIBLE);
            new
HiloConexion(myContext,myAcivity,"bajarVolumenTV",direccionIP,puerto
).execute();
        }
        else if(datos.equalsIgnoreCase("subir canal")){
            System.out.println("SUBIR CANAL");
            image.setImageResource(R.drawable.channel_up);
            image.setVisibility(View.VISIBLE);
            new
HiloConexion(myContext,myAcivity,"subirCanalTV",direccionIP,puerto).
execute();
        }
        else if(datos.equalsIgnoreCase("bajar canal")){
            System.out.println("BAJAR CANAL");
            image.setImageResource(R.drawable.down_channel);
            image.setVisibility(View.VISIBLE);
            new
HiloConexion(myContext,myAcivity,"bajarCanalTV",direccionIP,puerto).
execute();
        }
        else if(datos.equalsIgnoreCase("puerto genérico")){
            System.out.println("PUERTO GENERICO");
            image.setImageResource(R.drawable.generic);
            image.setVisibility(View.VISIBLE);
            new
HiloConexion(myContext,myAcivity,"puertoGenerico",direccionIP,puerto
).execute();
        }
        //En caso de decir configuración creamos un intent para ir a
la actividad de configuración
        // Indicamos al intent que cerramos la otra actividad (la
principal)
        //En caso de fallo reusamos el hilo de conexion para dar un
error y salir de la app.
        else if(datos.equalsIgnoreCase("configuración")){
            System.out.println("ENTRAR EN LA CONFIGURACION");
            Intent intent = new
Intent(myAcivity,ConfigurationApp.class);
            intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TASK);
            intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
            myAcivity.startActivity(intent);
        } else{
            new HiloConexion(myContext,myAcivity,"Not
Found",null,null).execute();
```

```
        }
    }

}
```

8.11 Clase Client (Android App)

```
package com.flores.jorge.interaction;


import java.io.DataOutputStream;
import java.io.IOException;
import java.net.InetSocketAddress;
import java.net.Socket;

//Clase reutilzada para conectarse con el microcontrolador
public class Client {

    private static Socket socket;
    private static DataOutputStream bufferOut = null;


    //Abrimos conexión
    public void openConnection(String ip, int puerto)  throws
Exception{
        try {
            socket = new Socket();
            //Establecemos un intento de conexion segun un tiempo
para que nos responda rapido si falla o no
            // al establecer la conexión.
            socket.connect(new InetSocketAddress(ip,puerto),300);
            System.out.println("Conectado a :" +
socket.getInetAddress().getHostName());
        } catch (Exception e) {
            System.out.println("Excepción al levantar conexión: " +
e.getMessage());
        }
    }

    //Abrimos flujo
    public void openStream() throws Exception{
        try {
            bufferOut = new
DataOutputStream(socket.getOutputStream());
            bufferOut.flush();
        } catch (IOException e) {
            System.out.println("Error en la apertura de flujos");
        }
    }

    //Enviamos datos
    public void send(String s) throws Exception{
        try {
            bufferOut.writeUTF(s);
            bufferOut.flush();
        } catch (IOException e) {
            System.out.println("IOException a enviar");
        }
    }
    //Cerramos conexión
```

```
    public void closeConnection() throws Exception{
        try {
            bufferOut.close();
            socket.close();
            //Lo ponemos a null para que vuelva a crearlo de nuevo
cuando se vuelva a llamar y controlar bien las excepciones
            socket=null;
            System.out.println("Conexión terminada");
        } catch (IOException e) {
            System.out.println("IOException on cerrarConexion()");
        }
    }

}
```

8.12 Clase ConfigurationApp (Android App)

```
package com.flores.jorge.interaction;

import android.app.Activity;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.preference.PreferenceManager;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

//Parte lógica de la vista de configuración
public class ConfigurationApp extends Activity {
    //Variables de clase
    Button botonGuardar;
    TextView textoIP;
    TextView textoPuerto;
    //Sobreescribimos el metodo onCreate donde le pasamos la vista
de configuration.xml
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.configuration);
        botonGuardar = (Button)findViewById(R.id.boton) ;
        textoIP = (TextView) findViewById(R.id.ip) ;
        textoPuerto = (TextView)findViewById(R.id.puerto) ;
    }

    //Guardamos en el archivo por defecto de sharedPreferences la
dirección ip y el puerto.
    //Salimos de la actividad y como la otra principal también esta
cerrada de la aplicación.
    public void guardarDatos(View vista){
        SharedPreferences datos =
PreferenceManager.getDefaultSharedPreferences(this);
        SharedPreferences.Editor miEditor = datos.edit();
        miEditor.putString("IP",textoIP.getText().toString());

miEditor.putString("PUERTO",textoPuerto.getText().toString());
        miEditor.apply();
        finish();
    }

