UNIVERSIDAD DE EL SALVADOR

FACULTAD DE INGENIERÍA Y ARQUITECTURA

ESCUELA DE INGENIERÍA ELÉCTRICA

Interfaz inalámbrica para un lector de código de barra.

PRESENTADO POR:

CARLOS MAURICIO RODRÍGUEZ CRUZ

CESAR EMERSON SANTOS RIVAS

PARA OPTAR AL TÍTULO DE:

INGENIERO ELECTRICISTA

CIUDAD UNIVERSITARIA, AGOSTO 2013

UNIVERSIDAD DE EL SALVADOR

RECTOR :

ING. MARIO ROBERTO NIETO LOVO

SECRETARIA GENERAL :

DRA. ANA LETICIA ZAVALETA DE AMAYA

FACULTAD DE INGENIERÍA Y ARQUITECTURA

DECANO :

ING. FRANCISCO ANTONIO ALARCÓN SANDOVAL

SECRETARIO :

ING. JULIO ALBERTO PORTILLO

ESCUELA DE INGENIERÍA ELÉCTRICA

DIRECTOR :

ING. JOSÉ WILBER CALDERÓN URRUTIA

UNIVERSIDAD DE EL SALVADOR

FACULTAD DE INGENIERÍA Y ARQUITECTURA

ESCUELA DE INGENIERÍA ELÉCTRICA

Trabajo de Graduación previo a la opción al Grado de:

INGENIERO ELECTRICISTA

Título :

Interfaz inalámbrica para un lector de código de barra.

Presentado por :

CARLOS MAURICIO RODRÍGUEZ CRUZ

CESAR EMERSON SANTOS RIVAS

Trabajo de Graduación Aprobado por:

Docente Director :

Dr. CARLOS EUGENIO MARTÍNEZ CRUZ

San Salvador, Agosto 2013

Trabajo de Graduación Aprobado por:

Docente Director :

DR. CARLOS EUGENIO MARTÍNEZ CRUZ

ACTA DE CONSTANCIA DE NOTA Y DEFENSA FINAL

En esta fecha, 29 de Julio de 2013, en la Sala de Lectura de la Escuela de Ingeniería Eléctrica, a las 5:30 horas, en presencia de las siguientes autoridades de la Escuela de Ingeniería Eléctrica de la Universidad de El Salvador:

1. Ing. José Wilber Calderón Urrutia
   Director — Firma: Wilber Calderón

2. Ing. Salvador de Jesús Germán
   Secretario — Firma:

ESCUELA DE INGENIERÍA ELÉCTRICA
FACULTAD DE INGENIERÍA Y ARQUITECTURA
DIRECCIÓN

Y, con el Honorable Jurado de Evaluación integrado por las personas siguientes:

1- Ing. Darwin Vladimir Chavarría — Firma:

2- Ing. José Wilber Calderón Urrutia — Wilber Calderón

Se efectuó la defensa final reglamentaria del Trabajo de Graduación:

Interfaz inalámbrica para un lector de código de barra.

A cargo del Bachilleres:

- Carlos Mauricio Rodríguez Cruz

- Cesar Emerson Santos Rivas

Habiendo obtenido en el presente Trabajo una nota promedio de la defensa final, de: 8.75

(OCHO PUNTO SETENTA Y CINCO)

AGRADECIMIENTOS

Agradezco principalmente a Dios por haberme dado la fortaleza, fe y confianza para culminar la carrera de Ingeniería Eléctrica.

En segundo lugar a mi familia Mauricio Rodríguez, Polita Cruz, Gabriela Cruz y Karla Rodríguez. Que me han estado a mi lado en todos estos años y a los que les dedico este documento.

Agradezco grandemente al Dr. Carlos Martínez por ser la persona quien confió en nosotros desde Proyecto de Ingeniería, para poder lograr este Trabajo de Graduación, orientando, dando esa voz de confianza y apoyo en todas las etapas del proyecto.

A mi novia Keren Mendoza quien ha sido esa persona que siempre me ha apoyado en esta etapa de mi vida, siempre ha estado a mi lado, dando ese aliento cuando más lo necesitaba y dando el primer abrazo cuando las cosas salían bien.

A mis amigos que quiérase o no, es mi otra familia, Cesar Santos y su familia, Lester, Rubén, Luis, Chambita, Ricardo, Carlitos y a todos aquellos que han estado a mi lado en las buenas y en las malas.

A mis amigos del voleibol y a mi entrenador Erick Hernández, ya que gracias a ellos disfrute mucho y di todo por los colores de mi Universidad.

Gracias a todos.

Carlos Mauricio Rodríguez Cruz

AGRADECIMIENTOS

Me gustaría que estas líneas sirvieran para expresar mi más profundo y sincero agradecimiento a todas aquellas personas que con su ayuda han colaborado en la realización del presente trabajo, en especial al mis padres Julio Cesar Santos Gómez y Dora Alicia de Santos, por el apoyo prestado durante todos los años de mi carrera. Al Dr. Carlos Martínez director de esta investigación, por la orientación, el seguimiento y la supervisión continúa de la misma, pero sobre todo por la motivación y el apoyo recibido a lo largo de la realización del trabajo de graduación.

Quisiera hacer extensiva mi gratitud a mis compañeros de la Escuela de Ingeniería Electica y, especialmente a mi compañero de trabajo Carlos Mauricio Rodríguez por su amistad y colaboración.

Un agradecimiento muy especial merece la comprensión, paciencia y el ánimo recibidos de mis hermanas.

A todos ellos, muchas gracias.

Cesar Emerson Santos Rivas

ACRONIMOS

- **Baudio:** Es una unidad de medida, usada en telecomunicaciones, que representa el número de símbolos por segundo en un medio de transmisión analógico.

- **cURL:** Es una herramienta para usar en un intérprete de comandos para transferir archivos con sintaxis URL.

- **Dragino:** El Dragino es un aparato que se maneja mediante OpenWrt, este permite incorporar Linux. Es un costo, Linux motherboard hardware abierto bajo para micro controladores. Se ejecuta Linux usando OpenWRT, y tiene plena Ethernet y capacidades de WiFi 802.11b/g/n.

- **Google App Engine (GAE)**: Permite ejecutar tus aplicaciones web en la infraestructura de Google. Las aplicaciones App Engine son fáciles de crear, de mantener y de ampliar al ir aumentando el tráfico y las necesidades de almacenamiento de datos. Con App Engine no necesitarás utilizar ningún servidor: solo tendrás que subir tu aplicación para que los usuarios puedan empezar a utilizarla.

- **GQL:** Es un lenguaje parecido a SQL que se utiliza para recuperar entidades o claves del almacén de datos escalable de App.

- **HDMI**: (High-Definition Multimedia Interface ó interfaz multimedia de alta definición) Es una norma de audio y vídeo digital cifrado sin compresión apoyada por la industria para que sea el sustituto del euroconector.

- **HTML**: (HyperText Markup Language ó lenguaje de marcado hipertextual) Hace referencia al lenguaje de marcado predominante para la elaboración de páginas web que se utiliza para describir y traducir la estructura y la información en forma de texto, así como para complementar el texto con objetos tales como imágenes

- **Jinja2**: Es un motor de plantillas con todas las funciones de Python. Tiene soporte completo de Unicode, un entorno de ejecución de recinto de seguridad integrada opcional, ampliamente utilizado y licencia BSD.

- **Lector de barra:** Es aquel que por medio de un láser lee un código de barras y emite el número que muestra el código de barras, no la imagen.

- **LUA**: Es un lenguaje imperativo, estructurado y bastante ligero que fue diseñado como un lenguaje interpretado con una semántica extendible.

- **LUCI:** Es un conjunto de software libre de Lua para dispositivos embebidos. Luci cuenta con varias herramientas y bibliotecas útiles para los desarrolladores, así como MVC-Webframework y la interfaz de usuario de la web que forma parte de OpenWrt Kamikaze 8.09.

- **Nube**: La computación en nube es un sistema informático basado en Internet y centros de datos remotos para gestionar servicios de información y aplicaciones. La computación en nube permite que los consumidores y las empresas gestionen archivos y utilicen aplicaciones sin necesidad de instalarlas en cualquier computadora con acceso a Internet.

- **Python:** Es un lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis muy limpia y que favorezca un código legible.

- **RCA**: Es un conector frecuentemente llamado conector Cinch, es utilizado para transportar señales de video o de audio. El conector RCA se utiliza para enviar señales de video y audio (en mono o estéreo) a través de un cable de dos hilos, con un método de transmisión que puede ser tanto analógico como digital.

- **RS232:** (Recommended Standard 232, también conocido como Electronic Industries Alliance RS-232C). Es una interfaz que designa una norma para el intercambio de una serie de datos binarios.

- **Raspberry Pi:** Es un ordenador del tamaño de una tarjeta de crédito que se puede conectar a una TV o a un teclado. Es un PC en miniatura con procesador ARM que se puede utilizar para muchas de las cosas que un PC de escritorio puede hacer, como hojas de cálculo, procesador de Word y videojuegos. Además, puede reproducir vídeos en alta definición.

- **SAS 70**: Ofrece orientación a los servicios de auditores al evaluar el control interno de una organización de servicio y la emisión de un informe de un auditor de servicios. SAS 70 también proporciona orientación a los auditores de los estados financieros de una entidad que utiliza una o varias organizaciones de servicio.

- **SCB:** (Placa computadora u ordenador de placa reducida). Es una computadora completa en un sólo circuito. El diseño se centra en un solo microprocesador con la RAM, E/S y todas las demás características de un computador funcional en una sola tarjeta que suele ser de tamaño reducido, y que tiene todo lo que necesita en la placa base.

- **SDK**: (Siglas en inglés de Software Development Kit). Es generalmente un conjunto de herramientas de desarrollo de software que le permite al programador crear aplicaciones para un sistema concreto, por ejemplo ciertos paquetes de software, frameworks, plataformas de hardware, computadoras, videoconsolas, sistemas operativos, etc.

- **Servicio web:** Es cualquier sistema de software diseñado para soportar interacción máquina a máquina sobre una red. A la vez se define como un sistema de software diseñado para permitir interoperabilidad máquina a máquina en una red.

- **SQL**: (De sus siglas en inglés Structured Query Language o Lenguaje de Consulta Estructurado). Es un lenguaje declarativo de acceso a bases de datos relacionales que permite especificar diversos tipos de operaciones en ellas.

- **SSAE 16:** Es una ampliación de la norma actual de presentación de informes sobre los controles en una organización de servicios.

- **UART**: (Des sus siglas de Universal Asynchronous Receiver-Transmitter ó Transmisor-Receptor Asíncrono Universal). Éste controla los puertos y dispositivos serie.

- **URL:** (Del inglés Uniform Resource Locator ó Localizador de Recursos Uniforme). Es una secuencia de caracteres, de acuerdo a un formato modélico y estándar, que se usa para nombrar recursos en Internet para su localización o identificación, como por ejemplo documentos textuales, imágenes, vídeos, presentaciones digitales, etc.

- **USB**: Universal Serial Bus es un estándar industrial desarrollado mediados de los años 1990 que define los cables, conectores y protocolos usados en un bus para conectar, comunicar y proveer de alimentación eléctrica entre ordenadores y periféricos y dispositivos electrónicos.

- **Webapp:** Es un sencillo marco para aplicaciones web que se puede utilizar a fin de desarrollar aplicaciones web Python para Google App Engine.

- **Wi-Fi**: Es un mecanismo de conexión de dispositivos electrónicos de forma inalámbrica.

OBJETIVOS

OBJETIVO GENERAL:

- Dotar a un lector de código de barra de capacidad inalámbrica y almacenamiento de datos.

OBJETIVOS ESPECÍFICOS:

- Construir una interfaz que se adapte al router inalámbrico Dragino.
- Comunicar el router Dragino con un servidor web.
- Desarrollar el servicio web en los centros de datos de Google.
- Automatizar el inventario de la Escuela de Ingeniería Eléctrica.
- Automatizar el servicio de préstamo de bodega.

Contenido

Contenido Anexos

Lista de Figuras

Lista de Figuras Anexos

Lista de Tabla

1.0 INTRODUCCION

El trabajo de graduación desarrollado consiste en dotar a un lector de código de barras comercial de la capacidad de transmitir datos de manera inalámbrica hacia una aplicación web. El objetivo fue automatizar los préstamos de equipo realizados en la bodega de la Escuela de Ingeniería Eléctrica.

Para lograr esto se dividió el trabajo en 3 partes. La configuración adecuada del router Dragino, dispositivo de hardware libre, la creación de una interfaz de comunicación entre el lector de barras comercial y el router, y la programación de una aplicación web para manejar los datos obtenidos desde el lector.

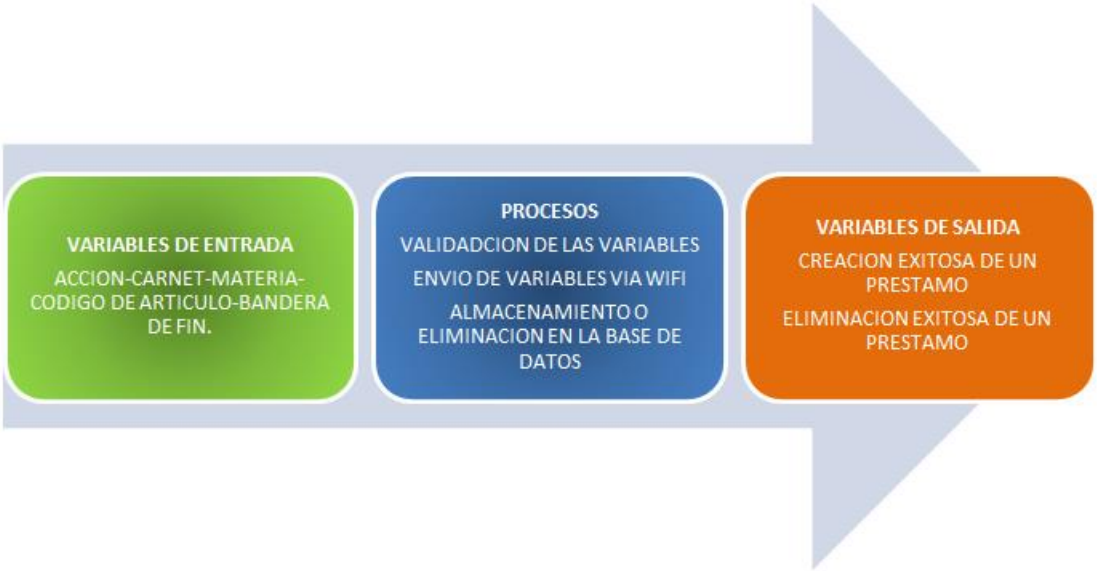


Figura 1.1 Diagrama de flujo del Proyecto.

1.1. DESCRIPCIÓN DEL PROYECTO

El proceso comienza, como se aprecia en la Figura 1.1 con la lectura de un grupo de códigos de barra generados para una de las dos operaciones básicas, creación de un préstamo o eliminación de un préstamo, el carnet de identificación de los estudiantes, y el código generado para cada uno de los artículos en la bodega de la Escuela de Ingeniería Eléctrica. Esta lectura se realiza por medio de un escáner de código de barras, específicamente un MS-9520 de la marca Metrologic, marca comercialmente conocida. Este dispositivo tiene una interfaz de comunicación USB se conecta a una Raspberry Pi modelo B, que sirve como interfaz entre el lector y el router Dragino. La Raspberry Pi, lee los datos desde su interfaz USB, y los envía al router por medio de su puerto serial. El router Dragino conectado directamente a la Raspberry Pi por medio de los su puerto serial recibe los datos y son enviados a la aplicación web a través de la red Wi-Fi de la Escuela de Ingeniería Eléctrica. La

aplicación web maneja las bases de datos de los estudiantes así como también de los artículos de la bodega, se realizan las comparaciones necesarias para validar la generación de los préstamos, también se verifica la eliminación de los préstamos. Los administradores del servicio web tienen acceso a cada una de las opciones de la aplicación web.

1.2. DISEÑO DE LA APLICACIÓN WEB

Para la generación de la aplicación web se eligió la herramienta Google Apps Engine. La filosofía para el diseño de la aplicación web del proyecto se basa en simplificar el uso de la aplicación para el usuario. Al mismo tiempo se facilita él envió de datos vía Wi-Fi por medio del router Dragino.

Todo el diseño se ha trabajado empleando aplicaciones en la nube de manera que no se necesite almacenamiento físico para los datos por parte del administrador. Para ello se utiliza las herramientas Google Drive para almacenar imágenes y archivos.

El servicio web consta de 11 plantillas HTML para presentar 11 interfaces gráficas para el usuario las cuales son:

1. Página principal: Es la página que se despliega primero al tener acceso mediante un navegador web esta solicitara al administrador del sistema su nombre de usuario y contraseña, contenidos en la base de datos de los usuarios validados, la cual contendrá un numero de administradores dados de alta por el administrador principal.

2. Menú Principal: Es el paso siguiente al iniciar sesión como administrador, éste selecciona una de las opciones mostradas en el menú. Estas opciones desplegaran una nueva página en el navegador con los campos y opciones necesarios para la actividad seleccionada.

3. Opción Ver Préstamo: Con el carnet del estudiante escaneado o digitado se da acceso a la base de datos de los préstamos pendientes de cada alumno, si se posee préstamos pendientes de entrega, muestra el artículo prestado, la fecha y hora en que se realizó el préstamo, para esto solo será necesario el carnet del estudiante. A esta opción tienen acceso los estudiantes y el administrador.

4. Agregar Préstamo: Se puede considerar una de las opciones principales de la aplicación web. A partir del carnet y el código del artículo extraídos de las bases de datos de los estudiantes y de la bodega crea una nueva entidad en la base de datos de los préstamos pendientes donde se agregan los datos del estudiante unidos con los datos del artículo que ha sido prestado. Por lo tanto los campos de entrada serán el carnet del estudiante así como el código del artículo, y una entrada adicional para tener registro de sobre la materia para la cual se ha necesitado el equipo.

5. Borrar Préstamo: Esta función complementa la anterior y borra el préstamo creado de la base de datos de préstamos pendientes, el préstamo asignado al carnet correspondiente. Para ello hace una consulta a la entidad basado en el carnet del estudiante y código del artículo, para borrar dicha entidad y liberar el préstamo del artículo. Por tanto los campos necesarios para realizar la operación son el carnet del estudiante y el código del artículo.

Las opciones 4 y 5 son las únicas accesibles por medio del router, que son las que el encargado de la bodega necesita para hacer los préstamos.

6. Nuevo Estudiante: Esta opción es de uso exclusivo del administrador y le permite agregar un nuevo estudiante a la base de datos de los estudiantes, para formar parte de la base de datos de la aplicación web, para ello se sigue el formato de base de datos proporcionado por la Escuela de Ingeniería Eléctrica y solo es necesario completar los campos carnet, nombre del estudiante.

7. Borrar Estudiante: Esta opción complementa la anterior y sirve para borrar una entidad de la base de datos de la aplicación web, basados únicamente en el carnet del estudiante. De la misma forma solo será accesible por el administrador del sistema.

8. Agregar Artículo: Con esta opción el administrador podrá agregar a la base de datos de la bodega, una nueva entidad, para estar disponible para realizar los préstamos los campos a completar están basados en el formato de inventario que maneja la universidad para llevar registro de los bienes estos son: número de inventario, descripción del bien, marca, modelo, serie.

9. Borrar Artículo: esta opción completa el manejo de la base de datos de la bodega, por parte del administrador, con ella se dará de baja a cualquier artículo que sea eliminado, y que ya no esté disponible para su préstamo.

10. Mostrar Estadísticas: basada en la información de la base de datos de estadísticas desplegara una imagen gráfica y dinámica de los prestamos mensuales, para esto se ha hace uso de la API de generación de graficas de Google Chart. De esta manera el administrador tendrá una forma de ver como se ha sido el nivel de préstamos en la bodega de forma mensual.

11. Acerca de los Autores: esta página muestra la información sobre los desarrolladores del proyecto además de un resumen del mismo.

1.3. LENGUAJES DE PROGRAMACIÓN UTILIZADOS EN EL TRABAJO DE GRADUACIÓN

Para programar el manejo de la aplicación web, y para programar la Raspberry Pi se utilizó Python. Este es un lenguaje de scripting multi-plataforma y orientado a objetos, Python está preparado para realizar cualquier tipo de programa, desde aplicaciones Windows a servidores de red o incluso, páginas web. Es un lenguaje interpretado, lo que significa que no se necesita compilar el código fuente para poder ejecutarlo, lo que ofrece ventajas como la velocidad de desarrollo.

La interfaz web se programó con HTML. Es el lenguaje con el que se definen las páginas web. Básicamente se trata de un conjunto de etiquetas que sirven para definir el texto y otros elementos que compondrán una página web.

Para programar el router Dragino se utilizó el lenguaje LUA. Es un lenguaje de extensión, suficientemente compacto para usarse en diferentes plataformas. Los programas en LUA no son interpretados directamente, sino compilados a código *bytecode*, que es ejecutado en la máquina virtual de LUA. El proceso de compilación es normalmente transparente al usuario y se realiza en tiempo de ejecución, pero puede hacerse con anticipación para aumentar el rendimiento y reducir el uso de la memoria al prescindir del compilador.

Para el manejo de la base de datos se usó el lenguaje GQL, es un lenguaje parecido a SQL que se utiliza para recuperar entidades o claves del almacén de datos escalable de GAE. Aunque las funciones de GQL son distintas de las de un lenguaje de consulta de una base de datos relacional tradicional, la sintaxis de GQL es parecida a la de SQL.

Para generar las gráficas se utiliza la herramienta *Interactive Charts*, de Google Charts Tools apropiada para la representación de datos en la cual está previsto algún tipo de interacción con el usuario. La herramienta de Google Charts Tools, se utiliza Javascrip, que es un lenguaje para el diseño de sitios web. No requiere de compilación ya que el lenguaje funciona del lado del cliente, los navegadores son los encargados de interpretar estos códigos. Javascript tiene la ventaja de ser incorporado en cualquier página web, puede ser ejecutado sin la necesidad de instalar otro programa para ser visualizado.

2.0 ETAPA DE HARDWARE

En este capítulo se presentan los resultados en el desarrollo del hardware. Se comienza especificando los lectores de barra con su parte de la electrónica. Luego el router Dragino. Por último el Raspberry Pi y su forma de comunicarse con el router Dragino.
Para tener una idea general de cómo será la conexión del Hardware se dará una breve explicación con la Figura 2.1 el cual muestra el diagrama de conexión del proyecto.

Figura 2.1 Muestra la forma de conexión del Hardware

La idea básica de cómo trabajaran todos los dispositivos y el proceso que lleva se describe a continuación: El escáner lee la correspondiente entrada, ver más acerca del escáner en el anexo B1 y B2. Los códigos de barra ya leídos, van directamente al Raspberry Pi, este tiene la función de recibir la señal por medio del puerto USB y transformarla en un sistema de comunicación serial UART, el cual es el sistema de comunicación entre Raspberry Pi y Dragino[1].
El router Dragino se encarga de mandar la información vía, Wi-Fi, hacia internet a los servidores de Google, teniéndolo como un servicio web ver más acerca del Dragino en el anexo B3.
Ya que este trabajo de graduación empezó hace un año como proyecto de ingeniería, se hicieron varias pruebas a lo largo de este tiempo a continuación de presentaran los resultados obtenidos y los problemas que surgieron, tomando en cuenta el lector de barras y el Dragino.

2.1 COMUNICACIÓN ENTRE ESCÁNER Y ROUTER DRAGINO

Desde un principio se manejó la idea de tener la conexión directa entre el escáner y el Dragino. Inicialmente se utilizó una computadora únicamente para alimentar encender el escáner y poder hacer las pruebas de esa manera.

La interconexión entre el escáner y el Dragino requirió de muchas pruebas. Se practicaron varias configuraciones al escáner y así como al Dragino. Ambos dispositivos tienen que utilizar el mismo protocolo de comunicación.

El router Dragino se debe comunicar con la misma velocidad que el escáner. Por defecto, el router Dragino trabaja a una velocidad de 9600 baudios. Por lo que se programa el escáner para que su velocidad de transmisión sea de 9600 baudios. En la Figura 2.2 se muestran los códigos de barra que permiten la configuración automática del escáner[2]. Leyendo el código de la Figura 2.2a se configura

el escáner en modo emulado RS232. Mediante el código de barra de la Figura 2.2b se ajusta la velocidad de transmisión de datos.

Figura 2.2 a) Programación del escáner al modo RS232 b) Cambio de velocidad de transmisión del escáner.

La señal del puerto serie del router Dragino es una señal UART con amplitudes de voltaje de 3.3V [3]. Los valores de voltaje de la señal RS232 del escáner con el código de la Figura 2.2 son de +/-12V. En la Figura 2.3 se muestra la amplitud de voltaje de la señal del "RS232 emulado" en verde se muestra la salida del escáner que es aproximadamente de 10.4 V por lo que es necesario bajar las tensiones a 3.3 V.

.

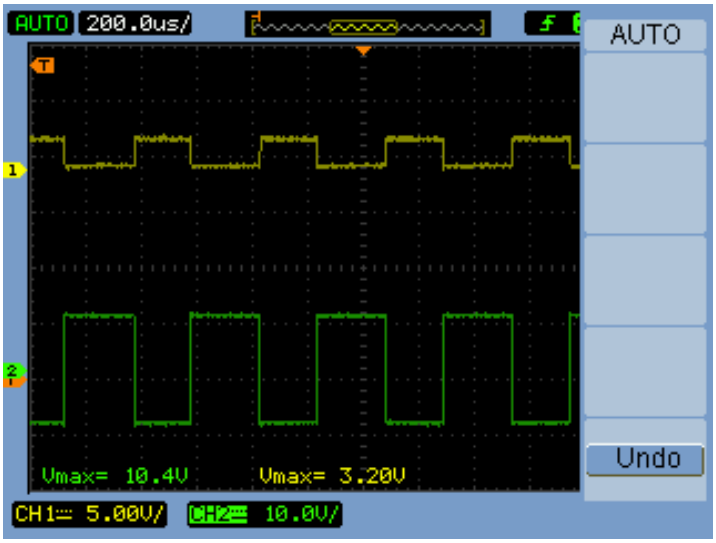


Figura 2.3. Muestra las amplitudes de los voltajes del escáner y después de la salida del circuito UART.

Para obtener estos valores de voltaje entre el escáner y el router Dragino se construyó un circuito para compatibilizar los voltajes entre los dos dispositivos. Este integrado convierte las señales de +/-12V a valores a 3.3V. El circuito utilizado se muestra en la Figura 2.4 y la construcción del circuito convertidor RS232- a 3.3V se muestra en la Figura 2.5.

Así mismo se muestra en la Figura 2.3 en color amarillo la respuesta del integrado MAX 232 al convertir la señal de 10.4V a 3.20V lo que se necesitaba para establecer una comunicación entre el router y el escáner. Ver anexo B4 para información del MAX 232.

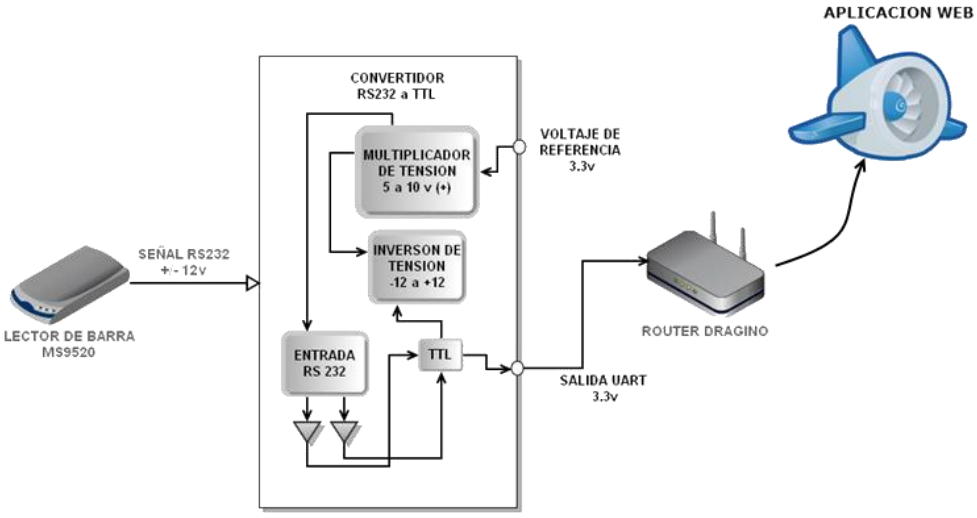


Figura 2.4 Muestra como es la conversión de RS232 a un voltaje de referencia que es 3.3v.

Figura 2.5Muestra el circuito y la implementación del mismo con el escáner.

Realizando todas las conexiones de este circuito se pudo leer los diferentes códigos de barra por medio del escáner. Los datos son enviados con éxito al puerto UART del router Dragino. Mediante la conexión de una PC al router el código se muestra a la computadora de manera casi instantánea [4].
Con la idea principal de hacer un solo circuito del escáner hacia el router. Se pasó al nuevo paso de eliminar la alimentación con la PC y poder alimentar al escáner independientemente. En el diseño original se consideró que el lector de barra se conectaría directamente al router Dragino. Para ello se tomaron en cuenta varias características del escáner que son de mucha ayuda para la conexión con el router Dragino como lo son:

La cantidad de baudios con las que por defecto trabaja el Dragino es de 9600 baudios. El lector de barras tiene la capacidad de cambiar su forma de transmitir sus datos que van desde 300 hasta 14500 baudios. El escáner tiene la capacidad de cambiar su configuración de transmisión a través de tres posibles vías: USB, RS232 y tipo keyboard. Esto es de gran ayuda ya que los circuitos de instrumentación de RS232 a UART que utiliza el Dragino y el Raspberry Pi necesitan de estos formatos para su comunicación para hacer las pruebas más información sobre el escáner ver en Anexo B.3.

EL ESCANER VOYAGER MS9520

Se tuvo algunos problemas con el uso del escáner MS9520. Para el escáner sus aplicaciones están hechas solo para aquellas que están dentro de aplicación por PC, por lo que una operación con un circuito externo carece de poca o ninguna información para su conexión o su uso. El escáner no puede encender con una fuente independiente. El escáner tiene su propio protocolo de comunicación con la PC para poder encender, el cual necesita del protocolo de comunicación USB2.0[5]. Se hizo varios circuitos independientes de la computadora por medio de los cuales se buscó separar al escáner.
FT232 es un dispositivo que convierte de USB a UART para una interfaz serial. Además tiene la capacidad de controlar los sistemas de comunicaciones asíncronos y síncronos (Puerto Serial) con modos combinación de bits de interfaz disponibles el chip Su diagrama de bloques y el circuito se muestran en la Figura 2.6 [6].

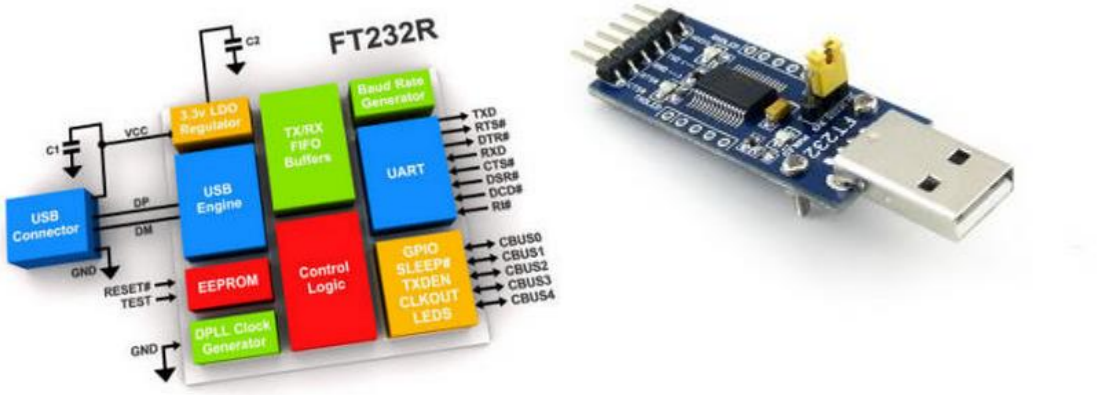


Figura 2.6 Muestra la forma interna del FT232 y su parte de aplicación física.

CIRCUITO 1: El circuito de la Figura 2.7 muestra la forma como se conectó el escáner con el circuito FT232 [7]. El circuito tratará de encender el escáner mediante el protocolo de comunicación USB 2.0 el cual ya viene incorporado en el circuito. La alimentación vendrá desde router Dragino con los pines de 5V y tierra, las cuales son indispensables para poder alimentar al escáner.

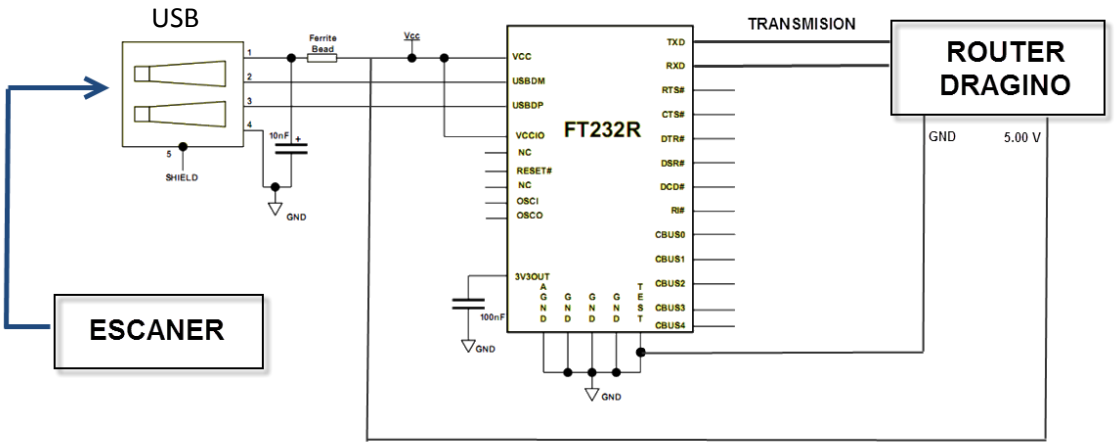
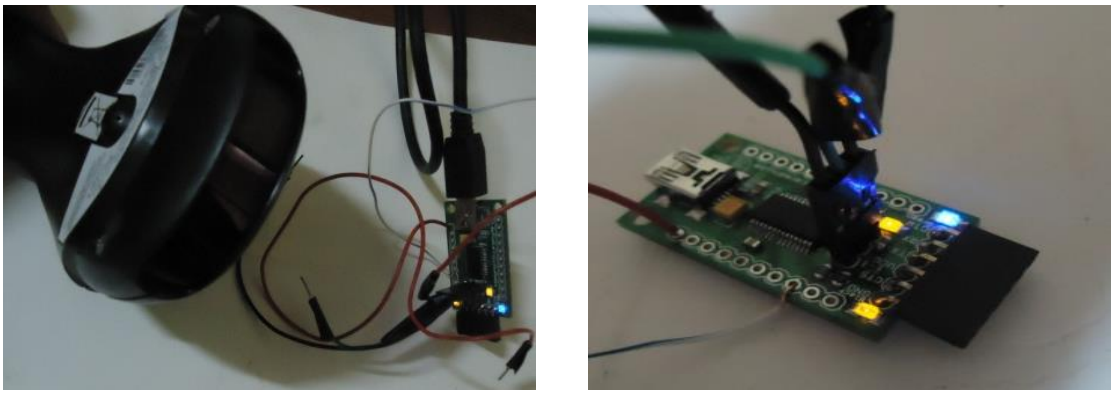


Figura 2.7 Circuito para alimentar USB.

La Figura 2.8 muestra el resultado del circuito las cuales generó las siguientes conclusiones. El escáner no pudo encender al realizar la prueba con el circuito, debido a los posibles problemas: El circuito con el FT232 genera una cantidad aproximada de corriente de 100mA establecido por la salida de 5V del router Dragino, esto debido a que las salidas del voltaje del router no son para alimentar a circuitos mayores a 100 mA. A diferencia el escáner requiere de 165mA para funcionar de manera óptima.
Así mismo el voltaje con el cual se manejó el router conectado a la carga del circuito generó 4.2 V, el escáner necesita de 4.75 V como mínimo para funcionar de manera óptima.



a) b)

Figura 2.8 a) Muestra el circuito unido con el escáner b) Circuito encendido y funcionando.

CIRCUITO 2: El circuito mostrado en la Figura 2.9 tiene una pequeña etapa de potencia. Esta etapa hace uso de una fuente de voltaje externa de 5V y 170mA. Este hace que fluya más corriente al escáner, el valor medido fue de 167mA a la entrada del escáner. El escáner su valor de operación es de 165mA aproximadamente con ello está dentro de los rango de corrientes aptos para el funcionamiento del escáner.

El valor de voltaje viene directo de la fuente independiente por lo que no hay un cambio significativo de voltaje de entrada hacia el escáner. El circuito externo consta de un transistor MOSFET, 2 capacitores de 0.1µF, y 2 resistencias de 1kΩ y 10kΩ

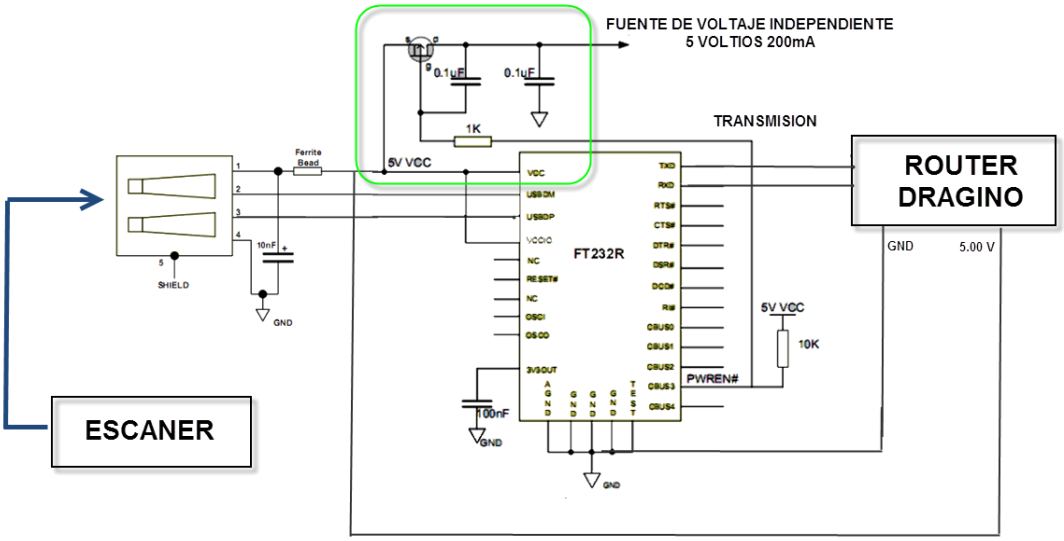


Figura 2.9 Muestra el circuito del escáner.

Como resultado de la conexión del circuito de la Figura 2.10 el escáner no logro encender con la etapa de potencia, esto debido a que el protocolo de comunicación de los dispositivos no concordó. Se trabajó en la programación del FT232 y no se tuvo ningún cambio, el problema siguió siendo el protocolo de comunicación USB 2.0 y la alimentación del escáner.

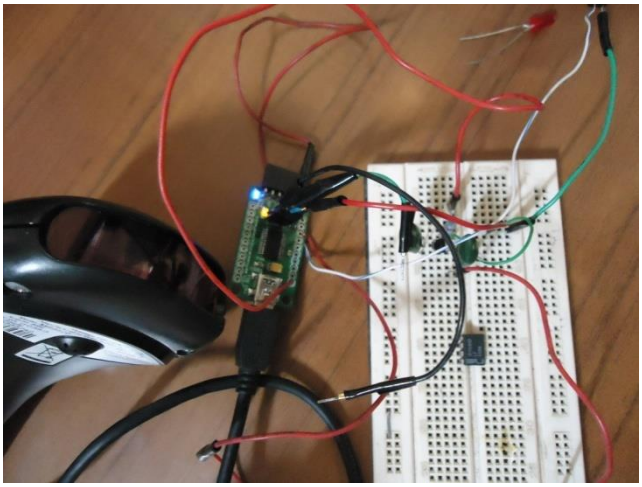
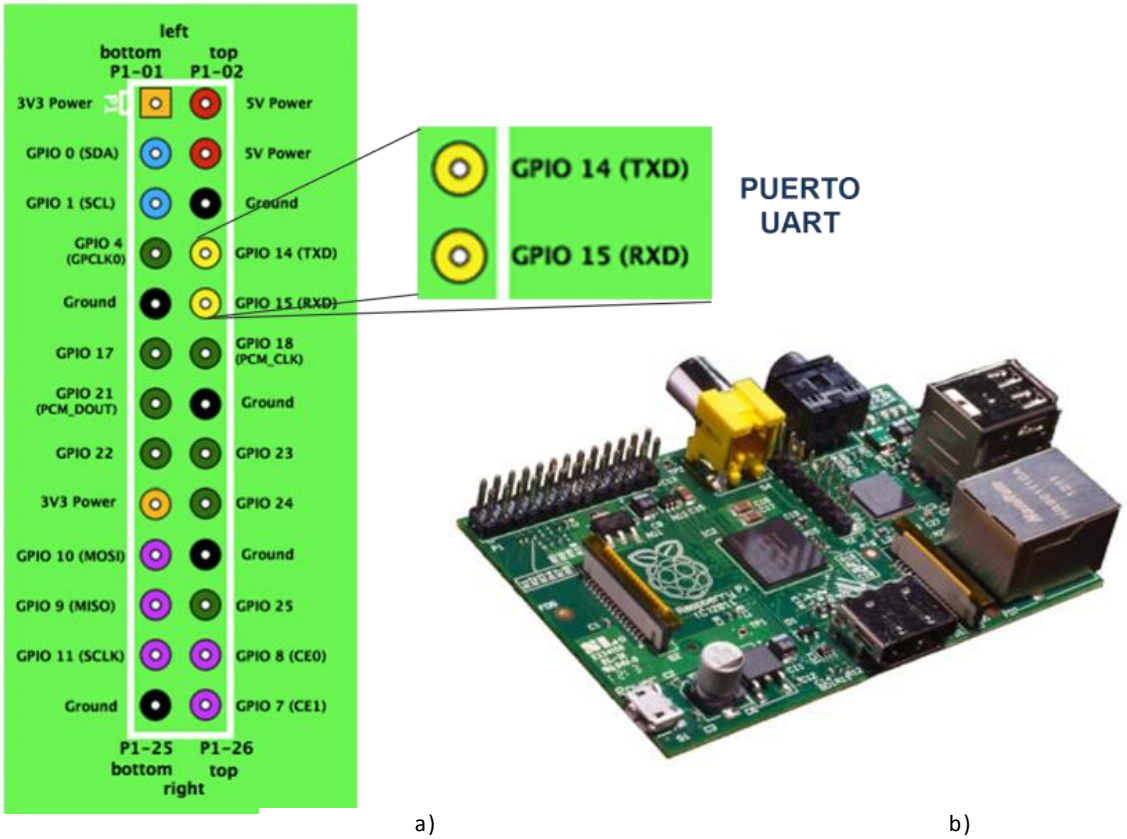


Figura 2.10 Muestra el circuito al cual se le da más corriente al circuito del escáner.

Este problema se solucionó incorporando el Raspberry Pi que se explicara de manera más detallada en la sección 2.2. Se resolvió el problema de la comunicación entre los dispositivos.

2.2 COMUNICACIÓN ENTRE RASPBERRY PI Y ROUTER DRAGINO

La nueva conexión entre el Dragino y el escáner se realizó por medio del Raspberry Pi mostrada en la Figura 2.11b, la cual es una computadora en un solo circuito, muy pequeño, de bajo costo y con una infinidad de aplicaciones [8]. A diferencia de la interconexión con el router Dragino, la conexión con el escáner se realiza por el puerto USB. Raspberry Pi cuenta con un sistema operativo llamado Raspian (Debian) Ver anexo B.5 para mayor información del Raspberry Pi y su sistema operativo.



a) b)

Figura 2.11a) Muestra las conexiones de los puertos GPIO del Raspberry Pi. b) Circuito Raspberry Pi.

La comunicación entre el Raspberry Pi y el Dragino se hizo siempre por medio de la comunicación serial UART, El Raspberry Pi cuenta con las salidas necesarias para este puerto de comunicación como lo muestra la Figura 2.11a que son los pines 14 y 15 de recepción (RxD) y transmisión (TxD) respectivamente [8].

La conexión del pin 14TxD del Raspberry Pi conectado con el RxD del Dragino hace que se puedan enviar los datos. Los pines de tierra de los dos dispositivos se tendrán que conectar para hacer una sola referencia y poder transmitir hacia Dragino. Para la utilización de estos puertos se hizo necesario cambiar algunos códigos del Raspberry Pi para que la señal serial funcione de forma eficiente y transmita valores que se desean.

CONFIGURACION PUERTO SERIAL

Luego de instalado todos los paquetes del Raspberry Pi como lo detalla el Anexo B.5.1. Debido a que de fábrica el puerto viene configurado como un puerto de consola para monitorear el funcionamiento del Raspberry Pi. Se utilizó las librerías de manejo de archivos de Linux para controlar el archivo asociado a este módulo. Este archivo es /dev/ttyAMA0, y funciona como una cola FIFO tanto en transmisión como en recepción.

Antes de poder utilizar este módulo, es necesario desbloquearlo, ya que la Raspberry Pi lo utiliza por defecto para mostrar mensajes de control y diagnóstico. Para esto, se modificó los siguientes dos archivos de configuración.[9]:

Como primer paso se abre el archivo del directorio `/boot/cmdline.txt` mostrado en la Figura 2.12, Es recomendable hacer un backup de este archivo por si la configuración falla:

```
$ sudo nano/boot/cmdline.txt
```

Figura 2.12 Muestra cómo acceder a la carpeta que contiene el archivo cmdline.txt.

El contenido de boot/cmdline.txt se muestra en la Figura 2.13:

```
dwc_otg.lpm_enable=0 console=ttyAMA0,115200 kgdboc=ttyAMA0,115200 console=tty1
root=/dev/mmcblk0p2 rootfstype=ext4 elevator=deadline rootwait
```

Figura 2.13 Muestra el contenido de cmdline.txt.

Del código de la Figura 2.13 se elimina la línea mostrada en la Figura 2.14:

```
console=ttyAMA0,115200 kgdboc=ttyAMA0,115200
```

Figura 2.14Muestra la línea a eliminar del archivo cmdline.txt.

Por lo que el archivo cmdline.txt quedará de la siguiente manera como en la Figura 2.15:

```
dwc_otg.lpm_enable=0 console=tty1 root=/dev/mmcblk0p2 rootfstype=ext4
elevator=deadline rootwait
```

Figura 2.15 Muestra cómo quedará el archivo final del cmdline.txt.

Luego de haber modificado el archivo cmdline.txt se abre el archivo del directorio `/etc/innitab` como lo muestra la Figura 2.16, siempre hacer el backup de este archivo.

```
$ sudo nano/etc/innittab
```

Figura 2.16 Muestra cómo acceder a la carpeta que contiene el archivo innittab.

Del código innittab se le agrega el carácter de comentario # a las líneas que se muestran en la Figura 2.17. Esto para dejarlo como comentario y que Python le de los parámetros como los baudios y el tiempo entre mensajes.

```
#Spawn a getty on Raspberry Pi serial line
#T0:23:respawn:/sbin/getty -L ttyAMA0 115200 vt100
```

Figura 2.17 Muestra las líneas a eliminar en el código innittab.

Una vez realizadas estas modificaciones se reinicia el sistema para que las configuraciones queden instaladas. Estas modificaciones evitarán que el Kernel envíe mensajes a la línea serial que nos pueden generar algún tipo de caracteres que no deseados.

INSTALACIÓN DE PYSERIAL

Ya configurado el puerto serial se necesita instalar la librería para poder tener la interacción con el puerto. Para ello se necesita de la librería PySerial, que es un módulo de Python para interactuar con la interfaz serie. Se puede descargar en https://pypi.python.org/pypi/pyserial. Para extraer el archivo e instalarlo se utilizan los siguientes comandos mostrados en la Figura 2.18:

```
1. mkdir pyserial-2.6
2. tar -zxvf pyserial-2.6.tar.gz pyserial-2.6
3. cd pyserial-2.6
4. python setup.py install
```

Figura 2.18 Muestra las líneas necesarias para instalar PYserial.

Tomando en cuenta Figura 2.20 se describe cada una de las acciones a realizar: Se tiene que generar un directorio llamado pyserial-2.6 para poder extraer los archivos que se descargaron.
Se extraen los documentos en dicha capeta. Se abre el folder pyserial-2.6, dentro de esa carpeta se instala con la función setup.py que es de Python.

CODIGO UTILIZADO PARA LA COMUNICACIÓN

El programa deberá leer el código de barras, luego, transformar ese código en lenguaje entendible de máquina, y al final mandar ese código ya traducido a la salida del puerto UART. Se utiliza el lenguaje de programación Python. Este es un lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis muy limpia y que favorezca un código legible [10].

El código se basa en el desarrollo de la dinámica en que se harán las peticiones del artículo, la primera petición será agregar ó borrar un artículo, como segundo paso, pedir el carnet. Tercero, la materia. Cuarto se hacen las peticiones de los artículos finalizando con un código de barra fin. Luego se hace una recursividad comenzando por agregar o borrar. El escáner lee el código, transforma lo que ha leído y lo manda al puerto serie hacia el Dragino como lo muestra el flujograma de la Figura 2.19:

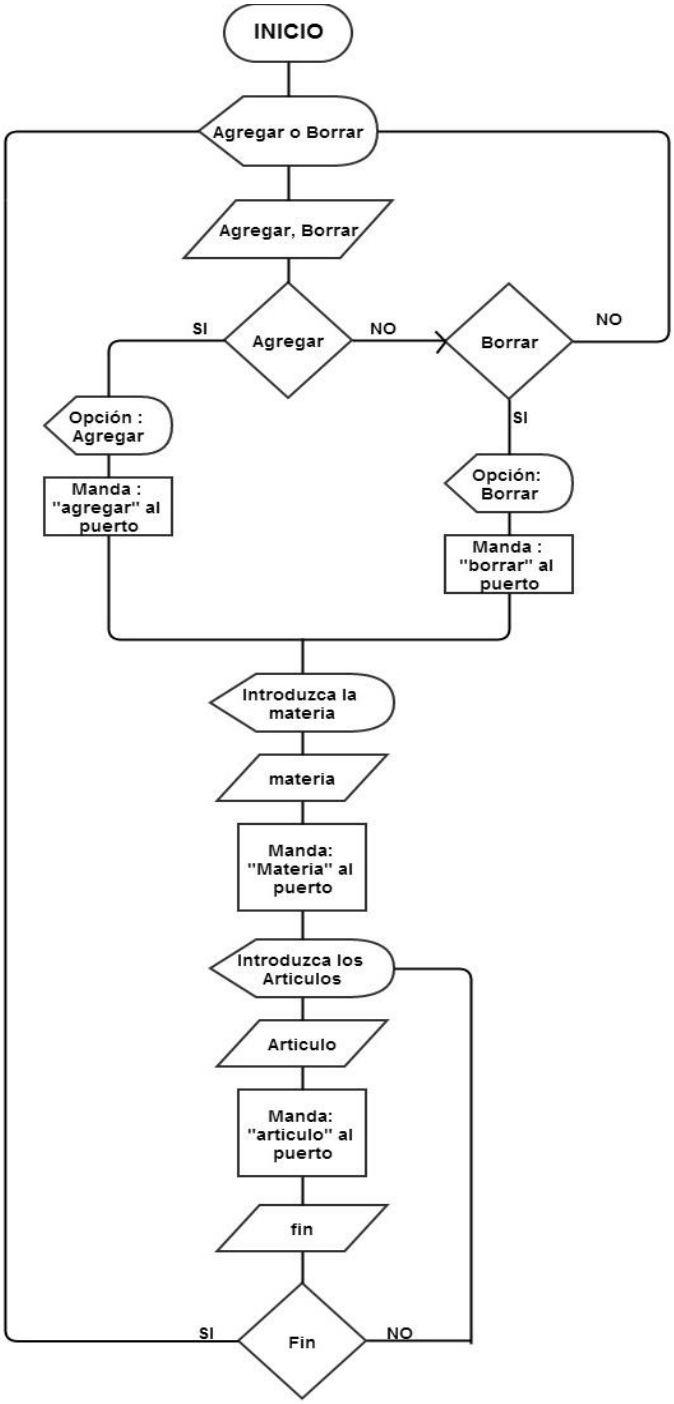


Figura 2.19 Muestra el flujograma para las acciones que hará el programa de Python.

PROGRAMA EN PYTHON

La aplicación de los flujogramas anteriores se ha transformado en la elaboración del código llamado progra.py mostrado en la Figura 2.20.

```
import sys
import time
import serial

ser = serial.Serial(port='/dev/ttyAMA0', baudrate=9600, timeout=1)
```

Figura 2.20 Muestra el inicio del código de Python.

En la Figura 2.20 se muestran los módulos de Python que se utilizaran en la creación del programa. Para el caso son: El módulo `sys` este trabaja con la entrada de datos como un archivo, analiza el contenido del texto para el caso, este módulo trabaja con varias formas como `stdin`, es la forma de entrada vista como archivo tomando en cuenta `readline`. El módulo `time` se encarga de darle tiempos de ejecución al programa, ya que si se hace todo al tiempo real se pueden introducir valores al Dragino que no son parte del código enviado. El módulo `serial` tiene el acceso al puerto serial y todas sus atribuciones, leer y escribir. La línea `ser` es la que le da los parámetros al módulo `serial` para establecer los siguientes parámetros:

`serial.Serial`: Este parámetro abre el puerto Serie del Raspberry Pi configurado anteriormente como lo muestra la Figura 2.17 donde se le quitan algunos parámetros del archivo cmdline.txt para poder dárselos a continuación.

`port='/dev/ttyAMA0'`: Indica donde tiene que ir a buscar la carpeta que maneja el puerto serial, ya que lo que este en esa carpeta se mandará hacia el tty del Dragino.

`baudrate= 9600`: Es la velocidad de transferencia a la cual se ha configurado los dos dispositivos. Si alguno tiene otro baudrate no se podrían comunicar entre sí o podrían mandar un tipo de datos no deseado.

`timeout=1`: Este manda datos cada 1 segundo hacia Dragino.

Todos estos parámetros sirven para que la comunicación entre los dos dispositivos síncrona y que no tengan problemas para la recepción de datos. Seguido de esta introducción, el puerto se da la lógica de programación al programa como tal con todas sus variables que se describirán a continuación.

```
1.while True:
2.        print "Introduzcamos agregar o borrar o salir"
3.        linea = sys.stdin.readline()
4.        if linea[:-1]=="agregar":
5.                mando(linea)
6.                print "escribir el carnet"
7.                car = sys.stdin.readline()
8.                mando(car)
9.                time.sleep(1)
10.               print "escribir la materia"
11.               mat = sys.stdin.readline()
12.               mando(mat)
13.               time.sleep(1)
14.               print "escribe los articulos "
15.               codigos = sys.stdin.readline()
16.               mando(codigos)
17.               art(codigos)
```

Figura 2.21 Muestra las peticiones iniciales del programa.

En la Figura 2.21 se muestra la lógica de programación desde el valor de entrada que se necesita, en la línea 3 el módulo sys llama y espera el valor que esta `stdin` que es la entrada del lector de barra del puerto USB y leerá la línea completa hasta EOF ó fin de archivo. Este devolverá un valor string para el caso que lo guardará en la variable llamada `linea`**,** con esto tendrá un control más preciso de la entrada del lector y el manejo de las acciones a realizar en el programa, agregar, borrar y salir.

Para la opción `agregar` mostrada en la línea 4 hace la comparación con lo que está en la variable `linea` si es igual a `agregar`. El apartado `==[:-1]` hace mención a la EOF mencionado anteriormente ya que en la línea 3, se pide la palabra completa incluyendo el ENTER. En la línea 5 se manda el valor hacia la función mando, y en la línea 9 se da una espera de un segundo en entre mandos hacia el Dragino esto con la intención que las peticiones se coordinen con el Dragino. A continuación se muestra la Figura 2.22.

```
if linea[:-1]=="borrar":
          mando(linea)
          print "escribir el carnet"
          car = sys.stdin.readline()
          mando(car)
          time.sleep(1)
          print "escribir la materia"
          mat = sys.stdin.readline()
          mando(mat)
          time.sleep(1)
          print "escribe los articulos "
          codigo = sys.stdin.readline()
          mando(codigo)
          art(codigo)
if linea[:-1]=="salir":
          break
else:
          print "sigue el programa"
ser.close()
```

Figura 2.22 Muestra el inicio del código de Python.

En la Figura 2.22 muestra la función borrar que es la misma que la llamada para agregar, solo se le agrega la función borrar al final la cual con break sale del programa o para el caso de Python.

```
1. def mando(param):
2. ser.write(param)
3. print "se le envio esto a Dragino>>>>>>>>>>>>>>>>>>>>>>>>>>>",param
4. time.sleep(1)
5. result = ser.read(ser.inWaiting())
6. print "esto se vera en el dragino>>>>>>>>>>>>>>>>>>>>>>>>>",result
```

Figura 2.23 Muestra la función de mando.

La función mando de la Figura 2.23 tiene por objetivo mandar los valores que da el escáner hacia Dragino, la línea 2 muestra la acción de transmitir hacia fuera del puerto serie del Raspberry Pi lo que está guardado en la variable `param,` la línea 5 se ocupó de prueba para saber si se podía recibir datos al puerto por medio del `ser.read` espera toda una línea con `in.Waiting.`

```
def art(param1):
time.sleep(1)
   print "se manda el articulo", param1
   codigos = sys.stdin.readline()
   if codigos[:-1] == "fin":
           mando(codigos)
           linea
   else:
           mando(codigos)
           art(codigos)
```

Figura 2.24 Muestra la función de art.

La palabra `fin` en la Figura 2.24 es la palabra clave para dar un paro a la cantidad de préstamos que se están haciendo, esto dentro de la función `art`.

2.3 CREACION DE DEMONIO PARA RASPBERRY PI

El programa que corre en el Raspberry Pi debe ser independiente de un teclado, un mouse y un monitor. Por ello se decidió darle al programa las características de un demonio.

Este tipo de programa se inicializa al encender el sistema, busca el código Python y lo ejecuta antes de entrar al sistema. Esto se incorporó en el Raspberry Pi, el código se ejecuta antes de la etapa en que pide usuario a la configuración del Linux, por lo que no hay necesidad de conectar un teclado para iniciar el Raspberry Pi.

PASO 1: Creación de archivo extensión SH.

Se crea un script del tipo `sh` llamado `progra.sh`, este código se muestra en la Figura 2.25, su función principal es crear un programa que ejecute una aplicación de Python al momento de encender el Raspberry Pi, sin la necesidad de ejecutarla por línea de comando:

```
#!/bin/sh

    case "$1" in
       start)
               python /usr/sbin/progra.py
               ;;
       stop)
               kill
               ;;
       *)
               echo "El uso es: $0 {start|stop}"
               exit 1
               ;;
    esac
    exit 0
```

Figura 2.25 Muestra el Código de SH para el programa en Python.

La Figura 2.25 detalla los siguientes pasos: Iniciando con `case "$1"`, sirve para dar parámetros de línea de comandos al programa, para el caso pediremos que Python como lenguaje de programación active por medio de `start` el programa `progra.py`, instantáneamente se corre el programa, luego de ello se desactiva el programa con la línea de comando `kill.`

PASO 2: Copiar archivo sh en `/etc/init.d/` y se cambian los permisos como se muestra en la Figura 2.26.

```
   cp progra.sh /etc/init.d
   /etc/init.d# chmod 755 /etc/init.d/progra.sh
   /etc/init.d# chown root:root /etc/init.d/progra.sh
```

Figura 2.26 Muestra los procesos para mover y otorgar permisos al archivo progra.sh.

La Figura 2.26 detalla los siguientes pasos: se guarda el comando `progra.sh` dentro del directorio `init.d`, dentro de este directorio se guardan todos los procesos que corren a la hora de arrancar la Raspberry Pi, luego se dan los permisos con `chmod 755` el cual da los permisos de lectura y ejecución para todos los usuarios menos el propietario, Y chown root:root sirve para cambiar el propietario del archivo y la información del grupo del usuario [12].

PASO 3: Se hace una prueba preliminar del programa mostrada en la Figura 2.27.

```
   sudo /etc/init.d/progra.sh start
   sudo /etc/init.d/ progra.sh stop
   sudo update-rc.d progra.sh defaults
```

Figura 2.27 Muestra el encendido y apagado automático del servicio así como también la forma de inicializarlo desde el modo de arranque.

Se inicia y se detiene el programa manualmente solo para un modo de prueba, el comando `update-rc.d progra.sh defaults` actualiza automáticamente los enlaces a los scripts de `init` tipo System. Estos son ejecutados por init cuando se cambia de nivel de ejecución y se usan generalmente para arrancar y parar servicios del sistema.

La opción `defaults` creará enlaces para arrancar los servicios en los niveles de ejecución 2345 y parar los servicios en los niveles de ejecución 016. Por omisión todos los enlaces tendrán el código de secuencia 20 [11].

PASO 4: Se mueve el archivo progra.py hacia `/usr/sbin` y se le dan los permisos como lo muestra la Figura 2.28

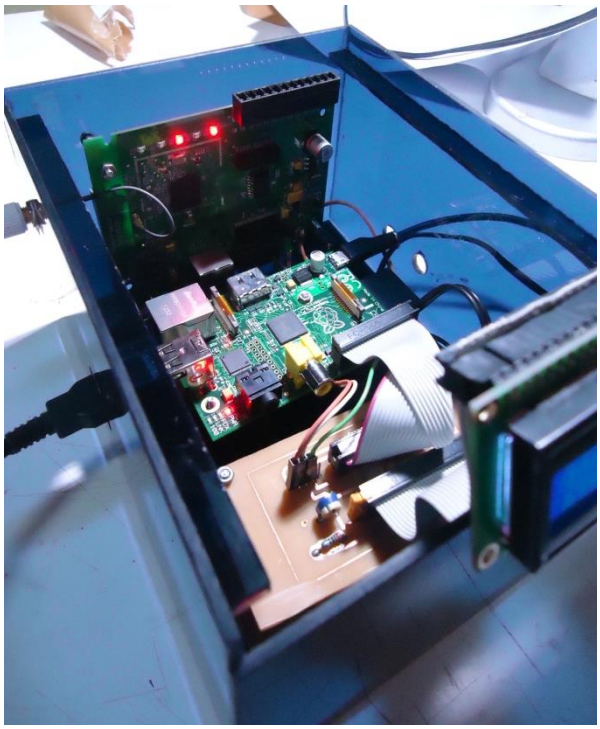
```
# cp progra.py /usr/sbin
/usr/sbin# chmod 755 progra.py
/usr/sbin# chown root: root progra.py
```

Figura 2.28 Muestra el movimiento de progra.py hacia sbin y a la vez se le otorgan los permisos.

Los comandos son igual que la Figura 2.27 Ahora el programa de Python está corriendo cada vez que se enciende el Raspberry Pi, automáticamente se enciende el programa y el escáner queda encendido esperando introducir una petición.

PRODUCTO FINAL

En la Figura 2.29 se muestra como se ha en ensamblado de los elementos del Hardware en uno solo. Para ello se ocupó material acrílico para el diseño de la caja, se unió el Raspberry Pi, el Dragino y un circuito para la pantalla LCD de 16x2 para mostrar las peticiones de router Dragino. Ver más en anexo B6.



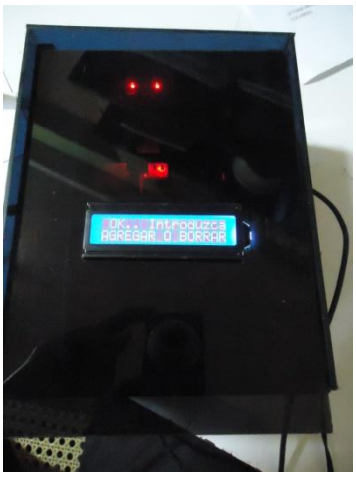


Figura 2.29 Muestra el trabajo final ensamblado en un solo circuito.

3.0 COMUNICACIÓN WI-FI

3.1. INSTALACIÓN DE FIRMWARE EN EL ROUTER DRAGINO

Antes de realizar la instalación del firmware al router se debe verificar si es compatible con el firmware seleccionado. Principalmente se debe verificar el espacio de memoria que ocupara el nuevo firmware. Para una mayor referencia al proceso de instalación ver el anexo B3.

Como primer paso configuramos IP a la tarjeta eth0 de la máquina de la forma que se muestra en la Figura 3.1. Esta no debe coincidir con la que se le da por defecto al router, que es 192.168.255.1.

```
ifconfig eth0 192.168.255.2 up
```

Figura 3.1. Línea para configuración de la IP de la máquina.

Luego se corre el comando mostrado en la Figura 3.2 con privilegios de super usuario. Para la instalación se utilizó la herramienta ap51-flash [13]. Los archivos de instalación del firmware deben de estar en la misma carpeta de la herramienta. Si se corre la línea de comandos fuera de este directorio se debe especificar la ruta completa de los archivos. Por lo que es preferible ubicarse en el directorio donde están los archivos del firmware y la herramienta para simplificar el proceso.

```
./ap51-flash eth0 openwrt-atheros-root.squashfs openwrt-atheros-vmlinux.lzma
```

Figura 3.2. Línea para correr la herramienta de instalación del firmware.

Linux buscará el Dragino en el Puerto Ethernet para hacer la instalación este debe estar conectado por medio del cable RJ45 pero sin la fuente de poder, al momento de que se muestre el mensaje de detección, se conecta el Dragino a su fuente de poder. AP51-flash detectara automáticamente el Dragino. Y comenzara a realizar la instalación. Al final del proceso nos aparecerá un mensaje para reiniciar el router y con esto se completa la instalación del firmware. La versión del firmware instalada en el router Dragino para el proyecto fue OpenWRT kAMIKAZE (8.09) como se puede apreciar en la Figura 3.3.

3.2. CREACIÓN DEL PROGRAMA PARA LECTURA DEL PUERTO SERIAL DEL ROUTER DRAGINO.

El código para la lectura y envió de datos a la aplicación web se escribió totalmente en LUA. El diagrama de flujo del programa se muestra en la Figura 3.4. Como se puede apreciar primero se abre el puerto serial, para hacer una lectura. Si esta lectura se mantiene, entra a un lazo `while` hasta que la lectura en el puerto sea diferente. Luego el valor leído se guarda en la variable `dato y acción.` Se realiza consecutivamente una segunda lectura del puerto. Se guardan esos valores en la variables `dato y carnet.` Se lee nuevamente el puerto para un nuevo valor esta vez asignado a las variables `dato y materia`. Luego entra a un lazo `while` de comprobación. De este lazo no saldrá hasta que lea el comando `fin.` De no leerlo, vera lo que está en el puerto y guardará ese valor en la variable

artículo. Procederá hacia una estructura `if` si la acción es `agregar` colocará en la página de préstamo nuevos de la aplicación web los datos de `carnet` y `artículo`, para crear el préstamo, este proceso se repetirá para varios artículos si es necesario, y terminara hasta que encuentre la cadena de caracteres `fin`. Al encontrar este artículo volverá a un lazo `while` infinito a pedir una nueva acción y se repetirá el proceso. Si la acción resultara que es `borrar` entonces colocara los datos en la página `borrar préstamo` de la aplicación web, realizando un procedimiento similar al de agregar préstamo.

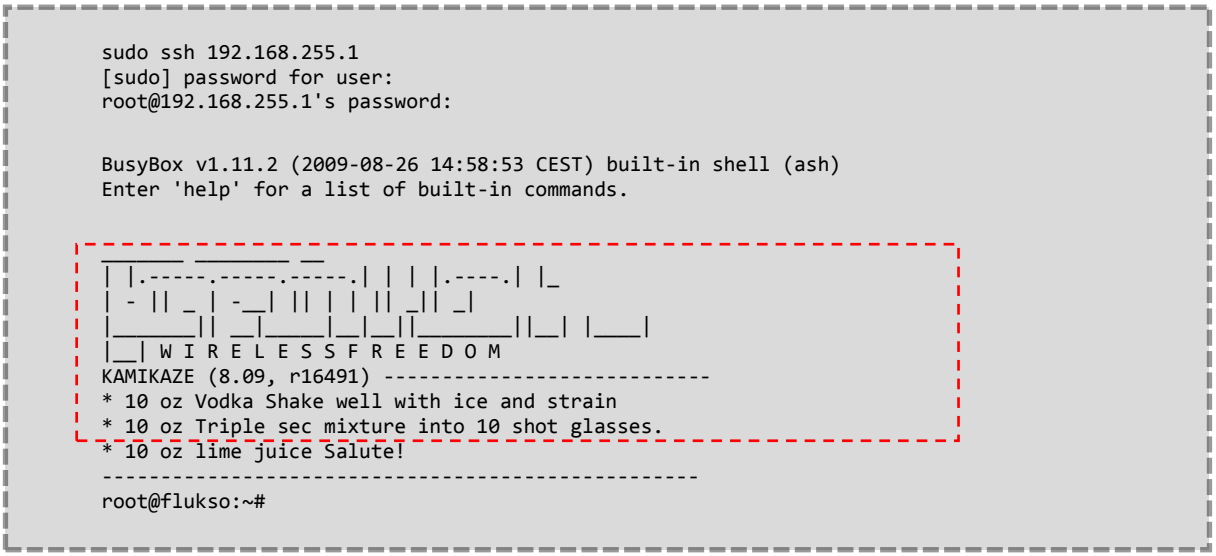


Figura 3.3.Acceso al router por medio de SSH y verificación del firmware instalado.

Los lazos `while` después de cada lectura sirven para verificar que el dato leído sea diferente. De no ser así, es decir si el usuario escanea el valor del carnet consecutivamente, el programa no podrá seguir al siguiente paso hasta que se introduzca el valor de una materia.

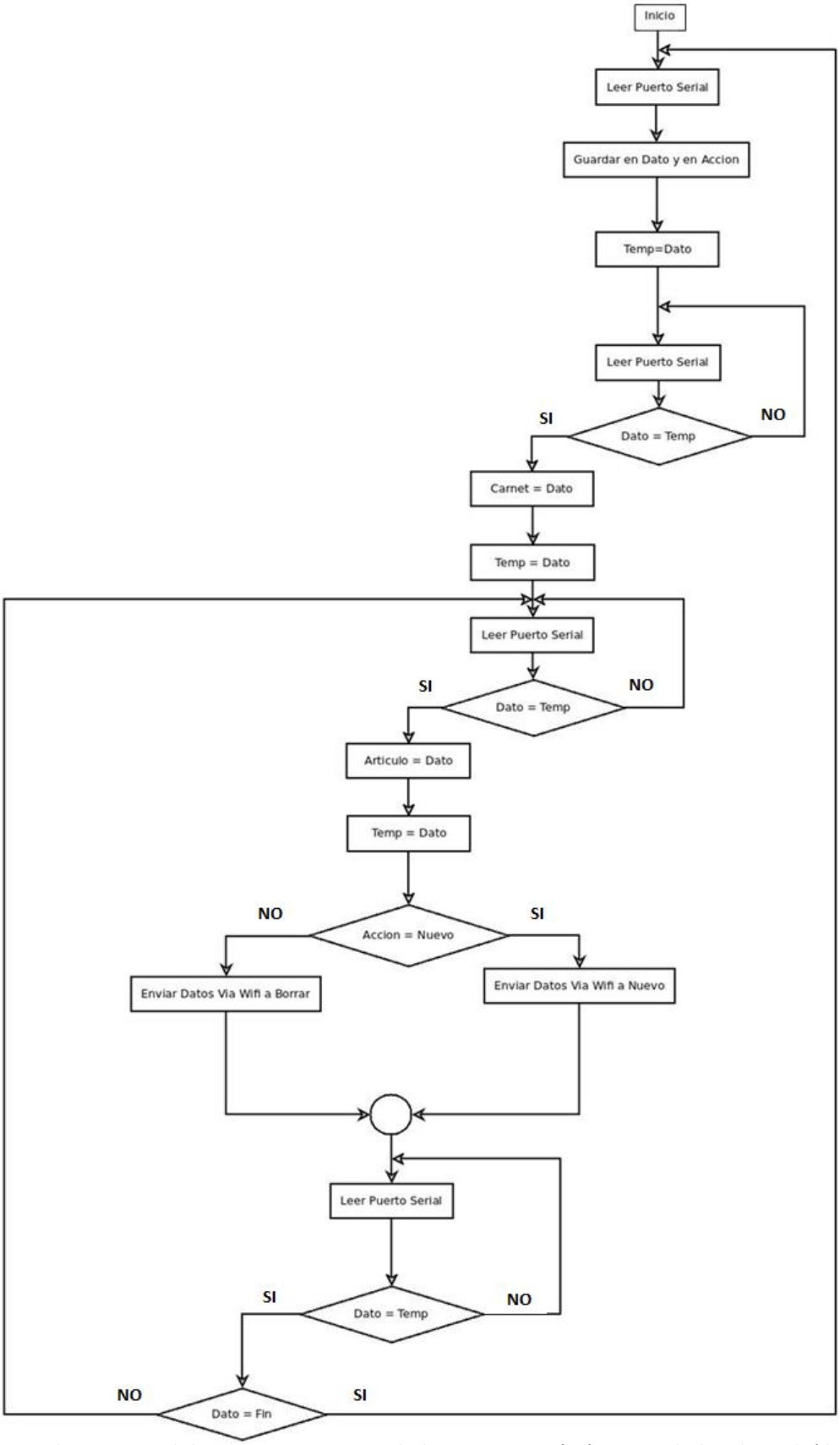


Figura 3.4. Flujo grama del Programa LUA para la lectura y envió vía WI-FI de los datos leídos desde el puerto serial del router Dragino.

El código LUA completo se muestra en la Figura 3.5, se aprecian claramente los lazos `while` y como se da el acceso al puerto como un archivo de lectura en la dirección `/dev/ttyS0`.

```
while true do

--leeserial=io.open("/dev/ttyS0","r")
while dato ~= "agregar" and dato~= "borrar" do
        print("Escriba una accion: ")
        dato = io.read()
        --dato=leeserial:read()
        --leeserial:flush()
        print("la accion es")
        print(dato)
end
accion=dato
temp=dato

while dato == temp do
        while  largo ~= 7 do
                print("Escriba un carnet")
                dato = io.read()
                --dato=leeserial:read()
                --leeserial:flush()
                print("el carnet es")
                print(dato)
                largo = #dato
        end
end

carnet = dato
temp = dato

while dato == temp do
        while largo ~= 6 do
                print("escriba una Materia")
                dato= io.read()
                --dato=leeserial:read()
                --leeserial:flush()
                print("la materia es")
                print(dato)
                largo=#dato
        end
end

materia = dato
temp = dato

while dato ~= "fin" do

        while dato == temp do

                print("escriba un Articulo")
                --dato=leeserial:read()
                --leeserial:flush()
                dato= io.read()
                print("el articulo es")
                print(dato)

        end
```

```
                articulo = dato
                        temp = dato
                if accion=="agregar" then
                        print("Pego en Nuevo Prestamo")
                        POSTEA_EN_APP=  'curl --data
"Carnet_in='..carnet..'&Materia='..materia..'&Codigo_in='..articulo..'&=%20Agregar%20"
http://lectorbarrasdragino.appspot.com/Nuevo_Prestamo'
                        os.execute(POSTEA_EN_APP)
                elseif accion=="borrar" then
                        print("Pego en Borrar Prestamo")
                        POSTEA_EN_APP=  'curl --data
"Carnet_in='..carnet..'&Codigo_in='..articulo..'&=%20Borra%20"
http://lectorbarrasdragino.appspot.com/Borrar_Prestamo'
                        os.execute(POSTEA_EN_APP)
                else
                        print("Accion Invalida")
                        break
                        end
                end

        end
```

Figura 3.5. Programa LUA para la lectura y envió vía WI-FI de los datos leídos desde el puerto serial del router Dragino.

Como se aprecia en la Figura 3.5 el programa comienza definiendo un lazo `while` infinito. Se escribió de esa manera para que el programa al iniciar no se detenga, y siga esperando órdenes cada vez que se termine un proceso de creación o eliminación de préstamos. Luego hace uso de la función `io.open` de LUA para tener acceso al puerto y luego se lee el dato en el puerto con la instrucción `leeserial:read()`. Esto se hace de esta manera ya que en un sistema Linux el puerto serial es manejado como un archivo de texto.

La variable `POSTEA_EN_APP` que se aprecia en la estructura `if` de la Figura 3.5 es una cadena de caracteres. Ésta concatena todos los valores desde el puerto para ser pasados subsecuentemente a la función `os.execute` de LUA, esta función ejecuta el argumento en la línea de comandos.

3.3. cURL PARA COLOCAR DATOS EN LA APLICACIÓN WEB.

Para colocar los datos en la aplicación web se utiliza la herramienta cURL. Como se aprecia en la Figura 3.5 la variable POSTEA_EN_APP comienza con cURL que es el comando que implementara os.execute en la línea de comandos.

cURL es una herramienta que transfiere archivos con sintaxis URL. Soporta FTP, FTPS, HTTP, HTTPS, TFTP, SCP, SFTP, Telnet, DICT, FILE y LDAP. cURL soporta certificados HTTPS, HTTP POST, HTTP PUT, subidas FTP, Kerberos, subidas mediante formulario HTTP, proxies, cookies, autenticación mediante usuario. El principal propósito y uso para cURL es automatizar transferencias de archivos o secuencias de operaciones no supervisadas.

cURL no viene incluido en la versión de firmware que se instaló en el router Dragino. Se requiere la instalación de la misma antes de ser usada. El procedimiento para instalar cURL se muestra en la Figura 3.6 y se instala como un paquete normal en cualquier distribución de Linux. Antes de realizar la instalación se debe configurar la conexión Wi-Fi del router Dragino, ver el apartado 3.4.

```
opkg install curl
```

Figura 3.6. Instalación de la Herramienta cURL en el OpenWRT.

La estructura básica para el uso de cURL es como la de la variable `POSTEA_EN_APP` que se ve en la Figura 3.5. Comienza con el nombre de la herramienta. Luego la opción `–data` seguida de un string que contiene la información a colocar y los nombres de los campos donde se colocaran en el formulario HTML al que se hace referencia. La opción `&=%20Borra%20` hace referencia al método que se llamara al colocar los datos en este caso en el formularios generados es Borrar o Agregar. La dirección URL del formulario HTML se coloca al final de la instrucción. Hay que estar seguros que los datos se amarran a los campos correctos si no la aplicación web devolverá mensajes de error.

3.4. CONFIGURACIÓN DE LA CONEXIÓN WI-FI DEL ROUTER DRAGINO.

Para la configuración de la conexión Wi-Fi del router Dragino se utilizara la interfaz LUCI. Para ello se abre un navegador y se coloca la dirección IP del router. La herramienta ap51-flash asigna por defecto, 192.168.255.1. Esta dirección se coloca en la barra de direcciones del navegador y desplegara una interfaz como la mostrada en la Figura 3.7.

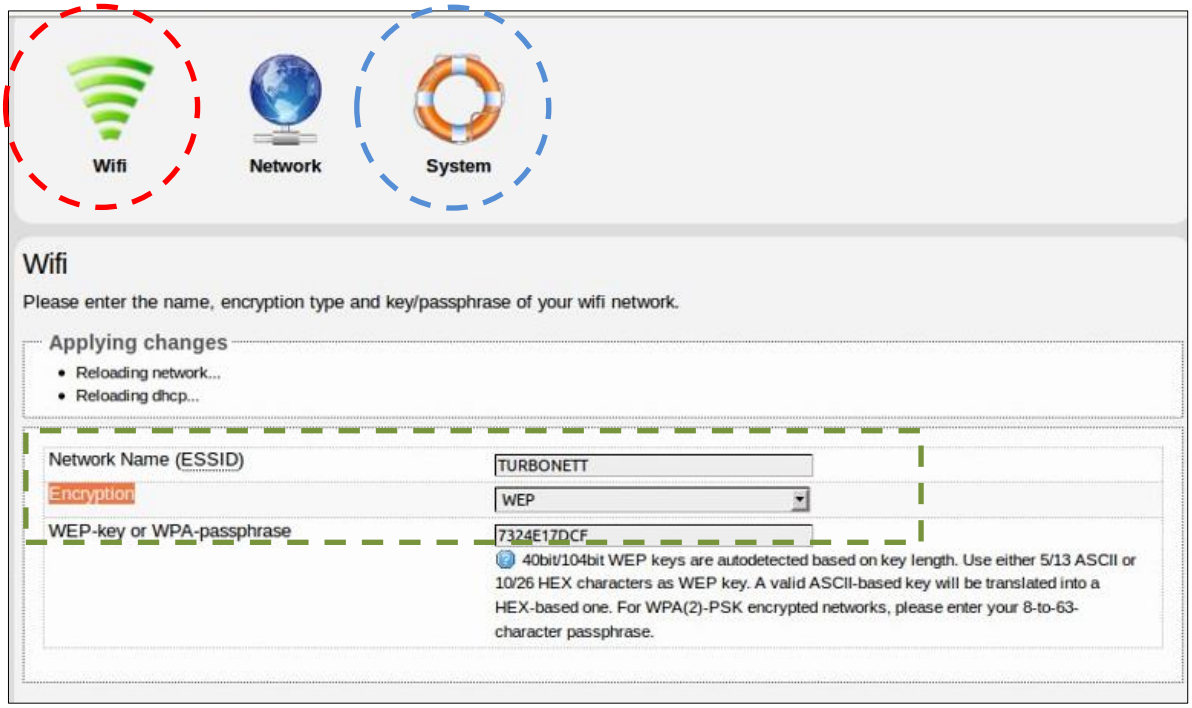


Figura 3.7. Interfaz LuCI de Dragino.

Al estar en esta interfaz se hace clic sobre el icono de sistema (System) marcado con azul en la Figura 3.7. Direccionará a la interfaz mostrada en la Figura 3.8. En ella se busca el botón *scan*, marcado con rojo en la Figura 3.8 para iniciar el proceso de búsqueda de las redes Wi-Fi disponibles. Al terminar el proceso de búsqueda se selecciona el nombre de la red que deseamos conectar y se hace clic en el

icono Wi-Fi, marcado en rojo en la Figura 3.7. En el campo *Network Name* colocamos el nombre de la red Wi-Fi seleccionada, en el campo *Encryption* seleccionamos el tipo de encriptación de la red seleccionada el más común es WEP. Si no posee encriptación, como es el caso de la red de la bodega de eléctrica se selecciona la opción *No Encryption* y se da clic al botón *save* para guardar la configuración El router Dragino recordará esta configuración siempre hasta que sea cambiada de ser requerido. Estos campos mencionados están marcados en verde en la Figura 3.7.

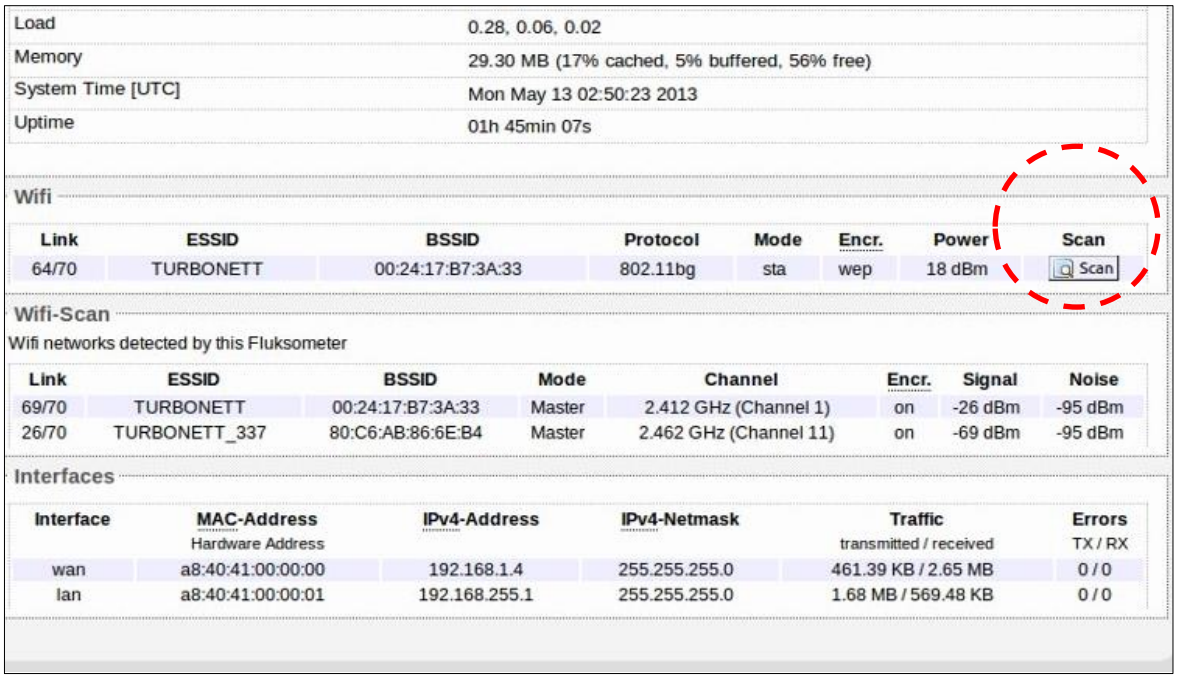


Figura 3.8. Interfaz LuCI de System para el Router Dragino.

3.5. EL *SCRIPT* LUA COMO DEMONIO.

Un demonio o *daemon* (*Disk And Execution Monitor*) en LINUX, es lo mismo que un servicio en Windows. Es un tipo especial de proceso no interactivo, es decir, que se ejecuta en segundo plano en vez de ser controlado directamente por el usuario. Este tipo de programas continua en el sistema, es decir, que puede ser ejecutado en forma persistente o reiniciado si se intenta matar el proceso dependiendo de configuración del demonio y políticas del sistema. Es necesario en el caso de la aplicación ya que no se tendrá una conexión directa del router a una computadora, por lo que no será necesario que el *script* LUA se ejecute de manera automática cada vez que el router se reinicie.

```
sasukeuchiha@ubuntu:~$ scp clector.sh root@192.168.255.1:/root/clector.sh

root@192.168.255.1's password:
clector.sh                                    100%  271     0.3KB/s   00:00
sasukeuchiha@ubuntu:~$ scp clector.lua root@192.168.255.1:/root/clector.lua
root@192.168.255.1's password:
clector.lua                                   100% 1422     1.4KB/s   00:00
```

Figura 3.9.Proceso de envió de los códigos generados al router Dragino.

Para crear el demonio en el router Dragino se siguieron los siguientes pasos:

PASO1: Se crearon en la computadora los archivos clectorsh presentado en la Figura 3.11 y clector.lua presentado en la Figura 3.5.

PASO2: Se enviaron los archivos al root del router como se muestran en la Figura 3.9.

PASO3: Se accedió al router con ssh y se movió el archivo `clectorsh` al directorio `/etc/init.d/` con el comando `cp`.

PASO4: Se cambiaron los permisos de ejecución del archivo clectorsh y también el propietario al root. Con los comandos chmod y chown.

PASO5: Se crearon los enlaces simbólicos al archivo clectorsh en el directorio correspondiente para que `init` sepa que tiene que arrancarlo (niveles 2, 3, 4 y 5) y pararlo (niveles 0, 1 y 6).

PASO6: El archivo clector.lua se movió al directorio `/usr/sbin` con el comando cp.

PASO7: Se le dieron los permisos necesarios al archivo `clector.lua` y se le cambio el propietario a `root` con los comandos `chmod y chown.`

PASO8: Se reinició el router para comprobar que el script lua iniciaba automáticamente al reinicio, esto se realizó con el comando `ps.` La Figura 3.10 muestra las líneas para realizar todos estos pasos.

```
root@dragino-562fff:~# cp clector.sh /etc/init.d
root@dragino-562fff:/etc/init.d# chmod 755 /etc/init.d/clector.sh
root@dragino-562fff:/etc/init.d# chown root:root /etc/init.d/clector.sh
root@dragino-562fff:/etc/rc.d# ln -s /etc/init.d/clector.sh /etc/rc.d/S76clector.sh
root@dragino-562fff:/etc/rc.d# ln -s /etc/init.d/clector.sh /etc/rc.d/K76clector.sh
root@dragino-562fff:~# cp clector.lua /usr/sbin
root@dragino-562fff:/usr/sbin# chmod 755 clector.lua
root@dragino-562fff:/usr/sbin# chown root:root clector.lua
```

Figura 3.10. Proceso de creación de un demonio en OpenWRT.

Si el sistema operativo lo permite se pueden cambiar las líneas `ln` por una sola instrucción `update-rc.d archivosh defaults.` En el caso del sistema utilizado en la aplicación esto no se aplicó.

3.6. EL ARCHIVO SH PARA EL DEMONIO.

La Figura 3.11 muestra el archivo clector.sh generado para la creación del demonio. Este es un script simple, su función principal es crear un programa que ejecute una aplicación de LUA al momento de encender el Dragino, sin la necesidad de ejecutarla por línea de comando. Se escribe la línea de comando `start()` del script, este dice al root que va a iniciar un programa**,** la siguiente línea pide cual es el programa que se necesita abrir en LUA**,** seguido de ello la dirección donde está alojado el archivo que se va a ejecutar y su respectivo lenguaje de programación LUA `/sur/sbin/clector.lua`**,** con el carácter `kill sirve` para detener el programa si la opción es stop.

```
#!/bin/bash

#
#!/bin/sh /etc/rc.common
#
START=98
start()
{
        lua /usr/sbin/clector.lua
}
stop()
{
        /etc/init.d/clectotsh kill
}
```

Figura 3.11. Archivo sh para la creación del dominio en el router Dragino.

4.0 DESARROLLO DE APLICACIÓN WEB GOOGLE APP ENGINE

4.1. CREACIÓN DE CUENTA EN GOOGLE APP ENGINE.

El primer paso para la creación de la aplicación fue crear una cuenta en Google [14]. La cuenta se puede crear con cualquier correo electrónico válido. Al iniciar la sesión se tiene acceso a la consola de administración de las aplicaciones mostrada https://cloud.google.com/console#/c=l. Esta consola se muestra en la Figura 4.1. Como se observa ésta muestra las diferentes aplicaciones que están relacionadas a la cuenta. Se pueden relacionar 10 aplicaciones web gratuitas por cuenta. La cuenta creada para la aplicación se llama *lectorbarrasdragino* y como se aprecia en la Figura 4.1 se ve que el estado de la aplicación es *Running*, que implica que está corriendo al momento. Para crear una aplicación se presiona el botón *Create Application* al final de la consola

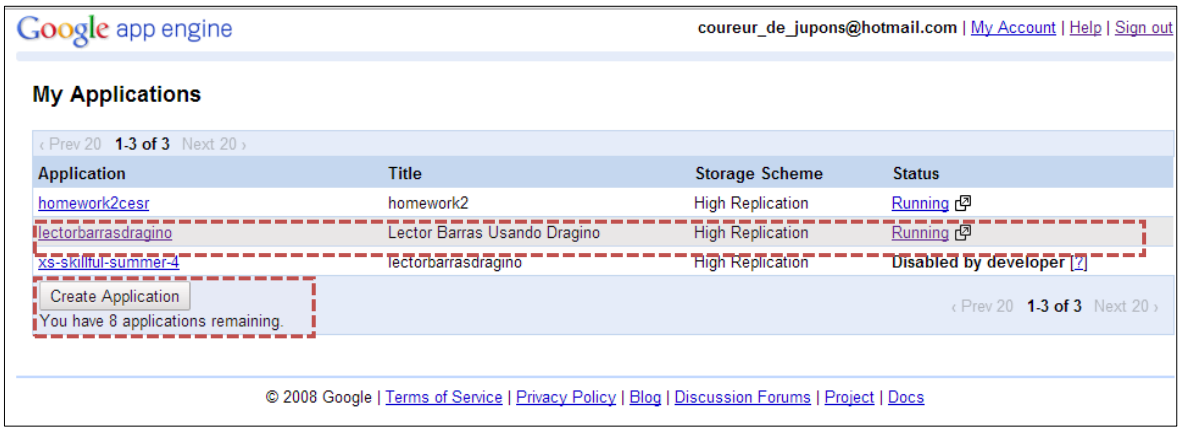


Figura 4.1. Consola de Administración de Aplicaciones de Google App Engine.

La ventana de creación se muestra en la Figura 4.2. En el primer campo se coloca el nombre de la aplicación. Este tiene la siguiente estructura del tipo: *miaplicacion.appspot.com.* Como se puede observar define el dominio de la aplicación. Al proyecto se le asignó el siguiente dominio: http://lectorbarrasdragino.appspot.com/

En el siguiente campo se coloca una descripción de la aplicación. Luego se observa otro campo con la opción para la autentificación de la aplicación (se recomienda abierta para cualquier usuario de *Google)*. El botón *Create Application*, marcado con rojo en la Figura 4.2 cierra el proceso de creación de la cuenta. Los nombres asignados deben de ser congruentes en todo el proceso de programación de la aplicación web.

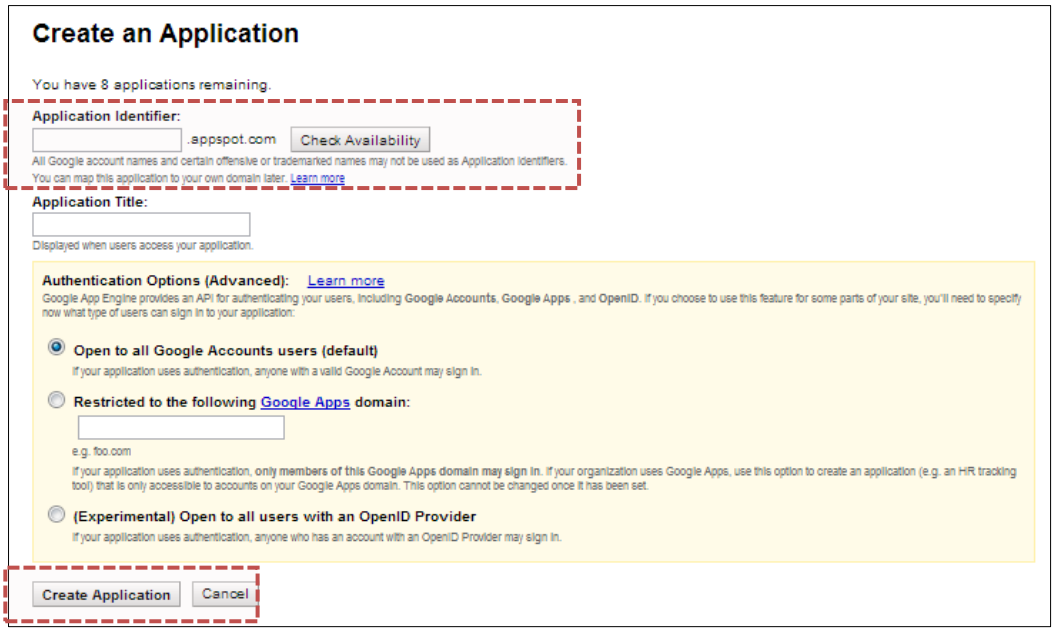


Figura 4.2. Página para la creación de una aplicación en Google App Engine.

4.2. CONFIGURACIÓN PREVIA.

Antes de comenzar a desarrollar código para la aplicación es necesario descargar el SDK o también llamado Kit de desarrollo de Sofware de Google App Engine [15]. Se seleccionó el SDK de Python ya que incluye herramientas para probar la aplicación, subir sus archivos de la aplicación, la gestión de los índices de almacén de datos y la descarga de los datos de registro. Al descargar el SDK se procede a crear una carpeta dentro del directorio googleapps, con el mismo nombre del indicador de la aplicación registrada en el sitio de *Google App Engine*. Dentro de este directorio se crean todos los códigos fuentes de la aplicación.

4.3. EL ARCHIVO APP.YAML.

Una aplicación creada con *Python App Engine* debe incluir un archivo de configuración denominado `app.yaml`. Este especifica la manera en la que las rutas de URL se corresponden con los controladores de solicitudes y con los archivos estáticos. También debe contener información sobre el código de la aplicación como, por ejemplo, el ID de la aplicación y el identificador de la última versión. Un archivo `app.yaml` debe incluir los siguientes elementos: Application, Version, Runtime, Handlers, api_version. Ver más en Anexo A.1.2.3.

El archivo app.yaml generado para la aplicación se muestra en la Figura 4.3.

```
application: lectorbarrasdragino

version: 1
runtime: python27
api_version: 1
threadsafe: true
libraries:
- name: jinja2
version: latest
handlers:
- url: /.*
script: lectorbarrasdragino.app
builtins:
- remote_api: on
```

Figura 4.3. Archivo .yaml de configuración para la aplicación generada.

En la Figura 4.3 se muestra la configuración de la aplicación del archivo .yaml, este incluyen dos opciones más –`remote_api` en estado on. Esto es necesario ya que se hizo uso del `bulkloader` que es una herramienta para la subida y descarga masiva de datos. Además, se incluye `libraries` donde se incluyó la librería `jinja2` para el manejo de las plantillas en la aplicación. El uso de estas dos herramientas se explicara más adelante.

4.4. EL ARCHIVO PRINCIPAL.

El archivo principal está escrito en Python. Este define todas las funciones necesarias para que la aplicación web maneje de manera adecuada todos los datos del Data Store, así como el intercambio de datos con el código HTML generado en cada una de las plantillas. La Figura 4.4 muestra las librerías incorporadas en el archivo principal. El nombre generado de la aplicación es lectorbarrasdragino.py, tal como se definió en la opción `application` del archivo de configuración `.yaml`. `lectorbarrasdragino` también coincide con el nombre dado en la consola de administración de las aplicaciones de la cuenta de *google app engine* , ver Figura 4.1

```
import os
import webapp2
import jinja2
import datetime
from datetime import date
import csv
from google.appengine.ext import db
template_dir =os.path.join(os.path.dirname(__file__),'plantillas')
jinja_env =jinja2.Environment(loader = jinja2.FileSystemLoader(template_dir), autoescape =True)
#clase contenedor
classHandler(webapp2.RequestHandler):
    defwrite(self,*a,**kw):
            self.response.out.write(*a,**kw)

    defrender_str(self, template,**params):
            t=jinja_env.get_template(template)
    return t.render(params)

    defrender(self, template,**kw):
            self.write(self.render_str(template,**kw))
```

Figura 4.4. Librerías importadas en Archivo Principal Python de la aplicación generada.

Como se muestra en la Figura 4.4, el archivo comienza haciendo las peticiones a las librerías que se utilizaron para el manejo de los datos de la aplicación web. La librería `webapp2`, es necesaria importarla ya que contiene todos los módulos de Google App Engine. Se importa también la librería `jinja2` para el manejo de las plantillas. Se importa la librería `db` de `google.appengine.ext` que contiene los módulos para la creación de las clases para la base de datos de la aplicación. Las librerías `os, datetime, csv,` y `date` son librerías para el manejo del sistema operativo, fechas y archivos csv de Python, respectivamente.

El código de la Figura 4.4 continúa definiendo la ruta al directorio que contiene los archivos HTML, de cada una de las páginas de la aplicación web. Luego se creó la clase Handler para manejar dichas plantillas.

4.5. DEFINICIÓN DE LAS ENTIDADES.

La Figura 4.5 muestra el código para definir las entidades creadas, estas entidades definen las bases de datos de la aplicación web. El esquema de la base de datos se muestra en la Figura 4.6 en la que se puede ver de forma resumida cuales fueron las entidades creadas y como se relacionan entre ellas.

```
#comienzan la definicion de las entidades
#IDENTIDAD PARA USUARIOS
classDatos_Usuarios(db.Model):
        usuario= db.StringProperty(required =True)
        contrasena= db.StringProperty(required =True)
        fecha= db.DateTimeProperty()
#IDENTIDAD PARA ESTUDIANTES
classDatos_Estudiantes(db.Model):
        Carnet=db.StringProperty()
        Estudiante =db.StringProperty()
        Direccion=db.StringProperty()
        Fecha_UPE=db.DateTimeProperty()
#IDENTIDAD PARA BODEGA
classDatos_Bodega(db.Model):
        Numero_Inventario=db.StringProperty()
        Descripcion=db.StringProperty()
        Marca=db.StringProperty()
        Modelo=db.StringProperty()
        Serie=db.StringProperty()
        Disponible=db.BooleanProperty()
        Fecha_UP=db.DateTimeProperty()
#IDENTIDAD DE PRESTAMOS PENDIENTES
classPrestamos_Actuales(db.Model):
        Carnet_P =db.StringProperty()
        Estudiante_P =db.StringProperty()
        Codigo_AP =db.StringProperty()
        Descripcion_AP=db.StringProperty()
        Materia =db.StringProperty()
        Fecha_P =db.DateTimeProperty()
#IDENTIDAD PARA REGISTRO DE ESTADISTICOS
classBase_Datos_Emes(db.Model):
        Meses_ES=db.StringProperty()
        Cantidad_Mes=db.FloatProperty()
```

Figura 4.5. Definición de las Entidades Generadas para la Aplicación.

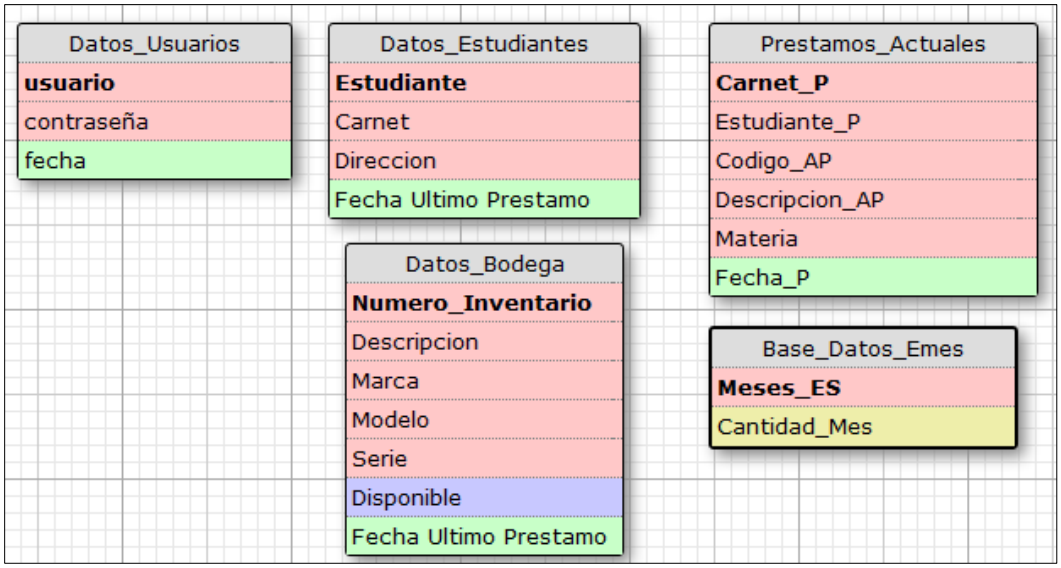


Figura 4.6. Bases de Datos generadas para la aplicación web.

Las entidades se definen en Google App Engine de la manera que se definen las clases en Python. La clase contiene cada una de las entidades necesarias, con un tipo definido para cada una de las bases de datos que se manejaron. La base de datos *Datos_Ususarios* contiene los usuarios dados de alta como administradores del sistema, con su nombre contraseña y fecha de último acceso. La Base de datos *Datos_Estudiantes* está conformada por la información de todos los estudiantes activos inscritos en la Escuela de Ingeniería Eléctrica, con su nombre, carnet, dirección y fecha del último préstamo realizado.

La base de batos *Datos_Bodega* contiene cada uno de las especificaciones de los artículos contenidos en el inventario de la bodega de la Escuela de Ingeniería Eléctrica. Además, posee dos campos para el control de los préstamos. El campo Disponible, que indica si el artículo no ha sido prestado. El campo *Fecha_UP* que muestra la última fecha en que fue prestado dicho artículo. La base de datos *Prestamos_Actuales* accede a la información de los estudiantes y de los datos de la bodega para crear una entidad de préstamo con el carnet, nombre del estudiante que realizó el préstamo, así como también el número de inventario de los artículos que se han prestado y de la descripción del mismo. Además añade la materia para la cual el estudiante prestó el artículo y la fecha en la que se realizó el préstamo.

Al devolver el artículo la entidad de ese préstamo es removida de la base de datos de *Prestamos_Actuales*. Por último, una base de datos extra se creó para llevar las estadísticas de préstamos de cada mes. Cada vez que se crea una entidad en la base de datos *Prestamos_Actuales*, se suma al mes un préstamo para luego mostrarlo en una gráfica.

4.6. LA CLASE MAINPAGE.

Esta clase maneja la entrada de los administradores de la aplicación. Posee una función post principal que lee los datos introducidos en el formulario HTML, para entrar un lazo *if* que compara si los datos introducidos por el usuario son primero están completos. Segundo, si están incluidos en la base de datos *Datos_Usuarios*. Si lo están redirige a la página del menú principal. Si los datos están incompletos muestra un mensaje de error y redirige nuevamente al formulario de entrada. La definición de esta clase se muestra en la Figura 4.7. Antes de hacer una verificación a la base de datos se verifica si el usuario es un invitado. Si lo es re-direcciona la aplicación a una página de consulta para estudiantes interesados en saber su estatus de préstamos.

```
classMainPage(Handler):

      defget(self):
             self.render("principal.html",usuario="",contrasena="",error="")

      defpost(self):

            Usuario_in =self.request.get("usuario")
            Contrasena_in =self.request.get("contrasena")

             if Usuario_in and Contrasena_in:
                    usuario_valido =db.GqlQuery("SELECT * FROM Datos_Usuarios WHERE usuario=
                    :1          and contrasena= :2",Usuario_in,Contrasena_in)
                     usuario_actual = usuario_valido.get()

             if Contrasena_in =="invitado"and Usuario_in =="invitado":
                    self.redirect("/Consulta_E")

             elif usuario_actual ==None:
                    error="Usuario o Contrasena no Validos Verificar"
                    self.render("principal.html",error=error )

             else:
                    fecha_in =datetime.datetime.today()
                    usuario_actualizado = Datos_Usuarios ( usuario = Usuario _in ,contrasena
                    =Contrasena_in,fecha=fecha_in)
                    db.delete(usuario_actual)
                    usuario_actualizado.put()
                    self.redirect("/Menu_Principal")
             else:
                    error="Introduzca Todos sus Datos"
                    self.render("principal.html",error=error)
```

Figura 4.7. Definición de la Clase MainPage.

4.7. LA CLASE MENU_PRINCIPAL_PAGE.

Esta clase maneja la página del menú principal. Primero accede a la base de datos de los usuarios para mostrar la fecha de acceso del usuario. Seguidamente procede con un lazo if para comparar la opción seleccionada por el usuario. De acuerdo a la opción marcada esta direccionara a la página seleccionada por el usuario. La cual tiene 9 opciones, para cada una de los procesos que el sistema ofrece. La definición de esta clase se muestra en la Figura 4.8. Para re-direccionar la aplicación se hace uso del método *redirect* del objeto *self* como se ve en cada uno de los casos de la clase de la Figura 4.8.

```
classMenu_Principal_Page(Handler):
        defget(self):
                usuarios_in = Datos_Usuarios.all()
                usuarios_in.order('-fecha')
                usuario_in = usuarios_in.get()
                timet=usuario_in.fecha
                fechat=timet.strftime("%d/%m/%Y a las %H:%M")
                self.render("menuprincipal.html", usuario=usuario_in.usuario,fecha=fechat)
        defpost(self):
                Opcion =self.request.get("Opcion")
                if Opcion =='1':
                        self.redirect("/Ver_Prestamo")
                elif Opcion =='2':
                        self.redirect("/Nuevo_Prestamo")
                elif Opcion =='3':
                        self.redirect("/Borrar_Prestamo")
                elif Opcion =='4':
                        self.redirect("/Nuevo_Estudiante")
                elif Opcion =='5':
                        self.redirect("/Borrar_Estudiante")
                elif Opcion =='6':
                        self.redirect("/Nuevo_Articulo")
                elif Opcion =='7':
                        self.redirect("/Borrar_Articulo")
                elif Opcion =='8':
                        self.redirect("/Estadisticas_Prestamos")
                else:
                        self.redirect("/Acerca_De")
```

Figura 4.8. Definición de la Clase Menu_Principal_Page.

4.8. LA CLASE VER_PRESTAMO_PAGE.

El método post de esta clase inicia leyendo el carnet introducido por el usuario en el formulario HTML. Luego, accede a la base de datos de Prestamos_Actuales, con el carnet introducido, si encuentra una coincidencia muestra los datos contenidos en esa entidad y accede con ese carnet a la base de datos de Datos_Estudiantes para mostrar la información del estudiante. De no encontrar coincidencias en la base de datos Prestamos_Actuales muestra un mensaje que no se tiene préstamos pendientes. Esta clase utiliza en su método post utiliza un lazo *for* para mostrar todas las coincidencias que se tienen en la base de datos Prestamos_Actuales. La definición de esta clase se muestra en la Figura 4.9. En la Figura de 4.9 se muestra la forma en cómo se modificó el formato de la fecha para que esta se presentara de manera correcta. El procedimiento es sencillo. Se extrae el objeto *datetime* de la base de datos, generado al crear el prestado, y se le resta otro objeto

datetime. Éste coincide en el día, mes y año, pero difiere en las horas que para el objeto a restar son 6 horas para logar la coincidencia con la hora local. La resta genera un objeto *deltatime* que luego es convertido a *string* para ser mostrado en la interfaz web.

```
classVer_Prestamo_Page(Handler):
    defget(self):
            self.render("verprestamo.html")
    defpost(self):
            Carnet_in =self.request.get("Carnet_in")
            prestamo_in =db.GqlQuery("SELECT * FROM Prestamos_Actuales WHERE Carnet_P=
      :1",Carnet_in)
            prestamo=prestamo_in.get()

            if prestamo ==None:
                    Texto="El Estudiante no Tiene Prestamos Pendientes!!!"
                    self.render("verprestamo.html", Mensaje=Texto)
            else:
                    Texto=""
                    for prestamo_in in prestamo_in:
                          timet=prestamo_in.Fecha_P
                          fechat=timet.strftime("%d/%m/%Y a la %H:%M")
                          Textot="El Estudiante "+prestamo_in.Estudiante_P+" Tiene Pendiente
                     de Entrega "+prestamo_in.Descripcion_AP+" Prestado el "+str(fechat)+"\n\n"
                          Texto= Textot+Texto
                    alumno_in=db.GqlQuery("SELECT * FROM Datos_Estudiantes WHERE Carnet=
                :1",Carnet_in)
                    alumno=alumno_in.get()
                    direccion=alumno.Direccion
                    Textof = Texto+"La direccion del estudiante es: "+direccion
                    self.render("verprestamo.html", Mensaje=Textof)
```

Figura 4.9. Definición de la Clase Ver_Prestamo_Page.

4.9. LA CLASE NUEVO_PRESTAMO_PAGE.

Esta es la clase con más accesos a la base de datos, y la que presenta una definición más extensa es mostrada en la Figura 4.10. El método post de esta clase comienza leyendo los datos introducidos en el formulario HTML, y calculando la fecha actual, con los datos del carnet del estudiante que realiza el préstamo accede a la base de datos Datos_Estudiantes para leer los datos de ese estudiante, y con el dato del número de inventario accede a la base de datos Datos_Bodega para obtener la información del artículo a prestar. Si no encuentra coincidencias en una o ambas bases de datos envía un mensaje de error. Si el artículo no está disponible envía un mensaje de que el artículo ya ha sido prestado. De lo contrario calcula el mes actual, con la función date.today() de Python para sumar a la estadística de ese mes, accediendo a la base de datos Base_Datos_Emes y borra el valor anterior y coloca el nuevo valor en ese mes. Luego cambia el actualiza el estado de préstamo del artículo seleccionado, realizando un acceso a la base de datos Datos_Bodega. Luego actualiza la fecha del último préstamo del estudiante con un acceso a la base de datos Datos_Estudiantes. Finaliza creando una entidad en la base de datos Prestamos_Actuales, que contiene la información del estudiante y del artículo prestado. Y envía un mensaje confirmando que el préstamo se realizó.

```
classNuevo_Prestamo_Page(Handler):

      defget(self):
              self.render("agregarprestamo.html")
      defpost(self):
              Carnet_in =self.request.get("Carnet_in")
              Codigo_in =self.request.get("Codigo_in")
              Materia_in =self.request.get("Materia")
              Fecha_in =datetime.datetime.now()
              estudiante_in =db.GqlQuery("SELECT * FROM Datos_Estudiantes WHERE Carnet=
          :1",Carnet_in)
              estudiante=estudiante_in.get()
              articulo_in =db.GqlQuery("SELECT * FROM Datos_Bodega WHERE Numero_Inventario=
          :1",Codigo_in)
              articulo=articulo_in.get()
              if articulo ==Noneor estudiante ==None:
                        Texto="El Estudiante o el Articulo no Estan en la Base de Datos!!!"
                        self.render("agregarprestamo.html", Mensaje=Texto)
              elif articulo.Disponible ==False:
                        Texto="El Articulo Ya ha Sido Prestado!!!"
                        self.render("agregarprestamo.html", Mensaje=Texto)
              else:
                        diaa= date.today()
                        if diaa.month ==1:
                              Mes_in="Enero"
                        elif diaa.month ==2:
                              Mes_in="Febrero"
                        elif diaa.month ==3:
                              Mes_in="Marzo"
                        elif diaa.month ==4:
                              Mes_in="Abril"
                        elif diaa.month ==5:
                              Mes_in="Mayo"
                        elif diaa.month ==6:
                              Mes_in="Junio"
                        elif diaa.month ==7:
                              Mes_in="Julio"
                        elif diaa.month ==8:
                              Mes_in="Agosto"
                        elif diaa.month ==9:
                              Mes_in="Septiembre"
                        elif diaa.month ==10:
                              Mes_in="Octubre"
                        elif diaa.month ==11:
                              Mes_in="Noviembre"
                        else:
                              Mes_in="Diciembre"
                        #maneja la base de datos de las estadisticas
                        Mes_sol =db.GqlQuery("SELECT * FROM Base_Datos_Emes WHERE Meses_ES=
                    :1",Mes_in)
                        Mes_temp=Mes_sol.get()
                        Cantmes=Mes_temp.Cantidad_Mes
                        Cantidad_nueva=1+Cantmes
                        db.delete(Mes_temp)
                        nueva_EsMes = Base_Datos_Emes(Meses_ES=Mes_in,Cantidad_Mes=Cantidad_nueva )
                        nueva_EsMes.put()
                        #maneja la base de datos de los prestamos
                        actualiza_articulo = Datos_Bodega(Numero_Inventario=
                    articulo.Numero_Inventario, Descripcion=articulo.Descripcion,
                    Marca=articulo.Marca,Modelo=articulo.Modelo, Serie=articulo.Serie,
                    Disponible=False,Fecha_UP=Fecha_in)
                        db.delete(articulo)
                        actualiza_articulo.put()
                        actualiza_estudiante =
                    Datos_Estudiantes(Carnet=estudiante.Carnet,Estudiante=estudiante.Estudiante,Dire
                    ccion=estudiante.Direccion,Fecha_UPE=Fecha_in)
                        db.delete(estudiante)
                        actualiza_estudiante.put()
```

```
                    nuevo_prestamo = Prestamos_Actuales(Carnet_P=estudiante.Carnet,
                Estudiante_P=estudiante.Estudiante,Materia=Materia_in,
                    Codigo_AP=articulo.Numero_Inventario,
                Descripcion_AP=articulo.Descripcion,Fecha_P=Fecha_in)
                    nuevo_prestamo.put()
                    Texto="Se ha Agregado a "+estudiante.Estudiante+" El Prestamo de
                "+articulo.Descripcion
                    self.render("agregarprestamo.html",Mensaje=Texto)
```

Figura 4.10. Definición de la Clase Nuevo_Prestamo_Page.

4.10. LA CLASE BORRA_PRESTAMO_PAGE.

Es el complemento de la clase anterior, su método post comienza leyendo los datos del estudiante y artículo que se desean borrar, y accede a la base de datos Prestamos_Actuales, si no encuentra coincidencias muestra un mensaje de error que no hay prestamos pendientes. De lo contrario actualiza la disponibilidad del artículo en la base de datos Datos_Bodega, y borra la entidad que coincide con el carnet y artículo devuelto de la base de datos Prestamos_Actuales.

Estas dos clases se puede decir que son las clases principales del sistema, y son a las que el código LUA generado y que corre en el router Dragino tiene acceso. La definición de la clase Borra_Prestamo_Page se muestra en la Figura 4.11.

```
classBorrar_Prestamo_Page(Handler):

defget(self):
        self.render("borrarprestamo.html")

defpost(self):

        Carnet_in =self.request.get("Carnet_in")
        Codigo_in=self.request.get("Codigo_in")
        prestamo_in =db.GqlQuery("SELECT * FROM Prestamos_Actuales WHERE Carnet_P= :1 and
        Codigo_AP= :2",Carnet_in,Codigo_in)
        prestamo=prestamo_in.get()

        if prestamo ==None:
                Texto="No se Encuentran Prestamos Pendientes!!!"
                self.render("borrarprestamo.html", Mensaje=Texto)

        else:
                articulo_in =db.GqlQuery("SELECT * FROM Datos_Bodega WHERE Numero_Inventario=
                :1",Codigo_in)
                articulo=articulo_in.get()
                actualiza_articulo = Datos_Bodega(Numero_Inventario= articulo.Numero_Inventario,
                Descripcion=articulo.Descripcion, Marca=articulo.Marca,Modelo=articulo.Modelo,
                Serie=articulo.Serie, Disponible=True,Fecha_UP=articulo.Fecha_UP)
                db.delete(articulo)
                actualiza_articulo.put()

                Texto="Se Ha quitado El Prestamo al Estudiante "+prestamo.Estudiante_P+" De
                "+prestamo.Descripcion_AP
                db.delete(prestamo)
                self.render("borrarprestamo.html", Mensaje=Texto)
```

Figura 4.11. Definición de la Clase Nuevo_Prestamo_Page.

4.11. PROCESO DE PUESTA EN FUNCIONAMIENTO DE LA APLICACIÓN WEB.

Después de haber generado el archivo de configuración, el archivo principal en Python, y las plantillas HTML de la aplicación web, se procedió a actualizar la aplicación web para ello se utiliza la herramienta appcfg.py de google app engine. Primero nos colocamos en el directorio donde está la carpeta creada para la aplicación, esta carpeta se pone en el mismo directorio donde del *SDK de Google App Engine* y tiene el nombre dado a la aplicación. El proceso se muestra en la Figura 4.12.

```
:~$ cd apps

:~/apps$ cd google_appengine

:~/apps/google_appengine$ python appcfg.py update lectorbarrasdragino/
```

Figura 4.12. Proceso para la puesta en servicio de la aplicación web.

4.12. MOSTRANDO IMÁGENES EN LA APLICACIÓN WEB, USANDO GOOGLE DRIVE.

Para mostrar imágenes se puede usar el servicio Google Drive que tiene como máximo de almacenamiento 5G. Para ello se coloca el URL asignado por Google Drive en la Opción background de body para el fondo o en la opción img de scr para desplegar la imagen, en las plantillas HTML.

Las imágenes por estar almacenadas en Google Drive para ser mostradas, aun cuando no se ha subido la aplicación al servidor de Google, necesitan de una conexión a internet para ser mostradas, de lo contrario no serán desplegadas, pero esto no genera error al momento de probar los códigos.

4.13. ACCESO A LAS BASES DE DATOS POR EL DASH BOARD.

Las bases de datos que se han creado para la aplicación son 5 y se muestran en la siguiente tabla con sus tamaños en Kbytes y número de entidades que contienen. Se puede tener acceso a esta información por medio del *Datastore Admin* de la aplicación, el cual resume todas las bases de datos. Esto se muestra en la Figura 4.13.

Entities — Entity statistics last updated Mar 14, 2013 3:21 a.m. UTC

Entity Kind	# Entities	Avg. Size/Entity	Total Size
Base_Datos_Emat	4	844 Bytes	3 KBytes
Base_Datos_Emes	12	826 Bytes	10 KBytes
Datos_Bodega	487	2 KBytes	1020 KBytes
Datos_Estudiantes	360	2 KBytes	565 KBytes
Datos_Usuarios	3	1 KByte	3 KBytes
Prestamos_Actuales	2	2 KBytes	4 KBytes

Figura 4.13. Resumen de las bases de datos generadas para la aplicación GAE, como se muestran desde el data admin de la Dashboard.

Para borrar la base de datos desde el local host se procede como en la Figura 4.14.

```
python dev_appserver.py --clear_datastore lectorbarrasdragino/
```

Figura 4.14. Proceso para borrar la base de datos del local host.

4.14. PASOS PREVIOS PARA LA DESCARGA Y CARGA MASIVA DE DATOS.

Para hacer efectiva la descarga y carga de las base de datos se dan los siguientes pasos:

PASO1: Al tener una base de datos la hoja de cálculo. Se procedió a pasarla a formato CSV, tipo de fichero de formato abierto sencillo para representar datos en forma de tabla, en las que las columnas se separan por comas y las filas por saltos de línea.
PASO2: Modificar el fichero `app.yaml` añadiendo las líneas mostradas en la Figura 4.15 si no se ha hecho aún.

```
builtins:

- remote_api: on
```

Figura 4.15.Líneas a colocar en el archivo yaml paraInstalar el remote_api mediante la directiva builtins.

PASO3: Actualizas la aplicación con el nuevo archivo `app.yaml`. Como se mostró en la Figura 4.6.
PASO4: Para comenzar la subida masiva de datos a la aplicación web primero se debe tener creado por lo menos una entidad en cada una de las bases de datos que se quiera manejar.
PASO5: Es necesario, se puede crear el archivo `bulkloader.yaml` para configurar la forma en que los datos serán guardados o leídos. Estos pueden estar en formato Sqlite, XML o CSV, además con este archivo se puede controlar las transformaciones de los datos para guardarlos en tipos de datos completamente manejables. La siguiente línea mostrada en la Figura 4.16 creará el archivo automático con la configuración por defecto para la descarga y carga de los datos para su posterior manipulación.

```
python appcfg.py create_bulkloader_config --filename=bulkloader.yaml –
url=http://lectorbarrasdragino.appspot.com/_ah/remote_api
```

Figura 4.16. Línea para crear el archivo de bulkloader.yaml.

Los parámetros básicos utilizados en la Figura 4.16 son:

`--filename:` Opción para colocar el nombre del archivo de configuración de la carga y descarga de los datos, el nombre puede ser cualquiera pero para tener efecto se debe pasar ese nombre a la opción.

`--file_config:` Cuando se realiza la descarga o carga.

`--url:` Se coloca la dirección del `remote_api` que maneja las bases de datos activada anteriormente, unido al `url de` la aplicación.

Al final del proceso se habrá creado en la carpeta principal de SDK un archivo `bulkloader.yaml.`

Se puede pasar una dirección que complete el nombre para seleccionar la carpeta donde se cree el archivo de configuración pero se debe usar siempre esa dirección al utilizar el archivo de configuración.

4.15. DESCARGA MASIVA DE DATOS DESDE LA APLICACIÓN.

PASO1: Para descargar datos se corre en la ubicación del SDK la línea de comandos mostrada en la figura 4.17.

```
python appcfg.py download_data --application=s~lectorbarrasdragino --
config_file=bulkloader.yaml --url=http://lectorbarrasdragino.appspot.com/_ah/remote_api --
kind=Datos_Bodega --filename=basededatosapp
```

Figura 4.17. Línea para descargar masivamente una base de datos desde la aplicación web.

Los parámetros básicos utilizados en la Figura 4.17 son:

--`application`**:** ID de la aplicación, si se ha creado como *High Replication* se debe agregar al id "`s~`", si no mostrara un mensaje de error, sobre la validación del acceso a la aplicación. En general las aplicaciones creadas tienen la opción *Higth Replicaction* por defecto así que agregar "s~" se hace necesario.

--`config_file:` Este parámetro indica el fichero de configuración, si no se coloca la herramienta configura la descarga con parámetros por defecto. Esto genera archivos *sqlite* que solo pueden ser manejados por este programa, así como tipos de datos por defecto que muchas veces no coinciden con los esperados.

Por lo que es recomendable siempre pasar este archivo, si se coloca en la carpeta principal del SDK solo colocamos el nombre, si no se debe pasar toda la ruta al archivo.

--`url:` Se coloca en este parámetro la dirección de la consola de descarga de datos relacionada a nuestra a nuestra aplicación. Como se nota por `fecto rd /_ah/remote_api.`

--`kind`**:** Este parámetro indica que base de datos se quieren descargar, debe tener el nombre que se le ha dado en el código, para poder bajarlas debe por lo menos haberse creado una entidad por cada base de datos.

--`filename`**:** Indica el nombre del archivo donde se guardaran los datos. Si no se coloca una ruta para el archivo este es creado en la carpeta principal del SDK, si no en la ruta especificada.

Es recomendable que --filename tenga cierta referencia con el nombre de la base de datos que se ha seleccionado para evitar confusiones.

Aunque se pueden bajar las bases de datos en un solo fichero esto generaría confusiones al momento de trabajarlas así que es recomendable bajarlas una por una asignándolas a diferentes archivos.

4.16. CARGA MASIVA DE DATOS A LA APLICACIÓN.

Para subir una base de datos se debe tener la base en formato CSV, esto hace más fácil el procedimiento. Este archivo debe tener el mismo formato que las entidades creadas, es decir los mismos atributos, por ejemplo si se tiene una identidad con atributos de nombre y edad, los archivos CSV deben tener esos mismos atributos. Además la base de datos debe tener por lo menos una identidad creada. Para comenzar a subir los datos se corre desde la ubicación principal del SDK la línea de comandos mostrada en la Figura 4.12:

```
python appcfg.py upload_data --config_file=bulkloader.yaml --filename=BD_alumnos.csv --
kind=Datos_Estudiantes --url=http://lectorbarrasdragino.appspot.com/_ah/remote_api
```

Figura 4.18. Línea para cargar masivamente una base de datos a la aplicación web.

Los Parámetros Básicos son los Siguientes de la herramienta para la carga masiva de datos mostrada en la Figura 4.18 son:

--config_file: Este parámetro indica el archivo de configuración, es importando porque en dicho archivo se indica a la herramienta appcfg.py qué tipo de archivo se subirá, para el caso CSV. Si el archivo está en la carpeta principal del SDK solo se necesita pasar el nombre si no se tiene ahí ruta completa al archivo de configuración.

--filename: Con este parámetro se le indica el archivo que contiene nuestra base de datos, para el caso en formato CSV, pasamos solo el nombre del archivo si este se encuentra en la carpeta principal del SDK o toda la ruta a dicho archivo si está ubicado en otra carpeta.

--kind: Este parámetro sirve para indicarle a la herramienta de carga de datos, a que entidad de las que base de datos de las creadas se deben de subir las entidades que contiene el archivo CSV, hay que estar seguros que la base de datos tenga relación con el archivo CSV que vamos a subir para evitar errores como entidades que no tienen relación.

--url: Este parámetro indica la dirección de la aplicación remota que maneja los datos de nuestra aplicación, que es la dirección de la aplicación más la siguiente ruta /_ah/remote_api.

El url/_ah/remote_api: Es la ruta por defecto para la aplicación remota, pero puede ser cambiada en el archivo app.yaml, de ser así se debe cambiar este URL también en la línea de comandos.

Iniciando el proceso se pedirá las credenciales de administrador que son el correo asociado a nuestra cuenta y también la contraseña de nuestra cuenta. Luego se iniciara el proceso de carga de los datos. Al terminar este proceso se deberá de leer un mensaje similar mostrado en la Figura 4.19.

```
[INFO    ] 360 entities total, 0 previously transferred

[INFO    ] 360 entities (241792 bytes) transferred in 34.8 seconds

[INFO    ] All entities successfully transferred
```

Figura 4.19. Mensaje Mostrado al finaliza la Carga Masiva de Datos.

4.17. ENVIADO UNA VARIABLE A JAVASCRIPT PARA MOSTRAR UNA GRÁFICA EN GOOGLE CHART TOOLS.

Para mostrar las estadísticas se utilizó la biblioteca de Google Chart Tools, escrita en JavaScript. Se utilizó el ejemplo que instruye como dibujar un gráfico de barras [16]. El código se incrustó en la página HTML generada para mostrar los estadísticos esto dentro de la etiqueta head, como se muestra en la Figura 4.20. En el controlador principal en Python se acceso a la base de datos de las estadísticas. Se extrajeron los datos de todos los meses. Se formó un *string* nombrado a. Este *string* se envió por medio del método post hacia la página HTML, pasado al JavaScript utilizando '{{a}}'. De esta manera, se logró comunicar ambos lenguajes. El *string* en JavaScript se dividió en un arreglo de *string*. Luego se convierte cada uno de los elementos de los arreglos en flotantes.

```
!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="content-type" content="text/html; charset=utf-8"/>
<title> Estadisticas de Bodega  </title>
   <!--Load the AJAX API-->
<script type="text/javascript" src="https://www.google.com/jsapi"></script>
<script type="text/javascript">
google.load('visualization', '1.0', {'packages':['corechart']});
google.setOnLoadCallback(drawChart);
function drawChart() {
//se lee un string desde python
                var datos = '{{a}}';
                //se divide el string donde esten las comas
                var datosseparados=datos.split(",");
                //cada sub divicion se pasa a numero
                var datom1=parseFloat(datosseparados[8]);
                var datom2=parseFloat(datosseparados[7]);
                var datom3=parseFloat(datosseparados[4]);
                var datom4=parseFloat(datosseparados[11]);
                var datom5=parseFloat(datosseparados[3]);
                var datom6=parseFloat(datosseparados[5]);
                var datom7=parseFloat(datosseparados[6]);
                var datom8=parseFloat(datosseparados[10]);
                var datom9=parseFloat(datosseparados[0]);
```

```
                var datom10=parseFloat(datosseparados[1]);
                var datom11=parseFloat(datosseparados[2]);
                var datom12=parseFloat(datosseparados[9]);
var data = new google.visualization.arrayToDataTable([
['Mes','Prestamos'],
    ['Enero',datom1],
    ['Febrero',datom2],
    ['Marzo',datom3],
['Abril',datom4],
    ['Mayo',datom5],
    ['Junio',datom6],
['Julio',datom7],
    ['Agosto',datom8],
    ['Septiembre',datom9],
    ['Octubre',datom10],
    ['Noviembre',datom11],
['Diciembre',datom12]]);
var options = {'title': "Cantidad de Prestamos por Mes",
'width':900,
            'height':400,
var chart = new google.visualization.ColumnChart(document.getElementById('chart_div'));
chart.draw(data, options);
  }
</script>
</head>
```

Figura 4.20. Sección del Código HTML para mostrar la Grafica Utilizando Google Chart Tools.

La variable *options* en el JavaScript sirve para modificar las características de la gráfica como color y tamaño, además del fondo y si dar la opción que presente *grid,* los nombres de los ejes etc. Cambiando el tipo en la variable chart se obtiene varios tipos de graficas como pastel, barras horizontales, entre otras.

La variable data, hace uso del método *arrayToDataTable*. Transforma un arreglo de datos, a una tabla que la herramienta de Google Charts Tools pueda manejar. Hay otros métodos para crear estas tablas de datos que Google necesita para presentar los datos pero este es el más sencillo. Para mayor referencia visitar el sitio de Google Chart Tools, donde se encuentran todas las posibles graficas que se pueden.

La Figura 4.21 muestra la definición de la clase *Estadisticas_Page.* Se observa cómo se formó el *string* a para enviárselo al JavaScript de Google Chart Tools. Primero se extraen todas las entidades de la base de datos ordenados según el mes. Este arreglo retorna los meses ordenados de manera alfabética, junto con los valores de préstamos para cada uno. Luego con un lazo *for* se agrega a un *string* vacío los valores de cada mes, y se separan con comas, para luego ser enviados en el *string* a por medio del método post.

```
classEstadisticas_Page(Handler):

        defget(self):
                self.render('estadisticas.html')

        defpost(self):

                 Texcantmf=""E_meses =db.GqlQuery("SELECT * FROM Base_Datos_Emes
                 ORDER BY Meses_ES")
                 for E_meses in E_meses:

                        cantmes=E_meses.Cantidad_Mes
                        Texcnt=str(cantmes)+","
                        Texcantmf=Texcnt+Texcantmf

                        arreglo=Texcantmf
                        self.render("estadisticas.html",a=arreglo)
```

Figura 4.21. Definición de la Clase Estadisticas_Page.

5.0 LINEAS FINALES Y CONCLUSIONES

5.1 LINEAS FINALES

El programa para el control de bodega presentado es simple, y no aborda temas de control de pérdida de datos, un sistema más robusto, requiere que este tema sea tratado con más profundidad, en futuras aplicaciones los códigos pueden ser dotados de verificación de errores de conexión al servicio web, explotando las características de la función cURL mostrada en el apartado 3.3.

Los temas de seguridad de la base de datos, así como protección de las cuentas pagas ser profundizados, para brindar una opción segura para aplicaciones más específicas.

Otra línea de investigación es la reducción de los costos del dispositivo, evaluando la posibilidad de eliminar uno de los dos dispositivos que conforman el sistema, explorar estas posibilidades podría resultar en una reducción considerable en el costo del sistema.

El sistema no está restringido para ser aplicado en el inventario de una bodega, las posibilidades son muchas y abarcan aplicaciones con el mismo tipo de sensor, así como otras que involucren el uso de sensores para otros fines como monitoreo del consumo eléctrico, niveles de agua, sensores de temperatura, entre otros. Se presentan a continuación algunas posibilidades:

5.1.1 INVENTARIO DE BILIOTECAS

La utilización del código de barra es de gran importancia en una biblioteca así como sirve para los préstamos de los libros hacia los alumnos, así también es de gran importancia en el control de inventario, la aplicación de Raspberry Pi sirve para los dos casos, el servicio web será puesto para la biblioteca teniendo una ventaja ya que el Raspberry Pi utiliza salida HDMI y RCA se puede colocar una pantalla, para tener la interfaz gráfica, y poder ver en forma real el proceso de préstamo y devoluciones.

PRÉSTAMOS Y DEVULUCIONES

El lector de barra conectado a la Raspberry Pi y a la vez con router Dragino, puede ser utilizado para el préstamo de los libros teniendo en cuenta el carnet del estudiante ó docente, todos los libros estarán etiquetados con códigos de barra como lo muestra la Figura 5.1.

La aplicación web contendrá la hora, fecha, carnet, y numero de inventario del libro para cada préstamo. El entorno gráfico será por una pantalla, un teclado para observaciones o clientes externos, un mouse para poder desplazarse en pantalla y una impresora para dar un ticket de préstamo, el cual contiene el nombre del libro, fecha, hora, carnet, nombre y fecha que tendrá que devolver el libro.

Figura 5.1. Muestra la aplicación del lector de barra para una biblioteca.

CONTROL DE INVENTARIO

Para el control de inventario será necesario un lector de barra inalámbrico que esté conectado vía Wi-Fi al Raspberry Pi, el lector de barra almacenara los datos en el lector y cuando este cerca del Raspberry Pi detecte la señal y descargue automáticamente los datos en el mismo, la conexión con el router Dragino será para enlazar la página que maneja la base de datos de los inventarios. La Figura 5.2 muestra el ejemplo de cómo se podría hacer este procedimiento.

Figura 5.2. Muestra la aplicación del lector de barra para una biblioteca para control de inventario.

5.1.2 CONTROL AISLADO DE UNA ESTACIÓN METEOROLÓGICA

Teniendo en cuanta que la Raspberry Pi cuenta con puertos GPIO este puede recibir varias entradas de datos, debido a que una estación meteorológica cuanta con instrumentos como los mostrados en la Figura 5.3, termómetro, barómetro, termógrafo, pluviómetro, higrómetro, heliógrafo, anemómetro, veleta, nefobasimetro-[17]. