UNIVERSITAT POLITÈCNICA DE VALÈNCIA

CAMPUS D'ALCOI

Servidor de consolas de equipos Cisco con acceso web

MEMORIA PRESENTADA POR:

Israel Cortés Gisbert

GRADO EN INGENIERIA INFORMATICA

Convocatoria de defensa: *Octubre de 2017*

Resumen

El presente trabajo final de Grado en Ingeniería Informática tiene como objetivos el desarrollo de una solución software que permita a los alumnos el acceso a las consolas de administración de los distintos equipos de red Cisco de forma remota, mediante la emulación de ventanas de comandos GNU/Linux sobre web. Además, la solución debe desplegarse sobre una plataforma hardware de bajo coste, la cual realizara las conexiones con los distintos elementos de red vía conexión serie o RS-232.

Abstract

The presenta final grade in Computer/Software Engineering aims to develop a software that allows students to accesos the management consoles of the different Cisco network equipment remotely by emulating commands' windows GNU/Linux on web. In addition,the solution must spread out a low cost hardware platform,which will make connections with the different elementos of a route connection series or RS-232

Palabras Clave: *Django, RS-232, Cisco, gnu/Linux, Web*

Key words: *Django, RS-232, Cisco, gnu/Linux, Web*

Datos del proyecto

Título: Servidor de consolas de equipos Cisco con acceso web

Nombre del estudiante: Israel Cortés Gisbert

Titulación: Grado en Ingeniería Informática

Créditos: 12

Director: Raúl Llinares Llopis

Departamento: Departamento de Comunicaciones

Índice

1-Introducción

1.1-Motivación

La decisión de elegir el trabajo final de Grado no fue fácil, a pesar de no haber publicados demasiados proyectos informáticos en la web de la EPSA muchos profesores expresan su disponibilidad para realizar proyectos nacidos de ideas que pueden llegar a convertirse en trabajos finales de carrera, o de necesidades básicas en aulas o a nivel industrial que de la misma forma pueden cubrirse a través del desarrollo de un proyecto final de carrera. Es en este punto donde encontramos el trabajo presentado ya que nace de la idea del profesor Raúl Llopis Llinares de proveer al laboratorio de Comunicaciones de una herramienta que facilitará a los alumnos el acceso a los distintos elementos de red disponibles para prácticas. Todo ello de una forma cómoda, remota e integrada sobre una plataforma de bajo coste.

El proyecto se fundamentaba en resolver la dificultad que había en el laboratorio de comunicaciones a la hora de configurar los distintos elementos de red por parte de los alumnos, puesto que el requisito fundamental en el tratamiento de los equipos era la de configurarlos como si estos estuviesen salidos de fábrica. Lo cual implicaba la no disponibilidad de conexiones sobre el protocolo IP dejando como alternativa la interconexión de los equipos por medio del puerto serie o RS-232. Además, se suma la problemática de ofrecer a cada alumno una o varias conexiones simultaneas a los distintos elementos de red y la disponibilidad de estos en el laboratorio, ya que pueden encontrarse instalados en un pequeño armario móvil o en el cpd del laboratorio de forma que se dificulta la forma en que el alumno pueda trabajar con estos equipos.

Y es en este punto donde encontré la motivación principal pues esta idea conforme fue planteada constituía la aplicación directa de los conocimientos de diversas asignaturas y de sus conceptos teóricos y prácticos. Abarcaba áreas diversas como la comunicación por red, sistemas embebidos y de tiempo real, programación web, programación con distintos lenguajes y su integración en entornos Linux. Asimismo, se requería investigación de tecnologías que pudieran integrarse o adaptarse al proyecto para conferirle más solidez. Todo ello teniendo como finalidad la creación de un proyecto con aplicabilidad real en la docencia.

1.1.1-Contexto y análisis de la situación

La situación de partida del proyecto se centra en buscar una solución software, que permita a los alumnos interactuar con los elementos de red (routers, switches…) de una forma remota.

Esta situación se debe a la problemática existente en el laboratorio de comunicaciones a la hora de configurar los equipos destinados a las practicas del alumnado. Estos dispositivos de red pueden encontrarse instalados en el pequeño CPD (centro de procesamiento de datos) del laboratorio de comunicaciones o en un armario móvil para este tipo de dispositivos, lo que ocasiona una diversidad de problemas:

-Disponibilidad de equipos.

-Problemas de espacio para trabajar con seguridad y comodidad, tanto a nivel grupal como individual.

-En caso de instalar los equipos en el armario móvil, solo los alumnos cercanos pueden trabajar con los equipos. Además de verse limitado el número de equipos con los que se puede disponer en dicho rack.

-Los equipos destinados a docencia pueden estar instalados junto a otros equipos destinados a otros fines, lo que implica un problema de seguridad.

-Longitud del cableado disponible para la interconexión de equipos

Adicionalmente, la finalidad de estos equipos es que los alumnos puedan configurarlos desde cero, es decir, salidos de fábrica. Este hecho añade otra problemática al ser necesario configurarlos por puerto serie o RS-232 y al tratarse de un puerto en desuso no todos los equipos de usuario tienen un puerto COM necesario para la comunicación sin adaptadores. Este factor limita la forma en que el alumno trabaja con el hardware y solo se permite una conexión simultanea sobre un mismo dispositivo. Ya que, si varios alumnos intentasen conectarse simultáneamente sobre un mismo dispositivo, el sistema que gestiona la comunicación generaría un error y cerraría todos los enlaces serie sobre ese dispositivo.

1.2-Objetivos

Los principales objetivos que se pretendía cumplir con el proyecto son los siguientes:

. Adquirir soltura en el desarrollo de aplicaciones Web con el framework Django, así como con el lenguaje utilizado: Python

. Obtener competencia en el desarrollo de proyectos siguiendo las metodologías impartidas en asignaturas como Ingeniería del Software.

. Desarrollo de una aplicación totalmente modular siguiendo los siguientes principios de desarrollo de software: Kiss & DRY, basadas en mantener un código simple y limpio evitando la redundancia de código.

. Realizar un desarrollo que facilitara la escalabilidad y portabilidad.

. Creación de una herramienta de acceso web multiusuario para el acceso concurrente de varios usuarios a los distintos elementos de red, a través de una única conexión contra el dispositivo hardware vía comunicación serie.

. Comparativa de Hardware embebido de bajo coste para el despliegue del proyecto en el entorno de producción como en el de desarrollo. Así como la disposición del conjunto de paquetes software disponibles en dicha plataforma.

. Evaluación de viabilidad del proyecto sobre las plataformas elegidas y su relación con el conjunto de paquetes software disponibles en dichas plataformas.

. Implementación de una estructura de datos que permita el registro de nuevos usuarios, así como la asignación de políticas de seguridad para el control de accesos a diferentes áreas y funcionalidades.

. Investigar las distintas tecnologías y soluciones que ayuden en el desarrollo e implementación del proyecto.

. Creación de un dominio para aportar visibilidad y escalabilidad al proyecto en caso de que se le dé un ámbito más amplio, fuera de la red local para la que fue diseñado

1.3-Requisitos de la arquitectura

Debido a que el proyecto se basa en el desarrollo software, pero también en su integración sobre una o diversas plataformas hardware, se han separado los requisitos necesarios para su despliegue en dos partes: Cliente y Servidor.

1.3.1- Cliente

En este caso, solo será necesario un navegador web para que el usuario acceda a la aplicación. No obstante, el navegador web que utilice el cliente deberá tener habilitadas las opciones de ventanas emergentes para poder utilizar las distintas funcionalidades de la aplicación. Tanto la aplicación como su funcionalidad para el cliente ha sido probada en los navegadores siguientes: Chrome, Firefox, Internet Explorer y Opera. Se han apreciado variaciones en como se muestra la información al usuario según navegador, sin embargo, dichas variaciones no afectan a la funcionalidad.

La aplicación no precisa de conocimientos específicos en materia de informática para su utilización general, pero si en su funcionalidad especifica que radica en la manipulación, configuración y control de hardware de red como son switches y routers. Por lo que el usuario deberá tener unos conocimientos sobre como interactuar con los distintos dispositivos hardware.

1.3.2-Servidor

El servidor hace referencia a la plataforma sobre la que se desplegara nuestro proyecto. En un principio, la aplicación desarrollada funcionó en un sistema embebido sobre el cual se han hecho todos los estudios. Pero debido a cambios en la funcionalidad requerida en el proyecto en una fase avanzada de este, el sistema sobre el que se despliega la aplicación cambió. La limitación que provoco el cambio hardware se debió a un requisito software. La limitación y la decisión del cambio en los requisitos se explicará más adelante en el apartado “3.4-Estudió software”. En las líneas siguientes se detallarán en subapartados los requisitos hardware tanto iniciales como finales, y los requisitos software.

Los requisitos hardware, vienen definidos por la plataforma con menor capacidad de computo sobre la que se ha desplegado la aplicación.

1.3.3-Hardware requerido inicial

Inicialmente y tras un estudio de las plataformas hardware de bajo coste más extendidas se eligió el modelo 3B de RaspberryPI para el despliegue de la aplicación. Los requisitos basados en esta plataforma son los que siguen:

-Raspberry pi, modelo 3B que dispone del siguiente Hardware integrado: Cpu Quad-Core de 64 bits, ARMv8@1'2, 1GB de RAM.

-Tarjeta de memoria micro-sd, esta hará la función de almacenamiento masivo para la raspberry pi. Su tamaño mínimo debe ser de 16GB, garantizando así el espacio para el sistema operativo, la aplicación y la base de datos.

-Dependiendo del número de dispositivos hardware a conectar, se requerirá de los respectivos conversores serial RS232 a USB.

1.3.4-Hardware requerido final

Tras el cambio en los requisitos iniciales, se abordó el problema del cambio de hardware y por ende de arquitectura. Los detalles del cambio, así como el porqué de este se detallan en el apartado "3.3-Estudio Software".

Tras esto se decidió desplegar la aplicación sobre la arquitectura amd64 con la que se había desarrollado de principio a fin el proyecto, por lo cual garantizaba el buen funcionamiento y una completa compatibilidad tanto a nivel hardware como software. El proyecto fue probado en distintas plataformas determinando los requisitos mínimos que siguen:

-Sistema hardware (embebido o estándar) con una arquitectura de cpu amd64 bits y 1GB de Ram

-Disco duro con al menos 20GB para albergar tanto el sistema operativo como la aplicación y su base de datos.

-Dependiendo del número de dispositivos hardware a conectar, se requerirá de conversores serial RS232 a USB.

1.3.5-Software

En el apartado software, se incluye toda dependencia requerida, para que el despliegue del proyecto se produzca sin incidentes. A continuación, se van a definir desde los requisitos mínimos de sistema operativo a los requisitos de dependencias que se deben instalar mediante el gestor de paquetes. En el apartado "3.7-Instalación de la aplicación web y de sus componentes", se detalla el proceso completo de instalación de las dependencias y de las utilidades necesarias para desplegar la aplicación en su entorno de producción.

-Sistema operativo GNU/Linux, la aplicación ha sido desarrollada íntegramente en la distribución "Linux Mint", a pesar de ello se ha probado en distintas distribuciones y núcleos del sistema para probar su funcionamiento.

-Versión del kernel: 4.4.0-21-generic (buildd@lgw01-21)

-Python versión 3.5, y pip3.5.

-Django 1.11

-Servidor Web Tornado.

-Sqlite como Base de datos.

-Django-bootstrap3 v8.2.2,

-Gunicorn v19.7.1,

-Wheel 0.24.0,

-Tornado 4.5.1.

-Ttyd

-Tmux

1.4-Descripción de la Memoria

La presente memoria escrita describe la realización de un trabajo final de grado en Ingeniería Informática para la Universidad Politécnica de Valencia (campus de Alcoy). El proyecto mencionado nace de la idea del profesor Raúl Llopis Llinares de crear una interfaz de acceso Web al hardware de red que funcionase sobre un sistema de bajo coste. El cual debía proporcionar un acceso sencillo a sus alumnos para la configuración de los distintos dispositivos.

El contenido de la memoria describe el proceso de desarrollo e implantación de una aplicación y su integración con un hardware de bajo coste. Asimismo, se detallan tanto los objetivos y motivaciones perseguidos con el proyecto como las problemáticas y soluciones que se han debido afrontar para completarlo, siguiendo una planificación y unos requisitos.