}
```

8.13 Clase HiloConexion (Android App)

```
package com.flores.jorge.interaction;

//Imports
import android.app.Activity;
import android.content.Context;
import android.os.AsyncTask;
import android.widget.TextView;

//Clase para realizar la conexión en segundo plano con el
microcontrolador y manejar la vista
public class HiloConexion extends AsyncTask<Void,Void,Void> {
    //Variables de la clase
    private Context myContext;
    private Activity myAcivity;
    private String datosEnvio;
    private String direccionIP;
    private String puerto;
    private Client newClient = new Client();
    private TextView mensjaje;
    private boolean x=true;

    //Constructor usado para seleccionar el contexto, la actividad,
los datos a enviar y donde se envian.
    public HiloConexion( Context myContext, Activity myActivity,
String datosEnvio,String direccionIP,String puerto){
        this.myContext = myContext;
        this.myAcivity =myActivity;
        this.datosEnvio =datosEnvio;
        this.direccionIP = direccionIP;
        this.puerto = puerto;
        mensjaje = (TextView)myActivity.findViewById(R.id.mensaje);
    }


    @Override
    protected void onPreExecute() {
        super.onPreExecute();
    }

    //Metodo despues de ejecutar la conexion (doInBackground)
    //En este metodo si falla la conexion ponemos el fallo en la
vista.
    //Si todo va bien lo indicamos.
    //Esto lo haccemos mediante una variable  booleana llamada x.
    //Por  último cerramos la conexion.
    @Override
    protected void onPostExecute(Void aVoid) {
        super.onPostExecute(aVoid);
        if(x){

mensjaje.setTextColor(myContext.getColor(R.color.colorOK));
            mensjaje.setText("Conexión Correcta. Comando:  " +
datosEnvio);
        }else {

mensjaje.setTextColor(myContext.getColor(R.color.colorError));
```

```
            mensjaje.setText("Fallo de conexión. Comando: " +
datosEnvio);
        }
        new HiloFin().start();
    }

    @Override
    protected void onProgressUpdate(Void... values) {
        super.onProgressUpdate(values);
    }

    @Override
    protected void onCancelled(Void aVoid) {
        super.onCancelled(aVoid);
    }

    @Override
    protected void onCancelled() {
        super.onCancelled();
    }

    //Pasamos los datos al microcontrolador
    //Si falla en algún momento ponemos x a false
    @Override
    protected Void doInBackground(Void... aVoid) {
        try {

newClient.openConnection(direccionIP,Integer.parseInt(puerto));
            newClient.openStream();
            newClient.send(datosEnvio);
            newClient.closeConnection();
        }catch (Exception e){
            e.getMessage();
            x=false;
        }
        return null;
    }

    // Clase interna usada para cerrar la conexion.
    class HiloFin extends Thread {
        @Override
        public void run() {
            myAcivity.runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    try{
                        Thread.sleep(3000);
                        myAcivity.finish();
                    }catch (Exception e){
                        System.out.println("Fallo: " +
e.getMessage());
                        myAcivity.finish();
                    }
                }
            });
        }
    }

    }
```

8.14 Clase MainActivity (Android App)

```
package com.flores.jorge.interaction;
//Imports
import android.app.Activity;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.os.Handler;
import android.preference.PreferenceManager;
import android.speech.SpeechRecognizer;
import android.speech.tts.TextToSpeech;
import java.util.Locale;
import android.widget.TextView;
import android.widget.Toast;
import android.speech.RecognizerIntent;
import android.content.Intent;
import android.content.ActivityNotFoundException;

//Clase de la actividad principal la cual extiende de Activity
public class MainActivity extends Activity {
   //Declaracion de variables de la clase Main Activity
    private String direccionIP;
    private String puerto;
    TextToSpeech t1;
    TextView textIpPort;
    private SpeechRecognizer sr;
    private Handler temporizadorHablar;
    private Handler temporizadorEscucha;
    //Sobrreescribimos el metodo onCreate donde establecemos la
vista con el archivo activity_main
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        //Texto que mostrará la dirección ip  y el puerto que se
guarde en la configuración.
        textIpPort = (TextView)findViewById(R.id.texto) ;
        //Accedemos a las preferencias y ponemos No Data si no
encontrase nada
        SharedPreferences datos =
PreferenceManager.getDefaultSharedPreferences(this);
        direccionIP= datos.getString("IP","No Data");
        puerto= datos.getString("PUERTO","No Data");
        //Clase usada para el reconocimiento de voz
        sr = SpeechRecognizer.createSpeechRecognizer(this);
        //Instanciamos una nueva ClaseEscucha como
RecognitionListener ya que esa clase implementa esa interfaz
        //
        sr.setRecognitionListener(new
ClaseEscucha(getApplicationContext(),this,direccionIP,puerto));
        //Creamos los Handler para que se ejecute el hilo de hablar
despues del primer segundo
        // También se ejecutará el hilo para escuchar después de 3
segundos
        temporizadorHablar=new Handler();
        temporizadorEscucha=new Handler();
        temporizadorHablar.postDelayed(hiloHabla,1000);
        temporizadorEscucha.postDelayed(hiloEscucha,3000);
    }
```

```
    //Metodo para hablar y decir la primera frase de la app.
    private  Runnable hiloHabla= new Runnable() {
        @Override
        public void run() {
            String toSpeak1 = "¿En que puedo ayudarte?";
            t1.speak(toSpeak1, TextToSpeech.QUEUE_FLUSH, null,null);
        }
    };

    //Metodo para esuchar lo que dice el usuario
    private  Runnable hiloEscucha= new Runnable() {
        @Override
        public void run() {
            startVoiceInput();
        }
    };

    //Sobreescribimos el metodo onResume ya que pasará siempre la
aplicacíon por aquí
    // viendo el ciclo de la actividad si se pausa, si se
inciia,etc.
    @Override
    protected void onResume() {
        super.onResume();
        //En caso que los datos de dirección y puerto esten vacios
ponemos No Data
        if(direccionIP.isEmpty() || puerto.isEmpty()){
            direccionIP= "No Data";
            puerto= "No Data";
        }
        //Establecemos los valores de dirección y perto en la vista.
        textIpPort.setText(direccionIP +":"+ puerto);

        //Creamos una nueva instancia de TextToSpeech para hablar
donde sobreescribimos el metodo
        // onInit y si no hay error establecemos el lenguaje al que
trae nuestro dispositivo por
        // defecto.
        t1=new TextToSpeech(getApplicationContext(), new
TextToSpeech.OnInitListener() {
            @Override
            public void onInit(int status) {
                if(status != TextToSpeech.ERROR) {
                    t1.setLanguage(Locale.getDefault());
                }
            }
        });
    }

    //Metodo para escuchar
    private void startVoiceInput() {
        //Se crea un intent donde se le pasan algunos datos como el
idioma y un prompt
        // que aparece como un Toast para que el usaurio sepa cuando
hablar.

        Intent intent = new
Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH);
        intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL,
RecognizerIntent.LANGUAGE_MODEL_FREE_FORM);
```

```
        intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE,
Locale.getDefault());
        intent.putExtra(RecognizerIntent.EXTRA_PROMPT, "Habla
Ahora");
        //Ejecuta la instancia de speechRecognizer, si falla muestra
un Toast.
        try {
            //Linea de depuración.

System.out.println(RecognizerIntent.ACTION_RECOGNIZE_SPEECH.toString
());
            sr.startListening(intent);
        } catch (ActivityNotFoundException a) {
            Toast.makeText(getApplicationContext(), "Acivity no
encontrada",Toast.LENGTH_SHORT).show();
        }catch (Exception e){
            Toast.makeText(getApplicationContext(),
"Excepción",Toast.LENGTH_SHORT).show();
        }
    }

    //Metodo sobreescrito para destruir lo que se usa en la app
cuando esta termina.
    @Override
    protected void onDestroy() {
        super.onDestroy();
        sr.destroy();
        t1.stop();
        t1.shutdown();
    };



}
```

8.15 Activity Main.xml (Android App)

```
<?xml version="1.0" encoding="utf-8"?>
<!--Archivo de la actividad principal activity_main.xml donde
tenemos
La vista de nuestra aplicacion.
Esta vista esta formada por un ConstraintLayout donde dentro tenemos
Un Linear Layout (Con un text view ) , despues una ImageView
La cual hace que vaya cambando la imagen
Y varios TextView con el fin de mostrar informacion de la conexion

Algunos componentes estan referenciados con otros para elegir sus
posiciones


-->
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/gradiente"
    tools:context=".MainActivity"
    tools:layout_editor_absoluteY="81dp">
```

```
<LinearLayout
    android:id="@+id/linearLayout"
    android:layout_width="368dp"
    android:layout_height="90dp"
    android:gravity="center"
    android:orientation="horizontal"
    tools:layout_editor_absoluteX="8dp"
    tools:layout_editor_absoluteY="8dp">

    <TextView
        android:id="@+id/textView3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="APP INTERACTION"
        android:textAlignment="center"
        android:textColor="@android:color/holo_red_dark"
        android:textSize="30sp"
        android:textStyle="bold|italic" />
</LinearLayout>

