Dron Terrestre
Modular Multiaplicación

Autor: Esaú García Sánchez-Torija

Tutor: Santiago Ruiz Radigales

Centro: INS Baix Penedes

Curso: 2º Bachillerato

Dron Terrestre Modular Multiaplicación

1.- Introducción 3
2.- Antecedentes 3
3.- Hipótesis y objetivos 3
4.- Materiales y métodos 5
5.- Resultados 8
6.- Conclusiones 10
7.- Agradecimientos 10
8.- Bibliografía y Webgrafía 11
9.- Ilustraciones 13

1.- Introducción

Desde que tengo recuerdos la tecnología me ha gustado. Gracias a este interés innato he logrado adentrarme en el mundo de la informática a una edad muy temprana, unos 14 años, que me ha permitido llevar a cabo este proyecto y que me ha hecho evocar tanta pasión en él, siendo así mi principal motivación. Debido a que no enseñan a programar en ninguna asignatura de primaria ni secundaria (ni siquiera tecnología), tuve que ser autodidacta y aprender a programar

Como quería aumentar mis límites se me presentó este proyecto como una oportunidad única en la que podía ponerme a prueba para adentrarme en un mundo nuevo. Para ello tuve que aprender a programar en Java, aprender a usa este mismo lenguaje para crear una aplicación de *Android*, crear un servidor utilizando una herramienta desconocida para mi llamada *NodeJS* (en combinación con muchas otras) y cómo implementar todo esto para que trabaje en harmonía con un sector con el que nunca había experimentado: el hardware.

2.- Antecedentes

Antes de empezar este proyecto ya me planteé como quería orientarlo. Estaba seguro al 100% de que quería orientarlo a la informática y como el campo que abarca la informática era muy amplio decidí hacerlo diferente a lo que solía hacer y lo orienté de manera que pudiese adentrarme en el mundo de la robótica pudiéndome apoyar sobre unas bases firmes que sustentaran el trabajo, que son mis conocimientos de programación de software.

Para ello le di un segundo uso a mi robot *Lego Mindstorm NXT*, un robot orientado hacia los niños para introducirlos al sector de la tecnología y con el que ya experimentaba de más joven. Gracias a eso, me interesé más por la informática y aprendi a usar el software de desarrollo que venía incluido con el robot aunque programación únicamente visual, es decir, del tipo *WYSIWYG* (*What You See Is What You Get*, Lo que ves es lo que obtienes).

3.- Hipótesis y objetivos

Teniendo mis antecedentes en mente, me planteé si podría ser capaz de romper los límites que tenía mi robot y hacer de él un robot mejor. Para ello pensé qué quería hacer con el robot, ver si el robot era capaz de hacerlo e intentar solucionarlo, dando por supuesto que inicialmente no podía hacerlo.

Desde el primer momento en que utilicé el kit de desarrollo del robot me di cuenta de sus limitaciones. El robot podía ejecutar órdenes, ya fuesen arbitrarias o basándose en los múltiples sensores que posee, pero lo hacía con el lenguaje de programación *NXC* (*Not eXactly C*, No eXactamente C).

Esto quiere decir, que al ser un lenguaje derivado de C, la única forma de ejecutar un programa en el robot era compilando todas las ordenes y después enviando el programa compilado por cable USB para poder ejecutarlas definitivamente presionando un botón en el robot. Esto hacía que un simple programa como por ejemplo ir a la posición C solo

pudiese ejecutarse desde una posición A, por lo que si quería ir desde B, tenía que modificar por completo el programa, además de tener que estar literalmente detrás de él. Desde ese instante me dije, "¿sería posible hacer que el robot fuese completamente remoto y que funcionase independientemente de la posición donde se encuentre?".

A partir de esta hipótesis surgió el objetivo principal y los inconvenientes y limitaciones que tuve que solucionar para poder llevarlo a cabo.

Por otra parte, existen una serie de pautas que me impuse que al fin y al cabo, juegan el mismo papel que un objetivo, ya que me impuse la obligación de cumplirlas.

Para empezar quería que la aplicación pudiese, desde el primer momento, ser instalada en cualquier dispositivo, por lo que al crear la aplicación el propio *IDE* ya te informa de que versiones son las más utilizadas y que porcentaje de personas pueden instalar tu aplicación en base a la versión mínima que has especificado que es necesaria para instalar la aplicación. En el caso de mi proyecto establecí que la versión de la *API* de *Android* mínima sería la 9, que equivale a la versión 2.3 comúnmente conocida como *Froyo*.

Como versión máxima establecí la última que correspondía en su momento a la *API* 21 que corresponde a la versión de *Android* 5.0 comúnmente conocida como *Lollipop*. Basándose en esos dos datos y en los datos que obtiene Google sobre el número de dispositivos vinculados estableció que el 99.97% de los dispositivos *Android* del mundo eran compatibles con mi aplicación (teniendo en cuenta que de la *API* 21 a la *API* 22 no hay ningún cambio significativo que haga que la aplicación colapse; véase Ilustración 1).

Teniendo el problema de la compatibilidad del código de la aplicación solucionado, había otro problema y es que entre las versiones de *Android* las *API* no varían pero según el fabricante o la versión de *Android* la forma en que vemos visualmente un elemento sí que lo hace. Para solucionar este problema tuve que escribir porciones de código que dibujasen la interfaz porque dejar que lo hiciese nativamente *Android* desencadenaba en problemas de alineación y en consiguiente una experiencia de usuario mucho peor.

Por otra parte me había planteado seguir las guías de diseño de *Google*, llamadas *Material Design*, para darle un aspecto visual más moderno a la aplicación cosa que era imposible dejando en manos de *Android* el dibujado de la interfaz. No solo eran estrictas en cuanto a las medidas si no que los colores debían combinar entre ellos además de las relaciones entre ellos, es decir, que el color de la barra de notificaciones sea un 10% más oscuro que el color del tema.

Por otra parte, quería que la aplicación fuese un tanto personalizable pues no hay mucha configuración que se pueda manipular, ya que la aplicación sirve un único propósito, así que añadí una serie de configuraciones que cambiaban el comportamiento y las animaciones, además de poder dejar escoger al usuario el tema de colores que más desease de los 18 disponibles (véase Ilustración 2).

Un problema muy común al que toda aplicación web se enfrenta es hacer las páginas web adaptables o *responsive* sin tener que redirigir a los móviles a otro dominio (ejemplo m.facebook.com) y de manera que sea cual sea la plataforma los recursos se manejen de manera óptima sin que ninguna plataforma reciba un impacto perceptible

por ello. Para ello tuve que usar un diseño basado en tarjetas similar al usado en la herramienta social *Google+* y *Google Now* que cambia el número de columnas o tarjetas mostradas en una sola fila basándose en el campo de visión de la pantalla o *viewport*.

De esta manera los cálculos se hacían cada vez que se redimensiona y no tiene impacto cada vez que se añade o quita una tarjeta puesto que los valores ya se han precalculado. Además, debido a la sencillez que esconde este proceso las páginas web cargan rápidamente sin notar apenas que se trata de una aplicación web y no una nativa/de escritorio (véase Ilustración 3).

Como último problema se presentaba la compatibilidad de los servidores, y es que ya que hice la aplicación de *Android* lo más compatible posible no iba a no hacerlo con el servidor. Para ello, usé el lenguaje de programación *JavaScript* pero como en el navegador no tiene acceso a archivos (lo tiene pero muy limitado) y no puede abrir ni cerrar puertos decidí programarlo para ejecutarse en un servidor *NodeJS* o *io.js*.

Esto lo hacía muy flexible, además de multiplataforma ya que el código *JavaScript* se compila a *byte-code* en el tiempo de ejecución o *Runtime* y la aplicación, ya sea *NodeJS* o *io.js*, ya se encarga de ejecutar las respectivas órdenes en cada plataforma, ya sea **NIX* (*Linux*, *Ubuntu*, *Mac OS X*), *BSD*, *Windows* o el que estemos utilizando. Además, utilizando herramientas de superusuario o *root* somos capaces de emular *Linux*/*Ubuntu* en *Android* y ejecutar nuestro servidor en nuestro propio móvil (véase Ilustración 4).

4.- Materiales y métodos

Tratándose de un proyecto centrado en la programación de un robot, los materiales físicos son un tanto escasos, ya que principalmente hacen falta recursos de software y no recursos físicos. No obstante como los materiales son recursos necesarios para poder llevar a cabo el proyecto, se pueden calificar los recursos de software como tales.

Dentro de los recursos de hardware necesarios tenemos:

- El robot que por defecto es muy limitado pese a sus muchas combinaciones y que queremos lograr hacerlo remoto e independiente. En mi caso he usado un robot *Lego Mindstorm NXT V2.0* y aunque no haya sido probado en un *Lego Mindstorm NXT V3.0*, la compatibilidad está asegurada.

- Un móvil que sea Smartphone, que nos servirá para enviar las ordenes que recibiremos del servidor, sea donde sea que se ejecute, al robot que queremos mover. En la versión actual del proyecto se usa una autenticación temporal basada en *tokens* o claves de acceso único, por lo que si queremos usar los servicios deberemos tener servicio de GPS (restricción propia que he impuesto y que se puede eliminar) sin contar con los requisitos básicos como tener instalado *Android*, tener un servicio de *Bluetooth*, una cámara y micrófono para poder grabar audio y video para transmitirlo, conexión a Internet y un navegador compatible con *HTML 5*, como por ejemplo, *Google Chrome* o *Mozilla Firefox*.

- Este último no es un requisito indispensable pero se recomienda disponer de un mando compatible con el PC, ya sea de *Xbox 360*, *Xbox One*, *PS3*, *PS4* o un mando genérico ya que el control remoto se puede manejar con dichos mandos

haciendo que sea más preciso y proporciona más combinaciones de giro ya que cada joystick maneja cada rueda independientemente.

- Un dispositivo que maneje el robot remotamente, preferiblemente un PC, y que esté dotado de conexión a Internet, de un navegador compatible con *HTML 5*, como por ejemplo *Google Chrome* o *Mozilla Firefox* y si es posible que sea compatible con cualquiera de los mandos mencionados en el apartado anterior.

Dentro de los recursos de software necesarios tenemos:

- Eclipse: entorno de programación comúnmente utilizado para programar en *Java* y que, debido a que es *Open Source*, se puede programar en *Java* para *Android* haciendo uso de *plugins* creados por *Google* específicamente para ello.

- Android Studio: entorno de programación creado (mucho después de que empezase el proyecto), basado en la solución de *JetBrains* llamada *IntelliJ IDEA* y que ha sido específicamente modificado para dar únicamente soporte para crear aplicaciones para *Android*.

- Android SDK: kit de desarrollo para Android que contiene tanto dependencias usadas por *Android* Studio como herramientas que nos permiten acceder a la consola de *Android* o descargar la documentación, binarios e imágenes (.iso) de *Android* para poder emular.

- AVD: programa que es capaz de emular un dispositivo *Android* (pobremente) y que permite simular llamadas, mensajes de texto, notificaciones, niveles de batería, orientación y cualquier cosa posible si queremos depurar nuestra aplicación y no tenemos a nuestro alcance un dispositivo *Android* físico.

- JDK: entorno de programación que nos aporta las herramientas necesarias para compilar, depurar, optimizar y proteger nuestros programas en *Java* y que permite a *Android Studio* o *Eclipse* o compilar las aplicaciones de *Android*.

- Gradle: gestor de dependencias que se puede integrar con *Android Studio* para automatizar procesos como la compilación, depuración, optimización y firmado de las aplicaciones además de que permite definir macros para hacer más fácil la programación en general.

- Git: herramienta de *CVS* (*Concurrent Versioning System*, Sistema de control de versiones concurrentes) que permite mantener un historial de los cambios hechos en los archivos además de poder revertir los cambios, comparar versiones y ver diferencias, usado para que diferentes puedan colaborar en un proyecto (en mi caso el proyecto lo hecho solo pero lo he subido a un repositorio *Open Source* para el cual es necesario usar esta herramienta).

- LESS: lenguaje de pre-procesado que permite extender el lenguaje *CSS* (*Cascading Style Sheets*, hojas de estilo en cascada) y aportarle nuevas funciones como variables, operaciones aritméticas, macros, selectores complejos, mixins, compatibilidad para propiedades con prefijos específicos para cada navegador en propiedades experimentales, entre otras muchas propiedades y que hacen que

con unas simples líneas de código puedas generar hojas de estilo mucho más extensas sin tener que escribirlo todo manualmente.

- SASS: lenguaje de pre-procesado que permite extender el lenguaje *CSS* y que se puede combinar junto con *LESS* para añadir más propiedades como operaciones con colores para calcular tonos más oscuros, con más brillo, más saturados, etc.

- Compass: aplicación sencilla que permite facilitar el proceso de compilación de *SASS* y *LESS* para que automáticamente se recompilen y compriman cuando se detecte algún cambio.

- Open SSL: herramienta *Open Source* distribuida con la gran mayoría de sistemas operativos y que permite generar certificados y provee al sistema de generadores de números pseudoaleatorios a la vez que de algoritmos de encriptación. Es la misma herramienta que permiten tanto a los servidores como a los navegadores cifrar las comunicaciones a través del protocolo seguro *https*.

- SGit: aplicación de *Android* que permite usar los comandos de la herramienta *git* para poder manejar las versiones concurrentes.

- ADB: conjunto de herramientas distribuidas con el *SDK* de *Android* que nos permiten un acceso total a los recursos de nuestro móvil para poder depurar las aplicaciones de *Android*.

- Cygwin: emulador de *Bash* (Bourne against shell) conocido por ser el intérprete de comandos de *UNIX* por defecto, totalmente porteado a *Windows* para no tener que usar ningún comando de *Batch*, el intérprete por defecto de la línea de comandos de *Windows*.

- NodeJS: plataforma (basada en el motor de *Google Chrome* llamado *V8*) que permite ejecutar *JavaScript* en el ordenador sin depender de una navegador y que expone un gran conjunto de *API* que nos permiten hacer cualquier tipo de aplicación (de consola). Es la plataforma usada para ejecutar nuestro servidor.

- io.js: Alternativa a *NodeJS* que mantiene todos sus estándares y además permite programar con las últimas versiones de *JavaScript* (*Harmony* o *ES6*) además de que tiene un gestor interno que nos permite usar cualquier versión de *NodeJS* para ejecutar los scripts y depurarlos.

- GitHub for Windows: gestor visual de las versiones creadas con git que nos permite ver las diferencias entre los archivos marcando las líneas y coloreando la sintaxis como si de un editor de código se tratase.

El método de trabajo o proceso de investigación y creación del proyecto se compone de 4 fases diferentes. Tres de ellas tienen la misma categoría ya que cada una de ellas es el proceso de creación de la aplicación en cada uno de los dispositivos (móvil, aplicación web (PC) y servidor) y la última las engloba a todas ya que es el proceso que hace que se comuniquen todas entre sí.

- La aplicación de Android: Compuesta por 3 fases de creación:

  - Diseño del funcionamiento de la aplicación (no visual), es decir, decidir cómo se guardaran los datos y como se enviaran y recibirán los que deban serlo (como la comunicación por Bluetooth con el robot).

  - Diseño de la *UI* (*User Interface*, interfaz de usuario), teniendo en cuenta la *UX* (*User Experience*, experiencia de usuario) y diseñar como se mostrarán los datos que ya teníamos al haber diseñado ya el funcionamiento de la aplicación. En algunos casos el diseño de la interfaz se produce paralelamente al diseño del funcionamiento, porque uno limita al otro.

  - Un último paso es depurar la aplicación mediante lo que se denomina *Monkey Tests* o tests de monos, que consiste en intentar colapsar la aplicación por cualquier medio, es decir, interrumpir procesos, comunicaciones, cerrar la aplicación mientras se establece la conexión, etc. Con el objetivo de ver si la aplicación o la interfaz dejan de responder y poder solucionar errores accidentales.

- La aplicación Web: consta de un proceso de creación único muy complejo ya que se busca crear una única aplicación, compatible con el mayor número de navegadores a la vez que con el mayor número de resoluciones tratándolo de hacer flexible, es decir, sin redirigir al usuario a una página móvil especifica (ejemplo m.facebook.com) y consiguiendo que dicha flexibilidad no tenga el mínimo impacto en el rendimiento de la página en todos los aspectos, incluyendo tiempo de procesado (tiempo que tarda en renderizar) como de transmisión (tiempo que tarda el servidor en enviarnos todos los datos de la página). Todo este proceso de creación combinado al mismo tiempo con unas pautas y guías de diseño para hacer que se parezca a una aplicación nativa/de escritorio.

- El servidor *NodeJS*: consta de un solo proceso de creación en el cual se diseña el sistema que administra los usuarios, las sesiones, la validación de datos y ayuda a establecer las comunicaciones *P2P* entre dos usuarios. Por otra parte, como contiene información sensible hay muchos procesos de saneo de variables debido a que existen formas de atacar los servidores introduciendo cadenas de caracteres maliciosas.

- Interacción entre los 3 componentes: Este proceso es sin duda el más largo y difícil de mantener, debido a que hay que sincronizar los diferentes sistemas usados (*Java*, *NodeJS*, *JavaScript*, etc.) para que se comuniquen correctamente. Esto da lugar a que un solo cambio en uno de ellos nos obliga a tener que actualizar los otros porque de lo contrario la comunicación fallará.

5.- Resultados

Teniendo en cuenta los objetivos del proyecto puedo afirmar que los resultados han sido exitosos. Es más, muchas características del proyecto se han optimizado para que vayan más fluidas. Un ejemplo es, que los ajustes de las aplicación que se guardan como booleanos (verdadero o falso) se han guardado en forma de enteros, es decir, cada uno de estos booleanos han sido asignados a una posición en los bits de un número entero,

con lo cual podemos transmitir todos los datos con un solo número en lugar de transmitir múltiples booleanos múltiples veces, comportamiento similar a los permisos de lectura, escritura y ejecución en los sistemas *UNIX*.

Por otra parte existen alternativas que podrían mejorar el rendimiento de todo el sistema. Un ejemplo sería cambiar el sistema de usuarios no persistente a uno que si lo sea, como por ejemplo una base de datos *Redis* lo cual tendría un impacto directo en:

- La batería del móvil: el móvil usa el servicio de GPS para que los usuarios sepan cual es el robot que quieren controlar (en caso de que haya más de uno) y esto se hace mirando las coordenadas GPS que el móvil debe tener activadas. En caso de usar un sistema de usuarios persistente podríamos asignar un identificador único a un usuario y así no sería obligatorio tener GPS para poder identificar al dispositivo.

- La memoria del servidor: en el caso de que haya un solo robot el sistema actual es el más rápido y eficiente, pero en el caso de que hubiesen muchos usuarios conectados cada usuario es un espacio asignado en la memoria que aunque este se desconecte no se borra de ella por lo que el servidor podría llegar a colapsarse alcanzado un límite de usuarios que llenen la *RAM*.

- Eficiencia de las sesiones: a cada sesión que identifica un usuario se le asigna un toquen que contiene los datos encriptados del usuario con una clave privada. Más tarde el usuario tiene que volver a reenviar dicho *token* para iniciar sesión. En el caso de que funcionase con un sistema de usuario persistente el *token* podría ser un simple código aleatorio que identificase temporalmente al usuario sin la necesidad de exponer información privilegiada y sin la necesidad de cifrar y descifrar en cada inicio de sesión o reconexión.

Por otra parte, se podrían aplicar ciertos cambios que optimizarían ciertas características pero emporarían otras y que estos casos se deben analizar individualmente según cada dispositivo en cuestión. Por ejemplo:

- Duración de la batería y latencia de la conexión: si analizamos como circulan los datos a través de la red y de los dispositivos que forman el proyecto vemos que el PC que usa la aplicación web envía datos al servidor, de ubicación variable, y éste se los envía a la aplicación del móvil, también de ubicación variable, y que éste último se los transmite al robot, que podemos suponer que tiene una latencia muy baja y por lo tanto un *delay* o retraso nulo.

  Si quisiésemos optimizar la velocidad de transmisión de estos datos podríamos aplicar un cambio muy sencillo que por desgracia tiene un impacto muy grande en la duración de la batería del móvil. Si ejecutamos el servidor en el móvil lo que hacemos es anular la latencia variable que antes había entre el servidor y el móvil, con lo que únicamente tendremos una latencia variable entre el servidor y el PC pero que antes también existía.

  Con esto nos ahorramos un envió de datos, ya que se puede modificar las *DNS* para que el envío se haga localmente y ganaremos en velocidad sacrificando la batería de nuestro móvil ya que tendrá que ejecutar un proceso más.

Además de estas posibles mejoras que podríamos aplicar al proyecto existen mejoras hipotéticas, es decir, mejoras que serían posibles si se tuviesen los recursos necesarios, ya que son mejoras de hardware, como por ejemplo:

- Utilizar una versión más moderna del robot y hacer que la transmisión de datos entre el móvil y el robot se haga por cable en lugar de *Bluetooth*, ahorrándonos así mucha batería, ya que el *Bluetooth* es un servicio que consume muchos recursos y eliminando el retraso a los más mínimo posible ya que el cable es más rápido que la tecnología inalámbrica. Con el robot usado actualmente no se ha podido llevar a cabo ya que no existe ningún adaptador que sea micro USB macho y USB 1.0 macho.

- Fabricar el robot usando *Arduino* de manera que no haga falta ni siquiera tener que usar el móvil como puente y podríamos hacer que el servidor enviase las órdenes directamente al robot, porque de hecho, el robot sería el servidor y tan solo habría una única conexión activa.

6.- Conclusiones

De este trabajo de investigación puedo sacar como conclusión que a veces un trabajo puede parecer llevadero si el tema del que trata te gusta, pero eso no quita que no te vayan surgir una gran cantidad de problemas. He pasado días enteros buscando información sobre un tema, que hasta ahora, era completamente desconocido para mí y que no sabía que iba a tener que incluir en el proyecto, como es el caso de la instalación de *Linux*.

No obstante, la conclusión final es que, a pesar de todas las dificultades encontradas y el trabajo extra que han requerido, he aprendido mucho sobre como estructurar aplicaciones e incluso a usar muchas herramientas nuevas y lenguajes de programación. Esto ha hecho que mis habilidades de programación hayan aumentado en gran cantidad, y ahora, gracias a ello, seré capaz de hacer otros proyectos que hasta ahora no me había planteado hacer por lo difíciles que eran.

Además, me ha supuesto un gran reto y motivación, que me ha alegrado mucho finalizar y que me encantaría que algún día se convirtiese en realidad. Quien sabe, quizás algún día podamos controlar robots desde nuestras casas para hacer la compra, porque hacer que nos la envíen es muy aburrido.

7.- Agradecimientos

Me gustaría agradecer a mi tutor Santiago Ruiz Radigales por las pautas que me ha hecho seguir para documentar el proyecto de una manera limpia y profesional, a mi familia por el apoyo emocional que me ha supuesto y sobre todo a mi padre, quién me introdujo en el mundo de la informática de pequeño, ha conseguido que tenga un afición de la que espero vivir y porque cada día aprendo algo nuevo gracias a él.

8.- Bibliografía y Webgrafía

- Desarrollo de juegos Android:
  Autor: Mario Zechner
  Editorial: Anaya Multimedia.

- Android 4, Principios del desarrollo de aplicaciones Java
  Autor: Nazim Benbourahla
  Editorial: Ediciones Eni

- http://androlinux.com/android-ubuntu-development/how-to-install-ubuntu-on-android/

- http://developers.android.com

- http://masonry.desandro.com/

- http://nexusonehacks.net/nexus-one-hacks/how-to-install-ubuntu-on-your-android/

- http://romannurik.github.io/AndroidAssetStudio/icons-launcher.html

- http://sourceforge.net/projects/linuxonandroid/files/Ubuntu/13.04/

- http://stackoverflow.com/questions/8811594/implementing-user-choice-of-theme

- http://www.google.com/design/spec/material-d

- http://www.html5rocks.com/en/tutorials/speed/script-loading/

- https://about.gitlab.com/

- https://developer.android.com/training/material/lists-cards.html

- https://github.com/

- https://github.com/kolavar/android-su

- https://github.com/marmottes/marmottajax

- https://github.com/mkuklis/depot.js

- https://github.com/navasmdc/MaterialDesignLibrary

- https://github.com/traex/RippleEffect

- https://github.com/typicode/pegasus

- https://gitorious.org/
- https://www.gradle.org/
- https://www.heroku.com/

9.- Ilustraciones

Version	Codename	API	Distribution
2.2	Froyo	8	0.3%
2.3.3 - 2.3.7	Gingerbread	10	5.6%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	5.1%
4.1.x	Jelly Bean	16	14.7%
4.2.x		17	17.5%
4.3		18	5.2%
4.4	KitKat	19	39.2%
5.0	Lollipop	21	11.6%
5.1		22	0.8%

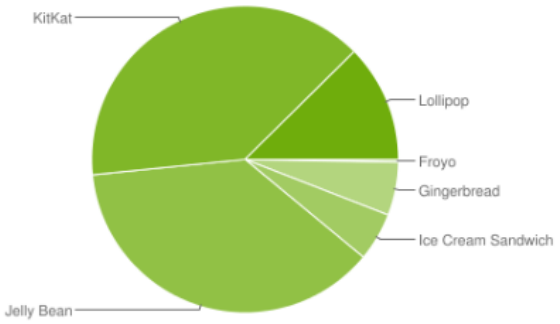

Ilustración 1. Versiones de Android utilizadas. Datos oficiales de Google.

Ilustración 2. Temas de colores de la aplicación de Android. Realización propia.

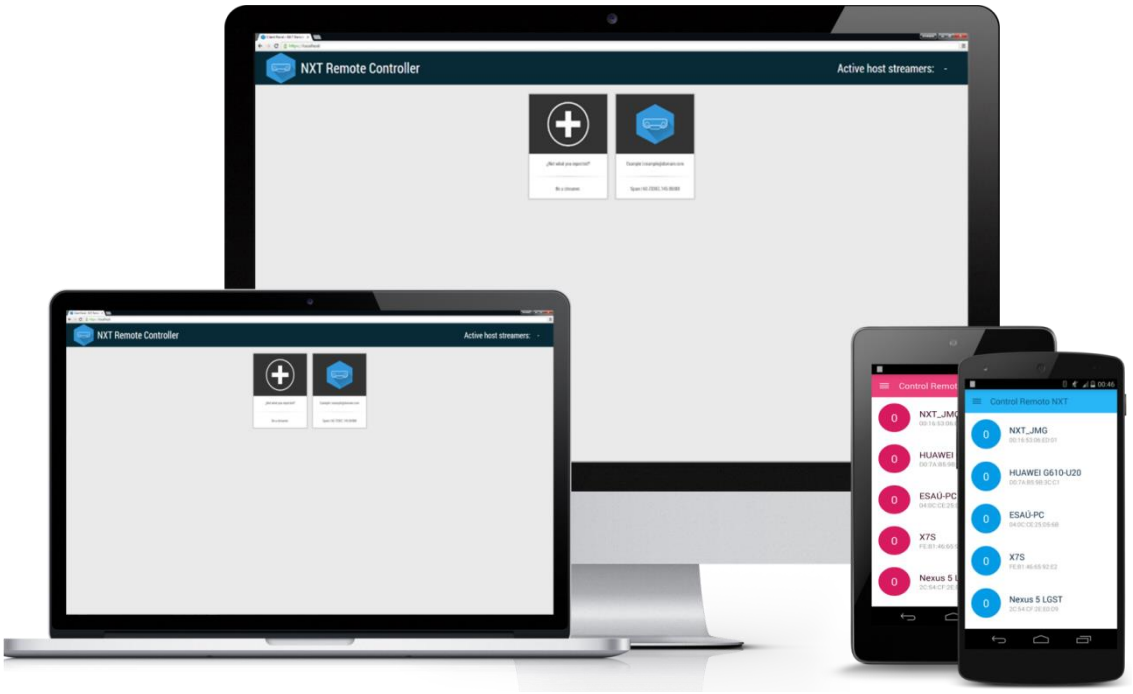

Ilustración 3. Diseño *responsive* de las aplicaciones. Realización propia.

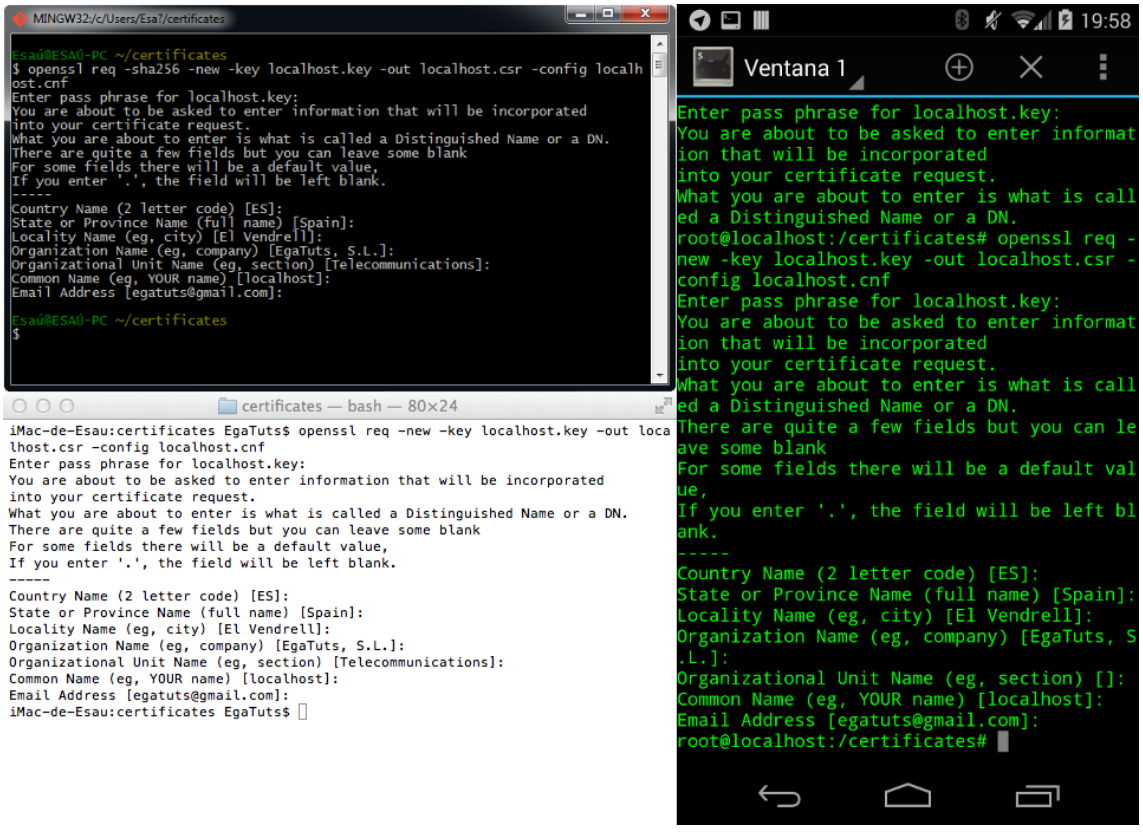

Ilustración 4. Compatibilidad multiplataforma del servidor con soporte para conexiones SSL con certificados auto-firmados. Realización propia.

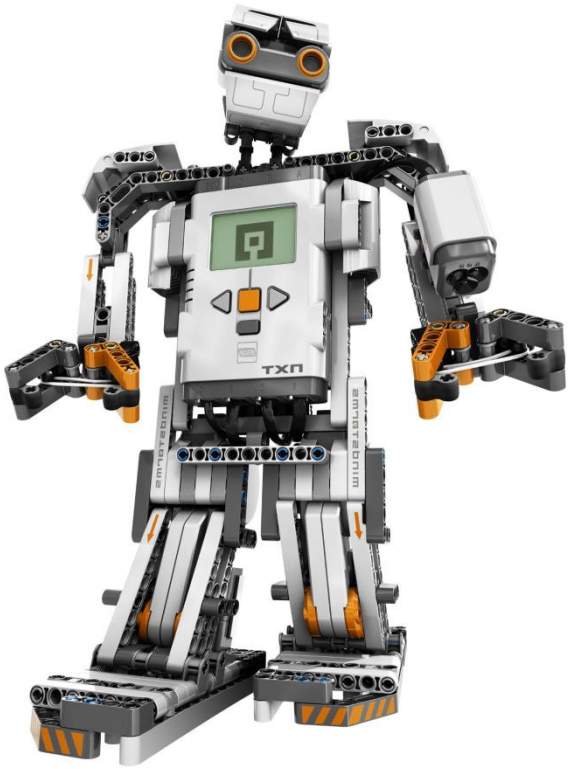

Ilustración 5. Robot Lego Mindstorm NXT V3.0. Imagen oficial de Lego.