A pesar de que el proyecto se desarrolla sobre una interfaz web, abarca diversas disciplinas y se relaciona a la perfección con diversas asignaturas de carácter general impartidas en el grado y con las cursadas en la especialización del alumno: Ingeniería De Computadores.

En concreto, podríamos dividir las relaciones del proyecto con las asignaturas cursadas en el Grado en Informática en 3 grupos:

1º-Desarrollo Web. Donde encontraríamos asignaturas como: Tecnología de sistemas de información en la red, Interfaces persona computador, Bases de datos y sistemas de información. Por su relación tanto en el diseño de la interfaz y su usabilidad como en la programación de nuestra aplicación web.

2º-Planificación de proyectos. En este caso, la relación está ligada directamente con el apartado “3.2-Planificación”, donde se recurre a los conocimientos, utilidades y recursos de las asignaturas: Ingeniería del Software y gestión de proyectos.

3º-Programación e interacción con hardware. Sería en este apartado donde encontraríamos asignaturas como concurrencia, programación, diseño de sistemas operativos, arquitecturas avanzadas. Debido en gran medida a la programación de código Bash embebido sobre código Python, a la implementación de soluciones particulares mediante websockets, serial-workers y a la relación de todo ello con el control de elementos hardware.

2-Marco Técnico

A continuación, se detallarán las tecnologías, lenguajes de programación y herramientas utilizadas a lo largo del proyecto.

2.1-Python

Lenguaje de alto nivel, orientado a objetos y a la programación de aplicaciones multipropósito, se pueden crear desde aplicaciones de escritorio hasta aplicaciones Web o scripts. El fuerte del lenguaje reside en ser un lenguaje de programación multiplataforma, además de ser interpretado y usar un tipado dinámico de datos.

Actualmente existen dos vertientes de desarrollo en Python, según la versión utilizada: Python 2.7 y Python 3.5. Python 3.0 fue lanzado en 2008 y la versión final de 2.X fue la versión 2.7 que vio la luz en 2010, pero con la declaración de fin de vida. Esto se traduce en que cualquier mejora, actualización o futuro desarrollo de bibliotecas estará solo disponible para la versión 3.5.

La problemática entre la elección de una versión u otra se debe en gran parte a que no son compatibles entre ellas, cualquier aplicación programada sobre 2.7 no podrá ser interpretada por Python 3.5 de forma nativa.

En el proyecto que nos ocupa se ha desarrollado con la versión de Python 3.5. La decisión se basaba en gran parte por la estabilidad del proyecto a largo plazo y por su capacidad de actualización, dotando el proyecto de durabilidad en el tiempo sin estar sujeto a cambios en el software de base que implicarán un estancamiento o un mal funcionamiento en el caso de actualización.

En lo referente a la programación Python en el proyecto, se ha utilizado en las siguientes áreas de la aplicación:

-Framework Django

-Web Server Tornado

-Websockets

-Workers

-Integración con lenguaje Bash para la gestión de comunicaciones Serial.

2.2-Django framework

"Django es un framework de desarrollo web de código abierto, escrito en Python, que respeta el patrón de diseño conocido como Modelo–vista–controlador. Fue desarrollado en origen para gestionar varias páginas orientadas a noticias de la World Company de Lawrence, Kansas, y fue liberada al público bajo una licencia BSD en julio de 2005; el framework fue nombrado en alusión al guitarrista de jazz gitano Django Reinhardt. En junio de 2008 fue anunciado que la recién formada Django Software Foundation se haría cargo de Django en el futuro." (Wikipedia)

La anterior cita refleja el modelo de desarrollo web seguido por Django (Modelo-vista-controlador) y que a su vez se ha seguido en este proyecto, pero con pequeños cambios en el controlador y la vista debido a la inserción del servidor web Tornado.

A su vez, el presente framework ofrece una solución simple en cuanto al desarrollo rápido de proyectos que contengan diversas aplicaciones, lo que permite modularizar un proyecto de cierta envergadura en multitud de pequeñas aplicaciones y que en conjunto ofrecen la misma funcionalidad. Esto en gran parte se debe a las herramientas que incorpora:

-un ORM (object – relational mapping)

-API de base de datos

-Sistema de plantillas ("templates"), estas se pueden usar de forma simple o mediante herencia.

-Un servidor web ligero para las tareas de depuración y pruebas.

-Sistema de vistas

-Extensión del sistema de autenticación

Estas son solo algunas de las herramientas utilizadas, pero Django ofrece muchas más, nos hemos limitado a las más importantes y a las que se adecuan al proyecto que nos ocupa.

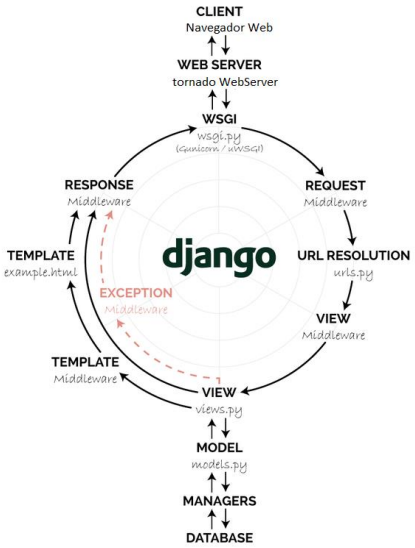

FIGURA - 1

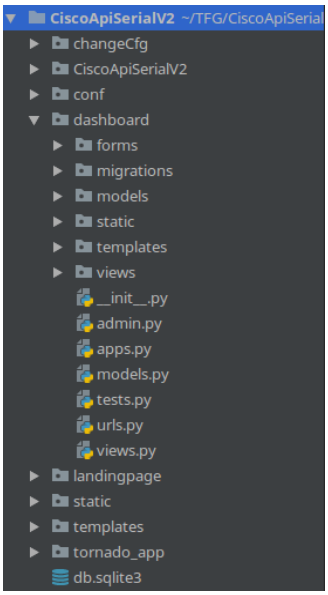

FIGURA - 2

Las figuras anteriores 1 y 2, se corresponden con el ciclo petición respuesta de django y el proyecto implementado. Se puede apreciar la modularidad del proyecto y de las distintas aplicaciones. Aunque solo se muestra desplegada la aplicación "Dashboard" que se correspondería con nuestra ventana principal en la web, podemos relacionar el ciclo petición-respuesta de django con cada módulo de la figura 2.

2.3-Bash

"Bash (Bourne again shell) es un programa informático, cuya función consiste en interpretar órdenes, y un lenguaje de consola. Es una shell de Unix compatible con POSIX y el intérprete de comandos por defecto en la mayoría de las distribuciones de GNU con Linux, además de macOS. También se ha llevado a otros sistemas como Windows. Su nombre es un acrónimo de Bourne-Again Shell ("shell Bourne otra vez") –haciendo un juego de palabras (born-again significa "nacido de nuevo") sobre el Bourne shell (sh), que fue uno de los primeros intérpretes importantes de Unix." (Wikipedia)

Bash, como se ha descrito en las líneas anteriores, es un intérprete y un lenguaje de consola. En nuestro caso es de gran utilidad debido en gran medida a la potencia de poder controlar el hardware de red a través de una ventana de comandos. En el proyecto, Bash ha sido utilizado ampliamente para el control, apertura y cierre de los distintos enlaces serie. Todo realizado desde Python y en combinación de ttyd, de forma que permite la gestión del hardware desde el servidor web mediante Python.

2.4-TTYd

Este software de línea de comandos nos ofrece la compartición de los terminales o shells GNU/Linux sobre la web. Lo que nos permite lanzar distintos programas de forma local, pero con acceso web. Es decir, podemos ejecutar ttyd en conjunto con un servidor web, para ofrecer vía http una interacción directa entre el cliente conectado a la web y el programa que se ejecuta en el servidor de forma local. A continuación, se muestra una imagen de su funcionamiento básico:

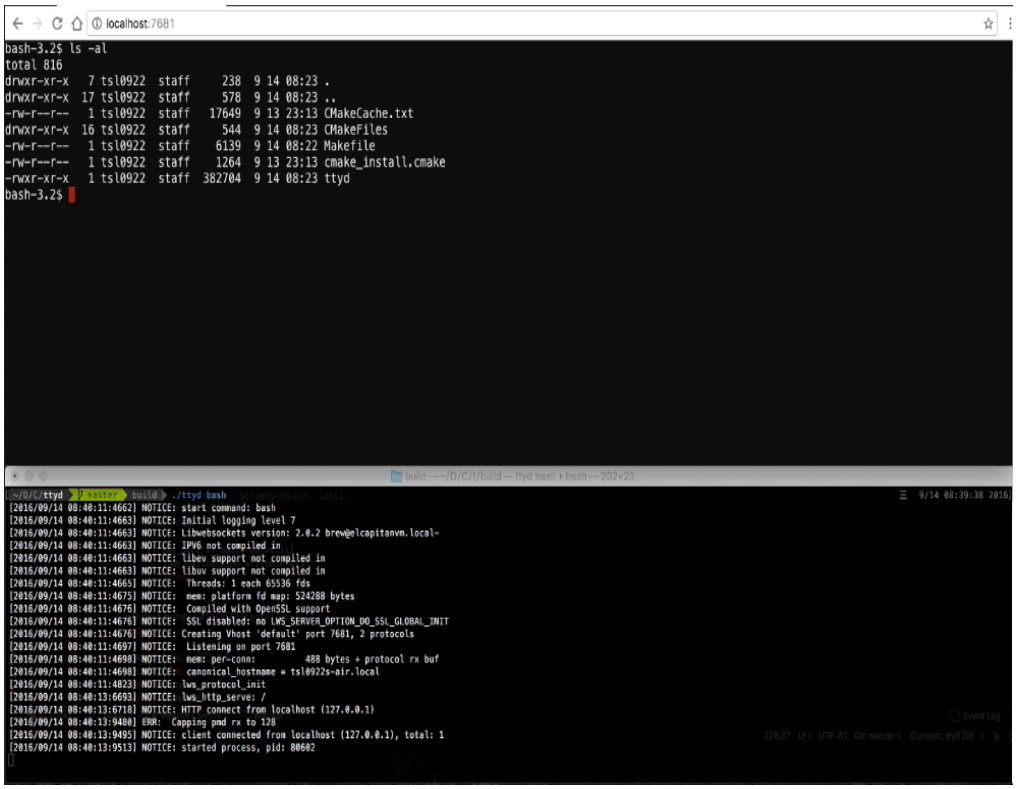

FIGURA - 3

En la figura 3, se puede observar la ejecución de ttyd sobre un intérprete Bash en la zona inferior, es la ejecución que veríamos en el servidor. En la zona superior se muestra su interacción desde la web, y operaríamos con total normalidad como si estuviéramos en el propio servidor.

2.5-Tmux

"Tmux es un multiplexor de terminal para sistemas tipo Unix, similar a GNU Screen o Byobu que permite dividir una consola en múltiples secciones o generar sesiones independientes en la misma terminal. "(Wikipedia)

Este multiplexor de ventanas hace posible la división de una consola en múltiples, ofreciendo una misma utilidad en diversas ventanas de forma síncrona. Este factor hace posible la emulación de ventanas mediante la utilidad ttyd y la separación de estas para ser servida a diversos usuarios.

En la figura siguiente se muestran dos terminales durante la ejecución de Tmux, aunque parecen el mismo, cada uno se ejecuta en un proceso independiente, pero ejecutando la misma utilidad, es decir, los dos procesos son independientes, pero ambos están enlazados con el mismo proceso hijo de forma que la información insertada en la ventana izquierda se replicara sobre la ventana derecha como si de un espejo se tratase.

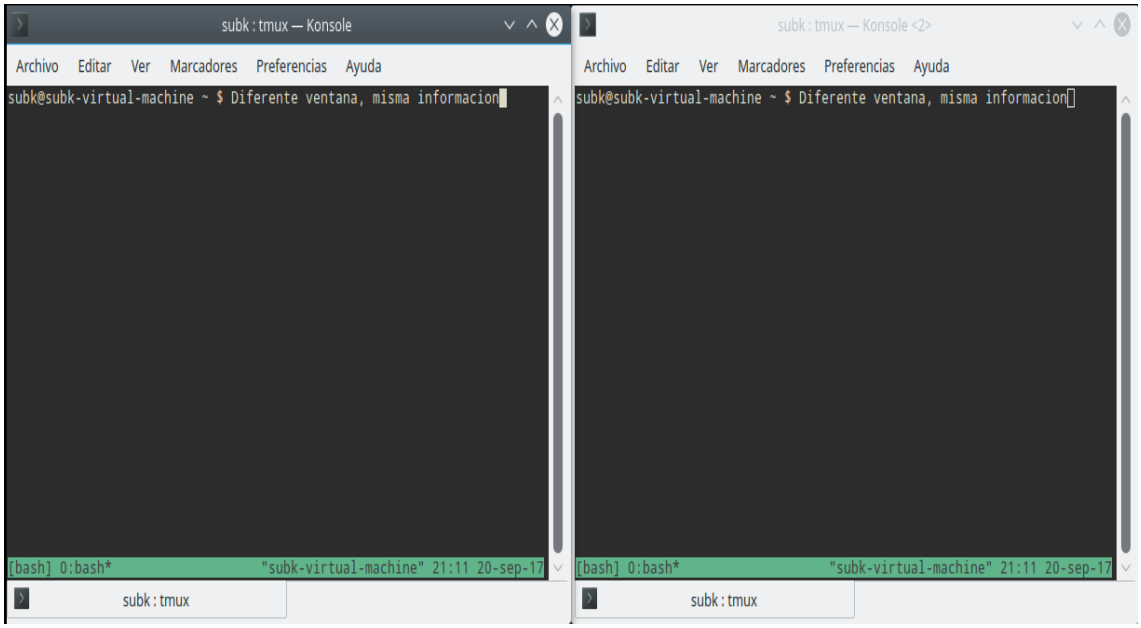

FIGURA - 1

2.6-IDE (Integrated Development Environment)

Debido a la extensión de la aplicación, se ha utilizado un Entorno de Desarrollo Integrado. Lo que permite mayor agilidad y rapidez en el desarrollo de la aplicación frente a editores de texto convencionales sin funciones ampliadas.

Como ejemplo, el IDE nos muestra errores varios en el código, desde errores gramaticales a errores por la instanciación de un objeto o la declaración errónea de una variable. Además de ofrecer diversas funciones para la ejecución del servidor web, su testeo y su depuración.

Asimismo, se ha utilizado PYCHARM. IDE desarrollado por JetBrains y al cual se tiene acceso mediante la licencia de estudiante proporcionada por la Universidad Politécnica de Valencia. Este IDE es uno de los más extendidos para la programación de aplicaciones web utilizando el framework Django. Entre sus principales características nos encontramos:

-Editor de código inteligente

-navegación inteligente de código

-Reestructuraciones rápidas y seguras

-Herramientas de desarrollo incorporadas

-Depuración y pruebas.

-VSX – Desarrollo y desarrollo remoto

-Herramientas de base de datos

-Frameworks web en Python

2.7-Modelo – Vista – Controlador

"El Modelo–vista–controlador (MVC) es un patrón de arquitectura de software, que separa los datos y la lógica de negocio de una aplicación de la interfaz de usuario y el módulo encargado de gestionar los eventos y las comunicaciones. Para ello MVC propone la construcción de tres componentes distintos que son el modelo, la vista y el controlador, es decir, por un lado, define componentes para la representación de la información, y por otro lado para la interacción del usuario. Este patrón de arquitectura de software se basa en las ideas de reutilización de código y la separación de conceptos, características que buscan facilitar la tarea de desarrollo de aplicaciones y su posterior mantenimiento" (Wikipedia)

El modelo descrito en las líneas anteriores es el utilizado en el framework django. Al utilizar una separación por capas, se facilita el desarrollo de soluciones eficientes con una alta reusabilidad y portabilidad. En la figura siguiente se muestra el ciclo del modelo presentado.

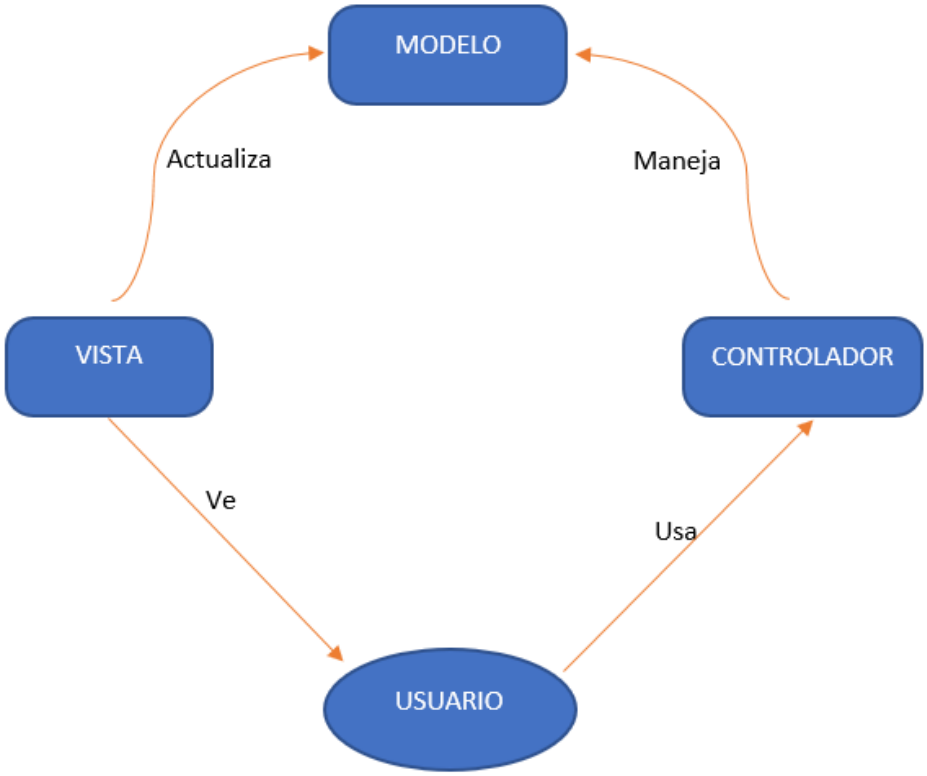

FIGURA - 4

2.8-Workers

En nuestro caso, utilizamos una implementación particular de "worker" sobre Python. Un "worker" es un proceso que se ejecuta en segundo plano y que realiza tareas largas o de bloqueo. Su uso en el proyecto se relacionaba con los "websockets", ambos eran usados en la primera versión de la aplicación para la comunicación entre la web, los clientes, el servidor y los elementos de red.

En el Anexo "18.2-Serial worker para comunicación rs-232" se ha incluido una implementación particular de un "worker" utilizado en las primeras versiones de la aplicación.

2.9-Websockets

"WebSocket es una tecnología que proporciona un canal de comunicación bidireccional y full-duplex sobre un único socket TCP. Está diseñada para ser implementada en navegadores y servidores web, pero puede utilizarse por cualquier aplicación cliente/servidor."(wikipedia)

Como se ha comentado en el subapartado anterior, el uso de los websockets está relacionado con el de los "workers", pero en este caso los websockets se encargaban de la comunicación entre el cliente y el servidor.

El websockets implementado inicialmente consistía en una interfaz Web en HTML y código JavaScript para la captura de los comandos introducidos por el cliente.

3-Implementación de la solución

3.1-Planteamiento de la Solución

Debido a la situación y problemática planteada en el apartado "1.1.1-Contexto y análisis de la situación", se decide plantear un proyecto que pudiera subsanar todos los factores limitantes en la interacción entre los alumnos y los elementos de red.

La opción abordada desde el principio fue la implementación de una interfaz web, que facilitara el acceso y control desde cualquier ordenador del laboratorio. La decisión se debió en gran medida a que una aplicación web garantiza la independencia del sistema operativo del cliente, necesitándose tan solo un navegador web y reduciendo el número de errores y puntos de fallo. De esta forma podemos desarrollar una aplicación escalable y portable de una forma sencilla.

La solución a la problemática planteada se realizó generalizando el problema en su conjunto y una vez definido ir reduciéndolo a problemas más pequeños y más sencillos de implementar.

Siguiendo esta metodología, se definieron las cinco abstracciones que definen el conjunto del proyecto:

1 -Cliente

2 -Comunicaciones entre Servidor y clientes

3 -Servidor para la implementación de la aplicación

4 -Comunicaciones entre el servidor y los dispositivos Hardware.

5 - Hardware de red

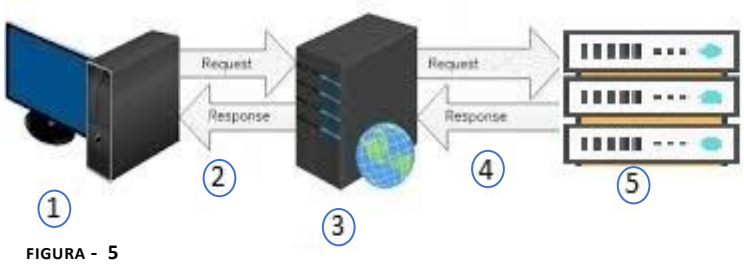

FIGURA - 5

3.2-Planificación

En las líneas siguientes se describe la planificación de todo el proyecto, sus tareas, el calendario que fue planteado inicialmente para todas ellas, el modelo de planificación y desarrollo de software utilizado.

El modelo utilizado en el proyecto se basa en el desarrollo en cascada retroalimentado:

"En Ingeniería de software el desarrollo en cascada, también llamado modelo en cascada (denominado así por la posición de las fases en el desarrollo de esta, que parecen caer en cascada "por gravedad" hacia las siguientes fases), es el enfoque metodológico que ordena rigurosamente las etapas del proceso para el desarrollo de software, de tal forma que el inicio de cada etapa debe esperar a la finalización de la etapa anterior. Al final de cada etapa, el modelo está diseñado para llevar a cabo una revisión final, que se encarga de determinar si el proyecto está listo para avanzar a la siguiente fase. Este modelo fue el primero en originarse y es la base de todos los demás modelos de ciclo de vida." (Wikipedia)

La definición anterior, es la correspondiente al modelo de desarrollo en cascada simple, pero debido a que los proyectos no suelen seguir el modelo secuencial propuesto, se ha implementado el modelo mostrado en la figura-6, que incorpora la realimentación entre fases evitando así el retraso en la obtención de una versión operativa y los estados de bloqueo.

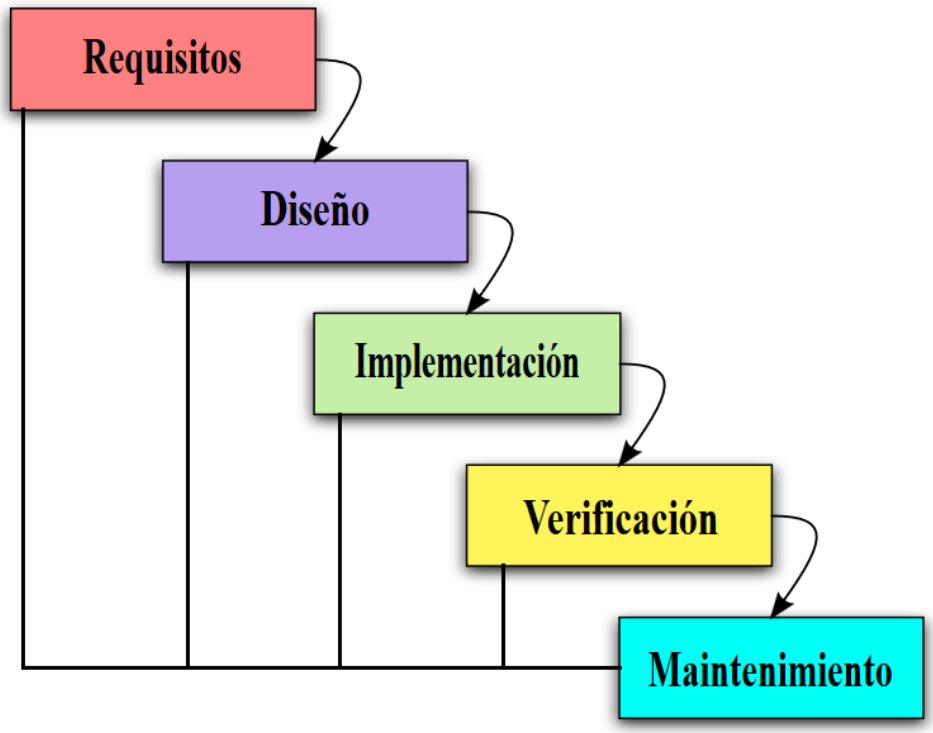

FIGURA - 6

Este modelo, a su vez, se implementa siguiendo los cinco pasos o etapas que se pueden observar en la figura anterior.

1-Planificación o Análisis de requisitos: Fase en la que se examinan las necesidades de los usuarios finales y la funcionalidad a cubrir por el software que se va a desarrollar.

2-Diseño: Modelado del proyecto, separando los elementos por unidades funcionales, creando a su vez elementos más pequeños y sencillos, de forma que sean fácilmente implementables.

3-Desarrollo e implementación: Fase de implementación de código.

4-Verificación: Etapa donde se realizan las pruebas oportunas para corrobora que el software no produzca fallos.

5-Mantenimiento: Etapa crítica, donde normalmente se destinan más recursos, debido a que muchas veces es en esta fase donde se determina que el software no cumple con las expectativas o con la funcionalidad deseada.

En la tabla siguiente, se muestran las tareas definidas para el desarrollo del proyecto, así como su duración estimada, y sus respectivas fechas de inicio y fin. Cabría destacar que las duraciones de las dos últimas tareas “Desarrollo en paralelo con Django y Tornado” y “Pruebas y Correcciones” se realizan de forma paralela, ya que al modular el proyecto y desarrollar por partes se intenta subsanar cualquier posible fallo al final de cada etapa de desarrollo, evitando así llegar a un punto en el proyecto donde la acumulación de errores ocasione la imposibilidad de avance.

Nombre de las Tareas	Duración Estimada Inicial	Fecha Inicio Previsto	Duración Final	Fecha Inicio Final
Planteamiento de la idea con el tutor responsable del TFG	3 días	09/01/17	7	18/01/17
Estudio de validez de la idea como TFG	4 días	12/01/17	4	24/01/17
Planteamiento inicial del problema a desarrollar	12 días	18/01/17	14	28/02/17
Separación del proyecto en módulos diferenciados por funcionalidad	10 días	03/02/17	12	16/03/17
Estudio Hardware	6 días	27/02/17	6	24/03/17
Estudio de las tecnologías a implementar	6 días	07/03/17	6	03/04/17
Análisis de los distintos framework web y web servers que mejor se adapten al proyecto	11 días	22/03/17	13	20/04/17
Definición del modelo de datos	10 días	05/04/17	10	04/05/17
Definición de vistas	7 días	14/04/17	9	17/05/17
Desarrollo en paralelo con Django y Tornado	33 días	31/05/17	57	04/08/17
Pruebas y Correcciones	28 días	08/06/17	45	06/09/17

TABLA 1

Como se puede observar, el proyecto se planifico con una duración de 5 meses, empezando a principios de enero y finalizándolo a principios de junio. Los plazos estimados a priori parecían suficientes y realistas, invirtiendo bastante tiempo en la labor de planificación para ahorrar tiempo en las fases de desarrollo y depuración. A continuación, se muestran los diagramas de Gantt correspondientes al estado inicial previsto del proyecto y el resultado obtenido finalmente.

3.2.1-Diagrama de Gantt inicial

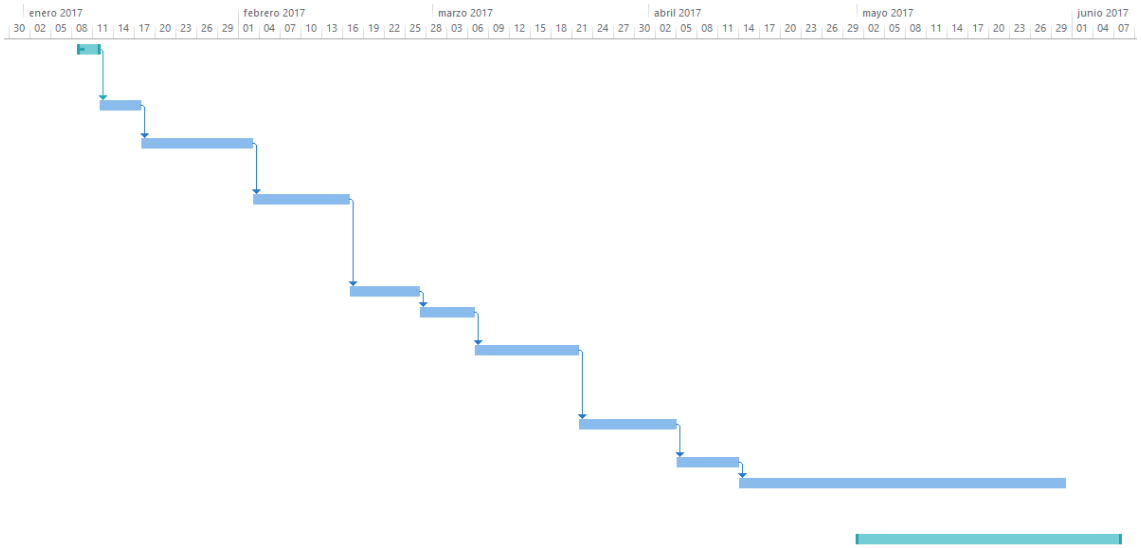

FIGURA - 7

3.2.2-Diagrama de Gantt final

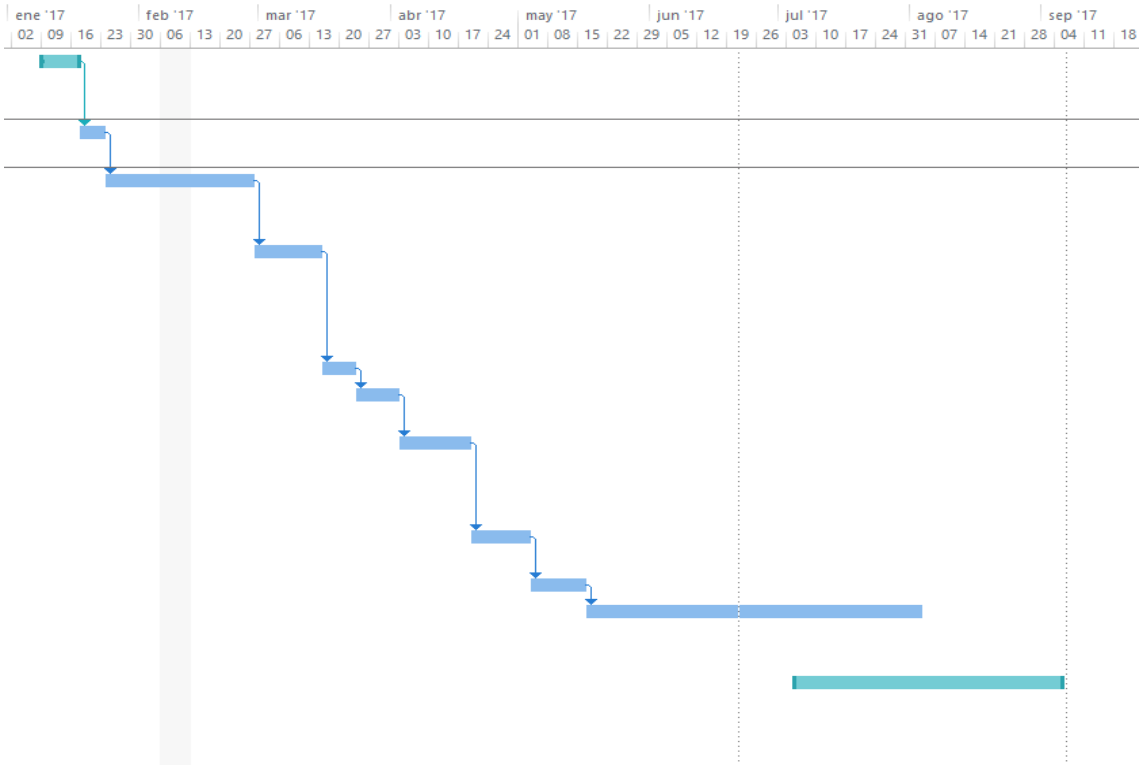

FIGURA - 8

Mediante la apreciación de los dos diagramas de Gantt, se puede visualizar la variación entre el diagrama inicial (Figura.7), planteado con fechas estimadas, respecto del diagrama resultante(Figura.8) tras el desarrollo del proyecto. El proyecto se demoró más de lo esperado, rebasando cualquier cálculo esperado o estimado con anterioridad, debido en gran medida a los siguientes factores:

-Falta de planificación.

-Mala gestión del tiempo.

-Comunicación insuficiente a la hora de definir la funcionalidad y requerimientos, que debían implementarse en la aplicación.

-Carencia de conocimiento en distintas áreas.

Los factores anteriormente comentados y que no fueron predichos en las fases de planificación como debería haber ocurrido, abarcan distintos ámbitos.

Por una parte, nos encontramos ante la falta de disponibilidad horaria por parte del alumno debido al trabajo extracurricular, y que no fue previsto en la planificación a la hora de definir las tareas y el tiempo asignado a estas. El retraso por este factor se puede observar en el área sombreada de la figura 4.

Por otra parte, y no menos importante, nos encontramos ante fallos en distintas áreas de la planificación. Estos fallos también varían en su ámbito de aplicación, desde fallos simples en la integración de las aplicaciones que conforman el proyecto a no predecir la falta de conocimiento en distintas áreas como: programación de websockets, características especiales de programación con el framework Django, funcionamiento particular del servidor web Tornado, etc.

Finalmente, una mala comunicación a la hora de definir la funcionalidad imprescindible que debía incorporar el proyecto, ocasiono la mayor demora en la entrega de este. Dicha funcionalidad se definió en una fase muy avanzada del proyecto y con una versión operativa. Este factor ocasiono una reestructuración completa del proyecto a nivel de funcionalidad y usabilidad, lo cual derivo en cambios a nivel de hardware y software, forzando el abandono de la solución implementada y de la plataforma elegida para el despliegue de la aplicación: RaspberryPi3.

3.3-Estudio Hardware

Debido a la naturaleza del proyecto se requería de la compra de un sistema embebido para las funciones de interconexión con el hardware de red y alojamiento del servidor web, se realizó la siguiente comparativa entre las diferentes plataformas más extendidas de sistemas embebidos de propósito general y bajo coste para determinar cuál se adecuaba mejor al proyecto.

Nombre	**Procesador**	**Ram-Ghz**	**NºPuertosUsb**	**Almacenamiento**	**Precio-$**
Banana Pro	1 GHz ARM Cortex-A7 Dual-core	1 GB DDR3 DRAM (shared with GPU)	2 USB 2.0 host, 1 USB OTG	SD slot (maximo 64 GB)	60.69
Rpi 3	Quad Core 1.2GHz Broadcom BCM2837 64bit CPU	1GB RAM	4	Micro-SD	38
Odroid-C1	Amlogic S805 SoC 4 x ARM® Cortex ®-A5 1.5GHz	1GB	4	Micro-SD	57.25
Odroid-U3	**Samsung Exynos4412 Prime Cortex-A9 Quad Core 1.7Ghz**	(2GB	4		88
Orange Pi	Quad-core Cortex-A7	1GB DDR3	4	Micro-SD	49,95
Beelink-X2	Allwinner **H3** Quad-core Cortex-A7	DDR3 **1GB**	2	Micro-SD	30€

TABLA 2

A la tabla anterior, se le suma el análisis de rendimiento o benchmark mediante la herramienta sysbench. Este análisis evalúa la cpu, memoria, velocidad de escritura y velocidad de lectura, de forma que los siguientes gráficos muestran una idea general de la capacidad de las plataformas, dichos gráficos se miden en unidades de milisegundos, por lo que cuanto más próximo sea a 0 mejor será una plataforma respecto a otra.

Plataforma	Cpu (ms)	Memoria(ms)	Escritura(ms)	Lectura(ms)
Banana Pro	158,5294	1,6089	0,0679	0,2721
Raspberry pi 3	45,6987	0,7053	0,2417	0,1951
Odroid-C1	55,239	0,6521	0,8662	0,1319
Odroid-U3	34,1491	0,4376	0,0644	0,1203
Orange-Pi-One	58,6344	0,5695	0,0624	0,1757
Beelink-x2	67,3794	0,555	0,2357	0,1185

TABLA 3

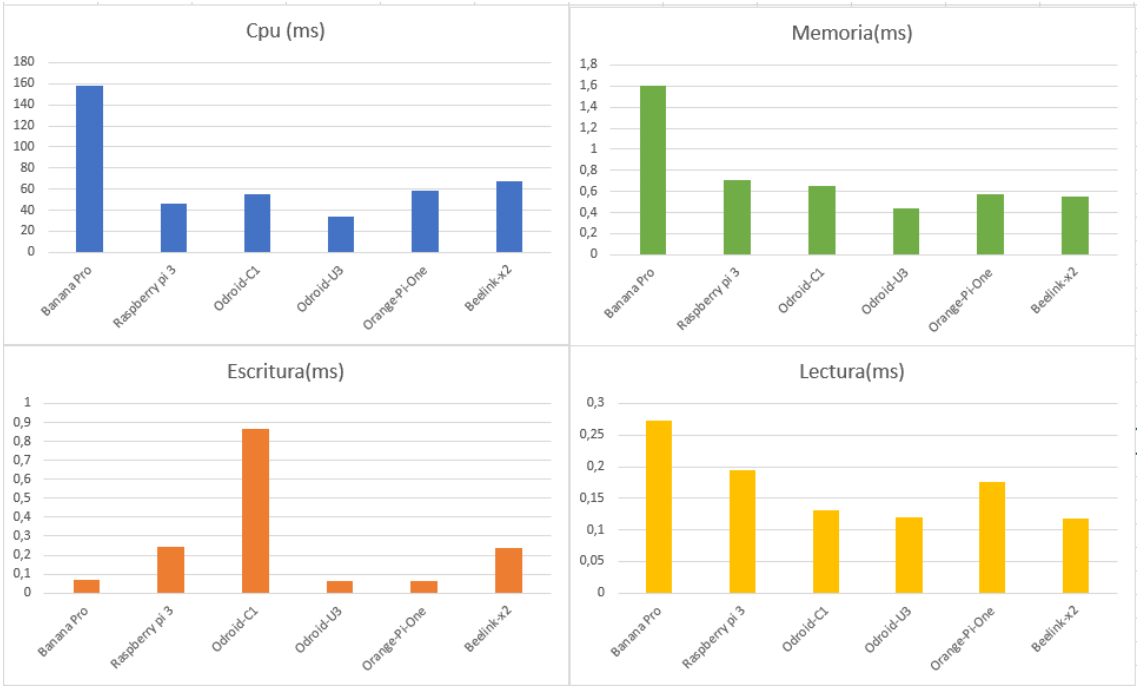

FIGURA - 9

Asimismo, todos los benchmarks fueron ejecutados sobre la totalidad de núcleos de cada plataforma, por lo que hay variaciones respecto a la ejecución de un código particular en “single-core” respecto a su ejecución en “multi-core”. Además, se han descartado del análisis la evaluación de puertos de entrada y salido como Gpio y características hardware específicas de cada plataforma ya que carecían de relevancia para la ejecución del proyecto.

Tras la evaluación de los datos, la decisión final previa a la reestructuración del proyecto fue la de utilizar la plataforma Raspberry pi 3 por las siguientes razones:

-Precio/Prestaciones. A pesar de no ser la plataforma más barata, se puede comprar por vendedores oficiales tanto en España como en Europa con sus respectivos certificados de calidad, emisiones electromagnéticas y su correspondiente garantía.

-El hecho de haber realizado pruebas con anterioridad sobre esta plataforma (en diferentes versiones) en asignaturas como "Arquitecturas Avanzadas" y "Diseño de Sistemas Operativos", lo cual facilita la integración del desarrollo con la plataforma al conocerla previamente y tener una base de conocimiento solida en esta plataforma en particular.

-La comunidad. Si bien es cierto que hay muchas comunidades, foros y webs donde se puede encontrar información sobre las plataformas evaluadas. Este es el punto fuerte de la plataforma elegida, ya que tiene una de las mayores comunidades de "makers" lo cual facilita la búsqueda de información, proyectos relacionados y depuración de errores.

3.4-Estudio Software

Debido a la naturaleza multidisciplinar del proyecto se realizó un estudio del software requerido para abordar las diferentes problemáticas descritas en el apartado "1.1.1-Contexto y análisis de la situación", en concreto se abordaron las siguientes:

1 -Cliente.

2 -Comunicaciones entre la Web y nuestro sistema embebido.

3 -Comunicaciones entre nuestro sistema embebido y los dispositivos Hardware.

Uno de los objetivos del proyecto, era adquirir experiencia en la programación de aplicaciones Web con el framework Django, se partió de esta base para la búsqueda del software que se pudiera integrar en Django y preferiblemente usando código Python, para minimizar los posibles errores y puntos de fallo ocasionados al utilizar un lenguaje sobre otro.

Tras la definición del desarrollo con Django, acto seguido se estudiaron el punto dos y tres, evaluando las soluciones particulares a estos puntos, pero teniendo en cuenta el conjunto de ellos. Esto se debe principalmente a que la elección particular de una solución condiciona el resto de soluciones

La comunicación entre el servidor y los distintos dispositivos, se debía realizar vía conexión serie (RS232) y se identificaron las posibles soluciones, para después evaluar mediante pruebas parciales e implementaciones particulares cuál de estos desarrollos era el que mejor se adaptaba a nuestras necesidades.

En primer lugar, se investigaron las siguientes soluciones para la comunicación serie entre servidor y elementos de red:

-Uso de software específico ejecutable por ventana de comandos para estas tareas:

-putty -minicom

-miniterm -ctkterm

-picocom -moserial

-cutecom -qterm

-Implementación de una clase en Python, utilizando la librería pyserial para realizar la comunicación entre nuestro servidor y los elementos de red.

-Uso de subprocesos en Python en combinación con PySerial (librería en Python que define las funciones necesarias para la comunicación serial).

-Identificación del número de elementos de red mediante Bash

En segundo lugar, se investigó que posibilidades existían en la comunicación cliente-servidor, mediante los agentes de mensajes ("message broker") y comunicaciones asíncronas. Relacionando estas con el tipo de servidores evaluados y las comunicaciones mediante websockets.

-Python Celery

-Rabbit MQ

-Django Channels

-Python Redis

Inicialmente, se implementó una solución particular con "websockets" y "serial-workers" como comunicación entre cliente-servidor y servidor-hardware respectivamente, fue una elección obvia una vez descubierta, ya que aportaban un elevado grado de sencillez y facilidad de uso además de una perfecta integración con django y tornado.

Tras los últimos cambios en la funcionalidad requerida para el proyecto, se requería implementar o emular una ventana de comandos similar a las que encontraríamos en sistemas GNU/Linux, pero sobre una interfaz web. Esta ventana de comandos en

conjunto con software que realizara la conexión con los distintos elementos de red proveería la interfaz final con histórico de comandos introducidos y atajos de teclado.

Se determinó que debía investigarse las distintas soluciones sobre emulación de terminales tty que existían sobre sistemas Linux para poder ofrecer una experiencia completa al usuario, ya que la solución implementada aun funcional y operativa, no cumplía con los nuevos requisitos.

Se evaluaron las siguientes soluciones particulares: ttyd, gotty, xterm. Tras realizar pruebas de rendimiento y usabilidad, se integró en el proyecto ttyd ya que es un descendiente directo de xterm implementado con libwebsockets, además ofrecía la funcionalidad deseada para el proyecto.

La elección anterior, propicio el cambio en la plataforma hardware elegida. Debido a la imposibilidad de integrar libwebsockets en su versión 2.3 sobre la plataforma ARM. Se decidió que la usabilidad y funcionalidad aportadas al proyecto merecían el cambio de plataforma y arquitectura de esta.

Así pues, se incorporó el servidor web "Tornado" en combinación con Django porque ofrecía un elevado grado de integración con la primera versión del proyecto desarrollada mediante "websockets" y "workers". Adicionalmente, es un servidor desarrollado en Python y ligero en cuanto al consumo de recursos del servidor, por lo que se determinó mantenerlo como el servidor web tanto para desarrollo como para el entorno de producción.

Como conclusión a este apartado, se aporta un diagrama de las peticiones-respuesta de la aplicación desarrollada, para intentar clarificar el funcionamiento de la aplicación.

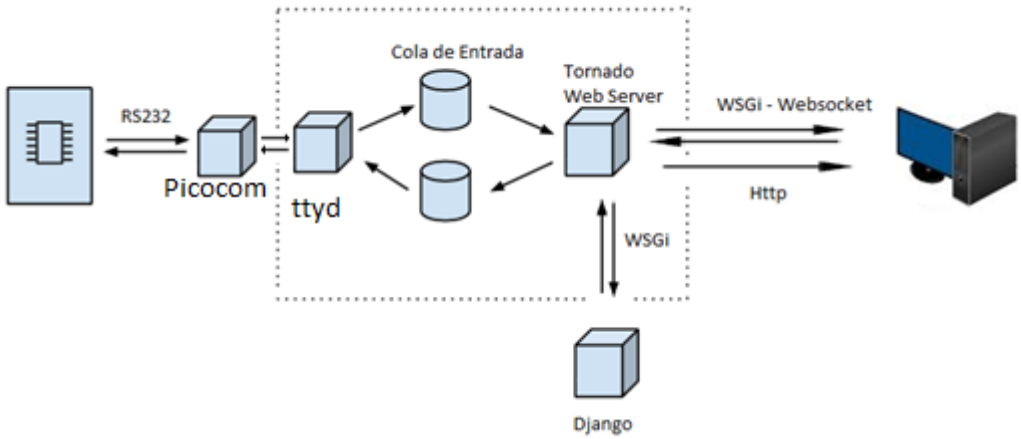

FIGURA - 10

Como podemos observar en la imagen anterior, se define todo el ciclo petición-respuesta, y como opera nuestra aplicación. La utilidad ttyd ejerce de enlace entre aplicaciones mostrando la salida estándar que se obtiene del enlace serie entre picocom y un elemento de red. Esta salida será la mostrada al cliente vía web.

3.5-Análisis de Requerimientos y Casos de Uso

Como se ha podido ver en el apartado planificación, previamente al desarrollo se realizó un análisis de los requerimientos de la aplicación y una descripción de los casos de uso. Destacar que la aproximación seguida es de acuerdo con el objetivo del requerimiento, es decir:

-Requerimientos funcionales

-Requerimientos no funcionales.

3.5.1-Requerimientos funcionales

"Este tipo de requerimientos ofrece una descripción detallada de lo que el sistema debe hacer y en ocasiones lo que no debe hacer, conocidos también como "capacidades". Además, contiene descripciones de los servicios que el sistema debe proveer, como el sistema debe reaccionar ante entradas particulares, como el sistema se debe comportar ante determinadas situaciones…" (Ingeniería del software)

Inicialmente la mayoría de requisitos funcionales, se detallaron en los primeros días del proyecto (cuando era solo una idea) en conjunto con el director del proyecto. Aunque como se ha comentado en otros capítulos algunos cambiaron en una fase avanzada.

-Permitir la creación de nuevas entidades que representen usuarios, tanto por parte de los usuarios como por parte del administrador.

-Asignación de Roles de Usuario (usuario estándar/administrador)

-Apertura y cierre de los canales de comunicación de los elementos hardware

-Posibilidad de apertura de canales de comunicación multiusuarios.

-Registrar los logs de funcionamiento de la aplicación.

-Emulación de ventanas tipo tty, con atajos de teclado para cada dispositivo hardware e histórico de comandos.

-Detección del hardware de red conectado al servidor de forma automática o de forma manual por parte del administrador de la aplicación.

3.5.2-Requerimientos no funcionales

"Los requerimientos no funcionales son los que no están directamente relacionados con los servicios específicos prestados por el sistema para sus usuarios"(Ingeniería del Software)

-Propiedades emergentes del sistema: También conocidos como requerimientos de calidad (rendimiento, seguridad, mantenibilidad…)

-Restricciones en la implementación del sistema: imponen condiciones en la solución.

Este tipo de requerimiento suele ser los más críticos dado que engloban a la aplicación y tienen una repercusión directa en su resultado final. En el proyecto que nos ocupa y debido a su naturaleza "software-hardware" los requerimientos juegan un papel fundamental para el éxito del proyecto, describiendo tanto los requisitos software como hardware y su relación.

-Requerimientos de eficiencia. La eficiencia hace referencia tanto a los requerimientos de rendimiento como de espacio, ya que el proyecto se implementa sobre un sistema embebido, este debe cumplir con un rendimiento eficiente mínimo, pues se dispone de hardware con potencia y espacio limitados.

-Requerimientos de usabilidad. La aplicación debe cumplir con los estándares de accesibilidad, facilitando el acceso a los usuarios de forma rápida y eficiente a los diferentes apartados de la aplicación.

-Requerimientos de interoperabilidad. Estos se defienden en el marco del intercambio de información entre los usuarios, el sistema desarrollado y los sistemas hardware conectados.

-Requerimientos de portabilidad. La aplicación se ha desarrollado sobre una plataforma concreta, pero siguiendo estándares y lenguajes que le permitan mediante pocos ajustes, ser portable entre distintas plataformas.

-Requerimientos de implementación. Estos se basan en el desarrollo de una aplicación por módulos para facilitar la depuración y la escalabilidad.

-Requerimientos de seguridad. Aunque el ámbito de aplicación es local, se deben seguir las pautas de securizacion para llegado el caso, escalar la aplicación a un ámbito mayor.

3.5.3-Casos de Uso

A continuación, se detallan en una tabla todos los casos de uso posibles según el rol de cada usuario. Se incluye un subapartado con los diagramas de casos de uso correspondientes a cada rol.

Funcionalidad	**Anónimo**	**Alumno**	**Profesor**	**Administrador**
Acceso Web (landingpage)	☒	☒	☒	☒
Contacto	☐	☒	☐	☒
Salir de la aplicación	☒	☒	☒	☒
Registro	☒	☒	☒	☐
Login	☐	☒	☒	☒
Visualizar conexiones serial	☐	☒	☒	☒
Utilizar conexiones serial	☐	☒	☒	☒
Creación de usuarios	☐	☐	☒	☒
Asignación de permisos	☐	☐	☒	☒
Creación de grupos	☐	☐	☒	☒
Asignación de usuarios a grupos	☐	☐	☒	☒
Listar usuarios	☐	☐	☒	☒
Listar grupos	☐	☐	☒	☒
Creación de permisos	☐	☐	☐	☒
Cambio de roles de usuario	☐	☐	☐	☒
Listar profesores	☐	☐	☐	☒
Listar Administradores	☐	☐	☐	☒
Eliminación de usuarios	☐	☐	☐	☒
Eliminación de grupos	☐	☐	☐	☒
Actualización Hardware	☐	☐	☐	☒

TABLA 4

3.5.4-Diagramas de casos de uso

Los diferentes diagramas de casos de uso se presentan de forma incremental por su rol, es decir, se irán presentando de menor número de funcionalidades asignadas a un rol, a mayor numero en este orden: Anónimo, Alumno, Profesor, Administrador.

3.5.4.1-Rol Anónimo

Partimos del usuario con los permisos más restrictivos, debido a que se trata de un usuario anónimo, solo se va a permitir el acceso a determinadas áreas y/o funcionalidades como se muestra en el siguiente diagrama:

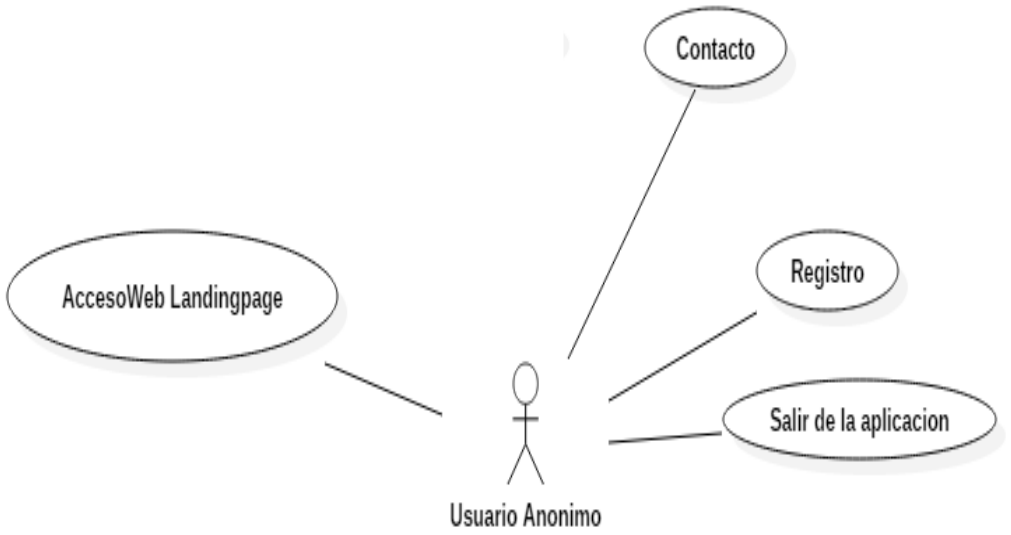

FIGURA - 11

El "Acceso Web Landingpage" se trata de la capacidad de acceso a la página de aterrizaje de la aplicación, es una mera bienvenida que presenta la aplicación. A su vez el usuario tiene la capacidad de registro lo que convertiría el usuario anónimo en alumno. Destacar que el ámbito de aplicación como ya se ha comentado, es meramente académico por lo que no se han contemplado usuarios externos a este ámbito. El resto de funcionalidades son obvias y comunes también al resto de roles.

3.5.4.2-Rol Alumno

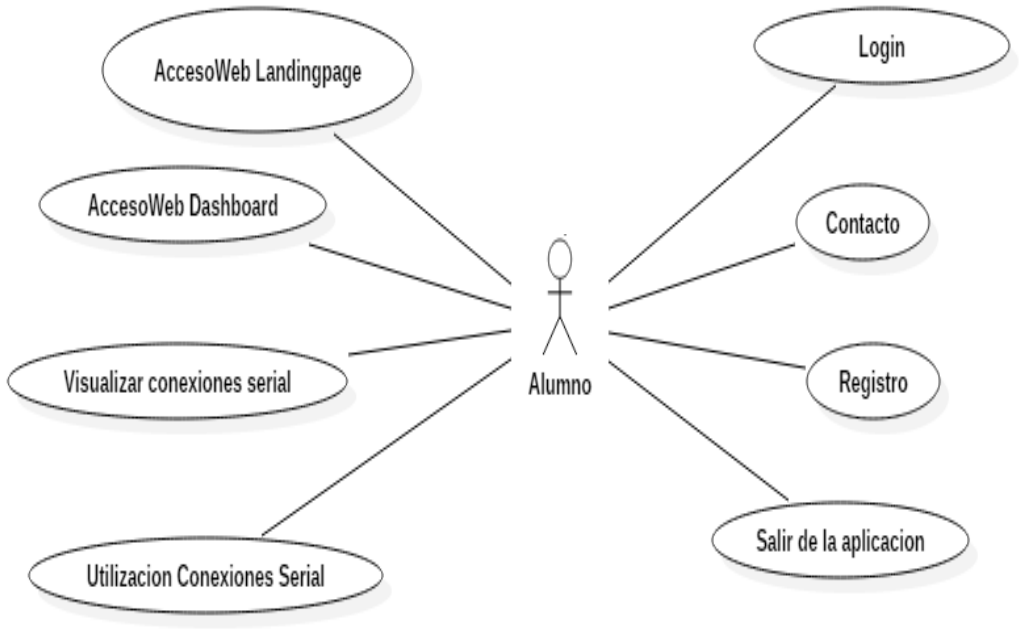

FIGURA - 12

En la anterior figura, nos encontramos el diagrama de uso correspondiente al Rol alumno. Este incorpora cuatro funcionalidades más que el rol anónimo: utilización conexiones serial, visualizar conexiones serial, acceso Web Dashboard y Login.

La visualización de las conexiones serial, así como su uso, habilitan al usuario a que pueda interactuar con el hardware. Dichas funcionalidades "visualizar conexiones serial" y "utilización conexiones serial" se declararon por separado para tener mayor control sobre la aplicación, de forma que un usuario administrador o profesor pueda deshabilitar dichas conexiones a los usuarios por los motivos que estime oportunos.

Por otra parte, el "Login" nos permite entrar en la aplicación como un usuario autenticado (previamente registrado), redireccionándonos y dándonos acceso a la página principal de la aplicación "acceso Dashboard".

3.5.4.3-Rol Profesor

El siguiente rol, Profesor, ya posee la mayoría de funcionalidades, solo superado por el usuario Administrador.

Al igual que anteriormente, las funcionalidades son incrementales por lo que este rol solo posee unas pocas funcionalidades distintas al rol Alumno: Listar usuarios, Listar grupos, Asignación de permisos, Asignación de usuarios a grupos, creación de grupos, Registro.

Este rol encajaría en la categoría de supervisión y control, ya que la gran mayoría de funcionalidades adicionales se basan en la supervisión de los usuarios que utilizan la aplicación.

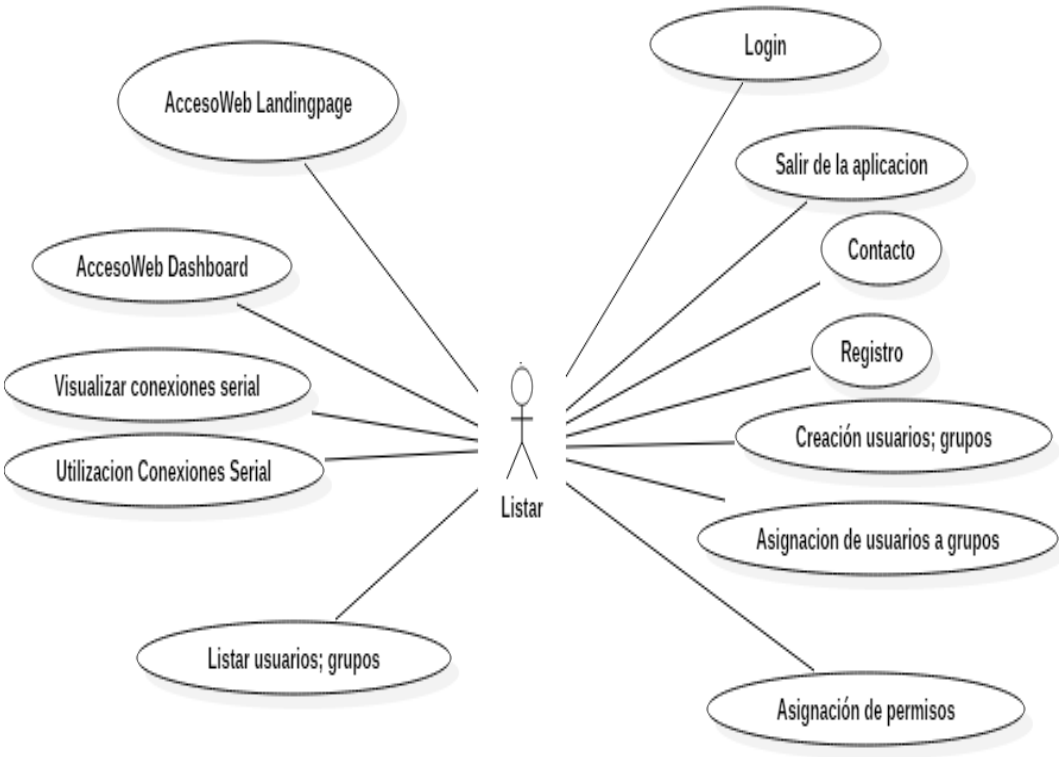

FIGURA - 13

3.5.4.4-Rol Administrador

El rol administrador, es el primero en ser creado en la aplicación. Este usuario difiere del resto en su creación y permisos porque se crea mediante línea de comandos cuando se implementa la aplicación, y no como el resto de roles mostrados que se crean vía registro mediante la interfaz web.

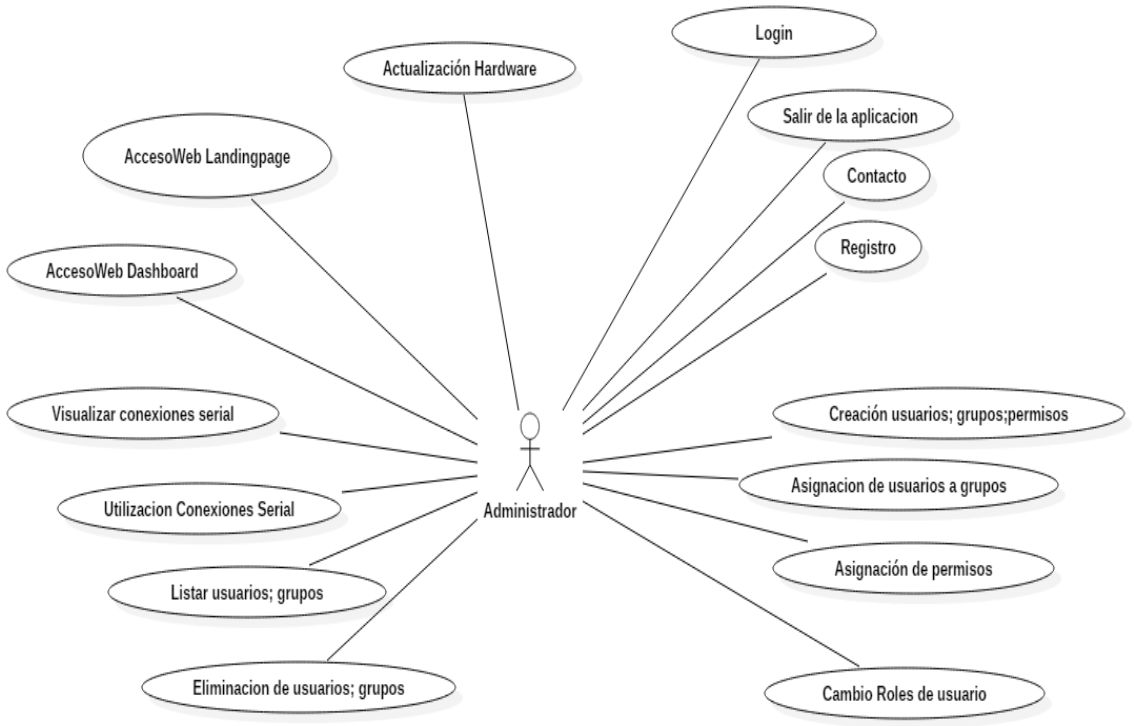

FIGURA - 14

Respecto a las funcionalidades asignadas, el presente rol contiene todas las funcionalidades a excepción del registro (por las razones anteriormente comentadas), pero a su vez, solo difiere en tres respecto al rol profesor: Eliminación de usuarios, Cambio de roles de usuario, Actualización Hardware.

Estas dos primeras funcionalidades dotan al usuario Administrador de la capacidad para eliminar usuarios de la base de datos y del cambio de roles de usuario. Esta última funcionalidad permite el cambio de rol entre alumno-profesor, así un usuario registrado que no sea alumno y si profesor, puede ver cambiado su rol vía Administrador.

Mención especial merece la funcionalidad “Actualización Hardware”, ya que su función es la de actualizar todo enlace serie conectado a nuestro servidor e insertar sus datos en la base de datos, de forma que esté disponible para el resto de usuarios.

3.6-Implementación

Tras todos los antecedentes, problemas y decisiones expuestos, la implementación del proyecto se realizó mediante el IDE Pycharm, utilizando el lenguaje de programación Python en su versión 3.5. El desarrollo de la aplicación global se modulo en distintas aplicaciones diferenciadas por funcionalidad: CiscoApiSerialV2, Dashboard, landingpage, tornado.

-CiscoApiSerialV2 reúne los archivos de configuración de todo el proyecto desarrollado.

-Dashboard representa la ventana principal de interacción entre el usuario y la web. Es en esta aplicación donde se engloba la mayor parte del desarrollo del proyecto puesto que aglutina gran parte de los objetivos descritos y de las funcionalidades implementadas tanto para la interacción usuario-web como servidor-hardware.

-Landingpage es la página de aterrizaje o bienvenida que un usuario no autenticado vería. Esta aplicación sirve de presentación de cara a los usuarios permitiendo el registro para poder ser un usuario autenticado.

-Tornado es el servidor web elegido y utilizado hasta la entrega del proyecto es Tornado, por su desempeño en este tipo de aplicaciones ya que consume pocos recursos y es ideal para una maquina embebida. Una vez iniciado el sistema base donde se despliega el proyecto y tras el arranque, el servidor web se ejecutará permitiendo a los usuarios acceder a la aplicación.

El desarrollo particular de la aplicación sobre el framework Django siguió las etapas de definición de modelos, formularios, vistas y finalmente las plantillas asociadas a las vistas, para las aplicaciones implementadas.

Debido a la idea inicial de desplegar la aplicación sobre un sistema embebido y con fin de reducir el coste de recursos, se eligió SQLite como motor de base de datos para almacenar los datos de usuarios y los datos relativos al hardware de red. SQLite ofrece un buen desempeño en aplicaciones con una base de datos con pocas tablas y se integra perfectamente en el desarrollo con django mediante el IDE nombrado.

Como se ha comentado en el apartado “2.6-Estudio Software”, la funcionalidad requerida en la aplicación se implementó mediante websockets y workers, obteniendo una aplicación viable, pero con un nivel bajo de usabilidad. En la siguiente imagen podemos ver un ejemplo de la interfaz inicial de interacción entre el usuario y el hardware:

FIGURA - 15

La figura anterior muestra la ventana inicial para la interacción del usuario con el hardware. Los comandos debían introducirse en el formulario inferior y él envió se realizaba mediante el botón "send". Esto reflejaba un serio problema de usabilidad debido a la forma en que se deben introducir los comandos, sumado al gran número de comandos que se han de introducir para la configuración de los elementos de red evidenciaba la necesidad del cambio. Además, no se guardaba ningún histórico de comandos introducidos ni se podían utilizar teclas especiales para la interacción.

Tras redefinir los requisitos que debía cumplir la aplicación, se realizó una nueva búsqueda de las tecnologías y software disponible para su realización, como hemos visto, nos decantamos por el uso de ttyd en conjunto con la utilidad Linux Tmux.

El uso de estas utilidades dotaba al proyecto de la capacidad de emular ventanas de comandos vía web. Es decir, la ejecución combinada de Ttyd con Bash, Python y Tmux permite la ejecución en el servidor de cualquier programa y este podrá ser manipulado desde la web en la dirección y puerto que nosotros deseemos. En nuestro caso y tras probar diversas utilidades, la combinación de picocom con Ttyd ofrecía la utilidad deseada. El resultado obtenido puede verse en la figura 16.

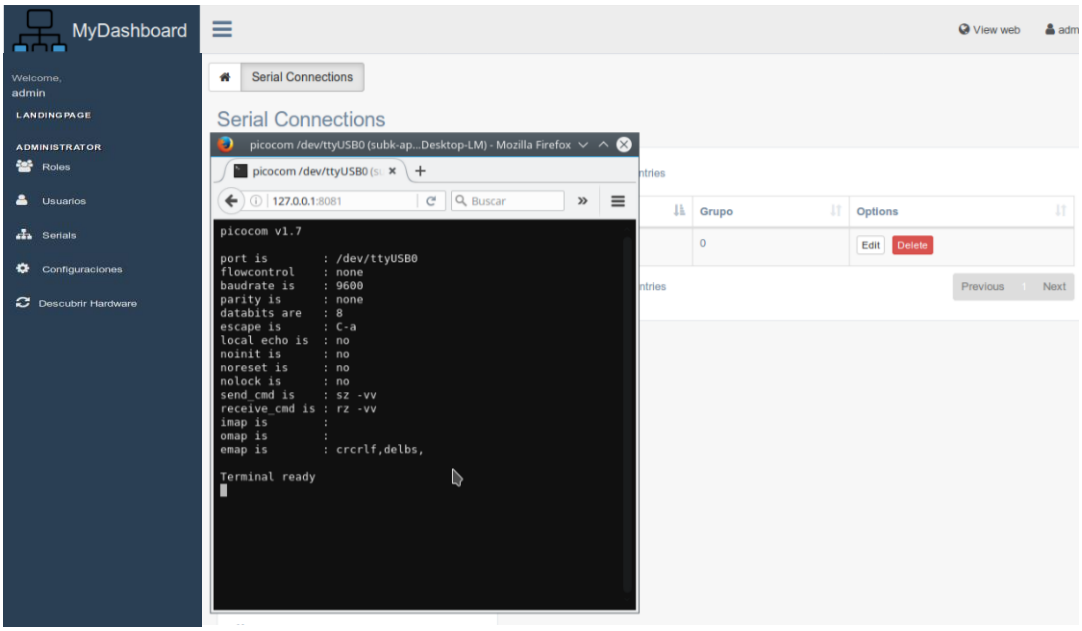

FIGURA - 16

La figura-16 muestra la etapa final de desarrollo del proyecto con la aplicación web en el fondo donde se puede apreciar el "sidebar" con las distintas opciones del administrador implementadas a la izquierda: Roles, Usuarios, Serials, Configuraciones, Descubrir Hardware. La ventana oscura que se aprecia en el centro es nuestra utilidad Ttyd junto con Picocom y Tmux enlazada con un router cisco 2600 preparado para la prueba. Una vez abierta la ventana, nos permite configurar el router como si estuviéramos conectados directamente al puerto serial del dispositivo.

La opción "Roles" permite la asignación de permisos a los distintos usuarios creados o registrados, mientras que la opción "Usuarios" muestra al administrador un listado de todos estos. El enlace "Serials" permite la visualización y de los dispositivos cargados en la base de datos a través del enlace "Descubrir Hardware", a su vez permite la modificación de los parámetros inicialmente cargados y de lanzar las ventanas de administración correspondientes a cada dispositivo hardware conectado. El enlace "Descubrir Hardware", supone el descubrimiento de los elementos de red conectados por puesto serie al sistema. Se realiza mediante Bash y Python para ver cuántos elementos de red hay conectados al sistema, para acto seguido cargarlos en la base de datos y crear una instancia particular de ttyd para cada dispositivo. Para realizar estos pasos, se deben cerrar primero todos los enlaces serial abiertos, por lo que, de haber algún usuario conectado en ese momento, seria desconectado instantáneamente. Es por este motivo por el que solo el administrador debe tener acceso.

Para finalizar, se ha implementado una gestión de usuarios y grupos, para que el administrador/profesor pueda elegir que alumno debe tener acceso a que hardware, o a todos los hardware conectados. Dicha gestión de usuarios proporciona información detallada al administrador de los inicios de sesión en el servidor, las horas, las ultimas conexiones, y el poder de habilitar o deshabilitar usuarios.

3.7-Instalacion de la aplicación web y de sus componentes

En las líneas siguientes, se describe el proceso de instalación de todos los componentes necesarios para hacer funcionar la aplicación desarrollada, de forma que mediante la instalación de las dependencias la aplicación podrá ser portada entre las plataformas compatibles.

Actualización del sistema

Sudo apt-get update && apt-get upgrade

Sudo apt-get dist-upgrade

Instalación del paquete Python en caso de que no estuviese instalado y de las dependencias.

Sudo apt-get install Python

Sudo apt-get install Python-pip

Sudo apt-get install python3-pip

Sudo -H install –upgrade pip

Sudo -H pip3.5 install setuptools

Instalar paquetes desde el fichero proporcionado en el proyecto, el cual incluye:Django, django-bootstrap3, gunicorn, Wheel, tornado.

Sudo -H pip3.5 install -r requeriments

Instalar Liberia pyserial

Sudo -H pip3.5 install pyserial

Instalar ttyd

Sudo apt-get install -y software-properties-common

Sudo add-apt-repository ppa:tsl09222/ttyd

Sudo apt-get update

Sudo apt-get install ttyd

4-Conclusiones

Finalmente, no se ha conseguido cumplir todos los objetivos planteados en el proyecto, puesto que en el inicio del proyecto se plantearon diversos objetivos enfocados en una aplicación web que debía funcionar sobre un sistema embebido y de bajo coste. Bien, esta aplicación se desarrolló y funciono mediante implementaciones particulares de websockets y serial-workers sobre la RaspberryPi, que como hemos podido ver en el estudio hardware era la mejor opción respecto al hardware disponible.

Tras la obtención de una versión operativa de la aplicación y en una fase avanzada se decidió modificar la funcionalidad principal de la aplicación, la cual debería ahora emular una consola estándar similar a las de GNU/Linux y su funcionamiento sobre los equipos de red. Esta variación, forzó el cambio de arquitectura sobre la que debía funcionar, ya que las librerías necesarias (libwebsockets) para el correcto funcionamiento de la aplicación no se encontraban disponibles para la arquitectura ARM, la cual incorporan la inmensa mayoría de hardware embebido de uso general. Todo esto derivo en la demora en la entrega del proyecto debido a la búsqueda de nuevas soluciones que aportaran la funcionalidad demandada, la readaptación del código y las pruebas. Como se ha comentado, el mayor error en el proyecto fue la falta de definición de los requerimientos y unos plazos demasiado optimistas en las tareas definidas.

A pesar de todo, se ha cumplido con los principales objetivos planteados en el trabajo, puesto que se ha desarrollado una aplicación totalmente modular que permite el acceso web a los distintos elementos hardware de red de forma remota y simultanea por parte de los usuarios y solucionado el problema planteado inicialmente. Además, a través del desarrollo e investigación de las soluciones adoptadas se ha adquirido soltura en el desarrollo de aplicaciones Web mediante el framework Django así como en la programación de soluciones en Bash y Python.

Respecto al desarrollo global del proyecto, ha supuesto una gran ventaja la implementación con el framework Django y la utilización del sistema Linux. Debido en gran medida a las grandes comunidades de usuarios y bibliotecas digitales donde se puede encontrar cualquier información para resolver dudas, tanto de implementación como errores de base al no conocer plenamente algunas tecnologías vistas en el proyecto.

4.1-Lineas Futuras

En cuanto al futuro del proyecto, cabe considerar que debido al carácter web de la aplicación y al estar basada en el modelo MVC, puede dar lugar a cambios, mejoras y ampliaciones.

Así pues, se podría mejorar en las siguientes áreas:

-Desarrollo de una solución inicial desde 0 con libwebsockets para un mejor control de la aplicación. Esto añadiría robustez a las comunicaciones si se desarrolla una utilidad específica para este proyecto.

-Mejora en la seguridad de la aplicación.

-Internacionalización de la aplicación.

-Gestión de configuraciones hardware por usuario a través de enlaces serie.

-Instalación de la aplicación en hardware embebido.

5-Bibliografía

5.1-Desarrollo con lenguajes Python y Bash

https://www.python.org/

https://docs.python.org/2/library/subprocess.html

https://docs.python.org/2/library/threading.html

http://kmkeen.com/multithreaded-bash/ Desarrollo Web con Django

5.2-Desarrollo con Django

Https://tutorial.djangogirls.org

Https://djangoproject.org

Djangobook.com

Mastering django core : The complete guide to Django 1.8 lts (Nigel George)

5.3-Hardware

http://www.orangepi.org/

http://www.hardkernel.com

https://www.raspberrypi.org

https://developer.mbed.org

5.4-Web Servers

http://www.tornadoweb.org/en/stable/

http://nginx.org/

http://www.apache.org/

5.5-Websockets

https://github.com/django/channels/

http://python-rq.org/docs/workers/

https://pythonhosted.org/pyserial/

5.6-Emulación Web de ventanas de comandos

https://tsl0922.github.io/ttyd/

https://libwebsockets.org/

https://warmcat.com

http://docs.xtermjs.org/

5.7-Bróker de mensajería

https://redis.io/

https://www.rabbitmq.com/

http://activemq.apache.org/

5.8-Redes de Ayuda

https://stackoverflow.com/

https://unix.stackexchange.com

6-ANEXOS

6.1-Vista implementada mediante el Framework Django

```
class SerialCreateView(LoginRequiredMixin, generic.CreateView):
    model = Serial
    form_class = SerialForm
    template_name = 'dashboard/serial/list.html'

    def dispatch(self, request, *args, **kwargs):
        return super(SerialCreateView, self).dispatch(request, *args, **kwargs)

    def form_valid(self, form):
        return super(SerialCreateView, self).form_valid(form)

    def form_invalid(self, form):
        messages.add_message(self.request, messages.ERROR, _('The Serial could not be created.'))
        return self.render_to_response(self.get_context_data(form=form))
```

```
    def get_context_data(self, **kwargs):
        context = super(SerialCreateView, self).get_context_data(**kwargs)
        context['action'] = 'new'
        context['action_label'] = _('New')
        context['serial_list'] = Serial.objects.all()
        return context

    def get_success_url(self):
        messages.add_message(self.request, messages.SUCCESS, _('The Serial has been created correctly.'))
        return reverse('dashboard:serial_list')


class SerialUpdateView(LoginRequiredMixin, generic.UpdateView):
    model = Serial
    form_class = SerialForm
    template_name = 'dashboard/serial/list.html'

    def dispatch(self, request, *args, **kwargs):
        return super(SerialUpdateView, self).dispatch(request, *args, **kwargs)

    def form_valid(self, form):
        return super(SerialUpdateView, self).form_valid(form)

    def form_invalid(self, form):
        messages.add_message(self.request, messages.ERROR, _('The Serial could not be modified.'))
        return self.render_to_response(self.get_context_data(form=form))

    def get_context_data(self, **kwargs):
        context = super(SerialUpdateView, self).get_context_data(**kwargs)
```

```
        # context['action'] = 'update' || fail when edit the serial form


        context['action_label'] = _('Update')
        context['serial_list'] = Serial.objects.all()


        return context


    def get_success_url(self):
        messages.add_message(self.request, messages.SUCCESS, _('The Serial has been modified correctly.'))
        return reverse('dashboard:serial_list')



class SerialDeleteView(LoginRequiredMixin, generic.DeleteView):
    model = Serial
    template_name = 'dashboard/serial/delete.html'


    def dispatch(self, request, *args, **kwargs):
        return super(SerialDeleteView, self).dispatch(request, *args, **kwargs)


    def get_context_data(self, **kwargs):
        context = super(SerialDeleteView, self).get_context_data(**kwargs)
        context['action'] = 'delete'
        context['action_label'] = _('Delete')
        return context


    def get_success_url(self):
        messages.add_message(self.request, messages.SUCCESS, _('The Serial Connection has been removed.'))
        return reverse('dashboard:serial_list')
```

6.2-Serial worker para comunicación rs232

Esta clase escrita en Python, era la empleada en una fase temprana del proyecto para la comunicación entre el hardware de red y el servidor.

```
import serial

import time

import multiprocessing

import base64


## Change this to match your local settings

SERIAL_PORT = '/dev/ttyUSB0'

SERIAL_BAUDRATE = 9600



class SerialProcess(multiprocessing.Process):

  def __init__(self, input_queue, output_queue):

    multiprocessing.Process.__init__(self)

    self.input_queue = input_queue

    self.output_queue = output_queue

    self.sp = serial.Serial(SERIAL_PORT, SERIAL_BAUDRATE, timeout=1)


  def close(self):

    self.sp.close()


  def writeSerial(self, data):

    self.sp.write(data.encode())

    # time.sleep(1)


  def readSerial(self):

    return self.sp.readline()
```

```
def run(self):

    self.sp.flushInput()

    while True:
        # look for incoming tornado request
        if not self.input_queue.empty():
            data = self.input_queue.get()

            # send it to the serial device
            self.writeSerial(data)
            print("writing to serial: " + data)

        # look for incoming serial data
        if (self.sp.inWaiting() > 0):
            data = self.readSerial()
            #print("reading from serial: " + data)
            # send it back to tornado
            self.output_queue.put(data)
```

6.3-Hilo de ejecución principal del webserver

```
import tornado.web, tornado.ioloop, tornado.websocket, tornado.options, tornado.httpserver
from tornado.options import define, options
import sys
import os
import multiprocessing
import json
import tornado.wsgi
from CiscoApiSerialV2 import wsgi
import serialworker
from CiscoApiSerialV2.settings import STATIC_URL, STATIC_ROOT
import tornado_websockets
import django.core.handlers.wsgi
import subprocess
from subprocess import check_output
from subprocess import Popen, PIPE
from dashboard.views.mydashboard import *



define("port", default=8000, help="run on the given port", type=int)



clients = []
input_queue = multiprocessing.Queue()
output_queue = multiprocessing.Queue()



class IndexHandler(tornado.web.RequestHandler):
   def data_received(self, chunk):
      pass
```

```
    def get(self):
        self.render('index.html')


class SerialDetection(tornado.web.RequestHandler):
    def data_received(self, chunk):
        pass

    # Call subprocess to obtain the number of Serial connections
    p1 = Popen(["ls", "/dev/"], stdout=PIPE)
    p2 = Popen(["grep", "ttyUSB"], stdin=p1.stdout, stdout=PIPE)
    p3 = Popen(["wc", "-l"], stdin=p2.stdout, stdout=PIPE)
    p1.stdout.close()  # Allow p1 to receive a SIGPIPE if p2 & p3 exits.
    outputChild = p3.communicate()[0]

    # Parse string to Int to Delete the additional b' & \n, we only need the number for know how
many dispositives we have.
    OutputChildInt = int(outputChild)
    print(outputChild)

    def get(self):
        self.render('index.html')


class StaticFileHandler(tornado.web.RequestHandler):
    def data_received(self, chunk):
        pass

    def get(self):
        self.render('main.js')
```

```
class WebSocketHandler(tornado.websocket.WebSocketHandler):
    def data_received(self, chunk):
        pass

    def check_origin(self, origin):
        return True

    def open(self):
        print('new connection')
        clients.append(self)
        self.write_message("connected")

    def on_message(self, message):
        print('message', message)

        print('tornado received from client: %s' % json.dumps(message))

        if message == "":
            print("msg vacio")
            input_queue.put('\r\n')
        else:
            print("msg lleno")
            input_queue.put(message)

    def on_close(self):
        print('connection closed')
        clients.remove(self)
```

```
# check the queue for pending messages, and rely that to all connected clients
def checkQueue():
    if not output_queue.empty():
        message = output_queue.get()
        for c in clients:
            c.write_message(message)


def serialDetection():
    # We stop the connections first to not cause any fault when instance new connections
    killSerialConnectionsOppened()


    # Call subprocess to obtain the number of Serial connections
    p1 = Popen(["ls", "/dev/"], stdout=PIPE)
    p2 = Popen(["grep", "ttyUSB"], stdin=p1.stdout, stdout=PIPE)
    p3 = Popen(["wc", "-l"], stdin=p2.stdout, stdout=PIPE)
    p1.stdout.close()  # Allow p1 to receive a SIGPIPE if p2 & p3 exits.
    outputChild = p3.communicate()[0]


    # Parse string to Int to Delete the additional b' & \n, we only need the number for know how
many dispositives we have.
    OutputChildInt = int(outputChild)
    print(OutputChildInt)  # only for see in cli the number of serial connections when one user
calls the function


    instanceConnections(OutputChildInt)
    return OutputChildInt


def killSerialConnectionsOppened():
    p1 = Popen(['pgrep', '-f', '/dev/ttyUSB'], stdout=PIPE)


    log_file_name = "Log_kill_pids"
```

```
    for pid in p1.stdout:
        try:
            os.kill(int(pid), signal.SIGTERM)

        except OSError as ex:
            print("Error while killing process by pid")
            # Save old logging file and create a new one. No se crea automaticamente!!!!!
            # os.system("cp {0} '{0}-dup-{1}'".format(log_file_name, dt.now()))


def instanceConnections(nSerials):
    for x in range(0, nSerials):
        print('/dev/ttyUSB' + str(x))
        p1 = Popen(["ttyd", "-p", "808" + str(x), "miniterm.py", '/dev/ttyUSB' + str(x)])
        output = p1.communicate()[0]

if __name__ == "__main__":
    os.environ['DJANGO_SETTINGS_MODULE'] = 'CiscoApiSerialV2.settings'
    sys.path.append('./CiscoApiSerialV2')
    wsgi_app = tornado.wsgi.WSGIContainer(django.core.handlers.wsgi.WSGIHandler())
    tornado.options.parse_command_line()
    app = tornado.web.Application(
        handlers=[
            # (r"/webSockets/static/(.*)", tornado.web.StaticFileHandler, {'path': './'}),
            # (r"/ws", WebSocketHandler),
            (r'%s(.*)' % STATIC_URL, tornado.web.StaticFileHandler, {'path': STATIC_ROOT}),
            tornado_websockets.django_app(),
            (r'/(.*)',                                                    tornado.web.FallbackHandler,
dict(fallback=tornado.wsgi.WSGIContainer(wsgi.application))),
        ], debug=True)
```

```
    httpServer = tornado.httpserver.HTTPServer(app)
    httpServer.listen(options.port)
    print("Listening on port:", options.port)

    mainLoop = tornado.ioloop.IOLoop.instance()

    ## adjust the scheduler_interval according to the frames sent by the serial port
    scheduler_interval = 100
    scheduler = tornado.ioloop.PeriodicCallback(checkQueue, scheduler_interval,
io_loop=mainLoop)
    scheduler.start()
    mainLoop.start()
```