<ImageView
    android:id="@+id/imageView"
    android:layout_width="216dp"
    android:layout_height="220dp"
    android:layout_marginStart="146dp"
    android:layout_marginTop="8dp"
    android:layout_marginEnd="238dp"
    android:layout_marginBottom="136dp"
    android:src="@drawable/comandos"
    android:visibility="visible"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.287"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/linearLayout"
    app:layout_constraintVertical_bias="0.224" />

<TextView
    android:id="@+id/texto"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="8dp"
    android:layout_marginTop="8dp"
    android:layout_marginEnd="8dp"
    android:layout_marginBottom="8dp"
    android:textColor="@android:color/background_dark"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.41"
    app:layout_constraintStart_toEndOf="@+id/textView4"
    app:layout_constraintTop_toBottomOf="@+id/imageView"
    app:layout_constraintVertical_bias="1.0" />

<TextView
    android:id="@+id/mensaje"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="8dp"
```

```
        android:layout_marginTop="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginBottom="8dp"
        android:textAlignment="center"
        android:textSize="24sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.501"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/imageView"
        app:layout_constraintVertical_bias="0.341" />

    <TextView
        android:id="@+id/textView4"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="16dp"
        android:layout_marginTop="8dp"
        android:layout_marginBottom="8dp"
        android:text="Direccion de conexión: "
        android:textColor="@android:color/background_dark"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/imageView"
        app:layout_constraintVertical_bias="1.0" />
</android.support.constraint.ConstraintLayout>
```

8.16 Configuration.xml (Android App)

```
<?xml version="1.0" encoding="utf-8"?>
<!--Vista para la actividad de configuración de la aplicación.
Esta vista esta compuesta por un ConstraintLayout
Donde se incluyen varios EditText para introducir información
También se incluyen TextView para mostrar texto y un botón para
realizar el guardado de puerto y dirección ip
Además se incluye un LinearLayout con el fin de mostrar en la parte
superior el título de la app.
-->
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/gradiente">

    <EditText
        android:id="@+id/ip"
        android:layout_width="154dp"
        android:layout_height="wrap_content"
        android:layout_marginTop="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginBottom="8dp"
        android:ems="10"
        android:inputType="text"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.806"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/linearLayout3"
```

```
        app:layout_constraintVertical_bias="0.122" />

    <EditText
        android:id="@+id/puerto"
        android:layout_width="152dp"
        android:layout_height="wrap_content"
        android:layout_marginTop="8dp"
        android:layout_marginBottom="8dp"
        android:ems="10"
        android:inputType="number"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.771"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/linearLayout3"
        app:layout_constraintVertical_bias="0.435" />

    <Button
        android:id="@+id/boton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="8dp"
        android:layout_marginBottom="8dp"
        android:onClick="guardarDatos"
        android:text="Guardar"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/puerto"
        app:layout_constraintVertical_bias="0.619" />

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="16dp"
        android:text="Dirección IP"
        android:textSize="24sp"
        app:layout_constraintBottom_toBottomOf="@+id/ip"
        app:layout_constraintStart_toStartOf="parent" />

    <TextView
        android:id="@+id/textView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="68dp"
        android:text="Puerto"
        android:textSize="24sp"
        app:layout_constraintBottom_toBottomOf="@+id/puerto"
        app:layout_constraintStart_toStartOf="parent" />

    <LinearLayout
        android:id="@+id/linearLayout3"
        android:layout_width="368dp"
        android:layout_height="90dp"
        android:gravity="center"
        android:orientation="horizontal"
        tools:layout_editor_absoluteX="8dp"
        tools:layout_editor_absoluteY="8dp">
```

```
        <TextView
            android:id="@+id/textView3"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="APP INTERACTION"
            android:textAlignment="center"
            android:textColor="@android:color/holo_red_dark"
            android:textSize="30sp"
            android:textStyle="bold|italic" />
    </LinearLayout>
</android.support.constraint.ConstraintLayout>
```

8.17 Colors.xml (Android App)

```
<?xml version="1.0" encoding="utf-8"?>
<!--Recursos de colores usados
-->
<resources>
    <color name="colorPrimary">#008577</color>
    <color name="colorPrimaryDark">#00574B</color>
    <color name="colorAccent">#D81B60</color>
    <color name="colorOK">#24d81b</color>
    <color name="colorError">#9f1448</color>
</resources>
```

8.18 Strings.xml (Android App)

```
<!--Recursos de texto donde se muestra el nombre de la app
 aunque no se han agregado otros idiomas
-->
<resources>
    <string name="app_name">Interaction</string>
</resources>
```

8.19 Styles.xml (Android App)

```
<!--Recursos de estilos de la app
-->
<resources>

    <!-- Base application theme. -->
    <style name="AppTheme"
parent="Theme.AppCompat.Light.DarkActionBar">
        <!-- Customize your theme here. -->
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
    </style>

</resources>
```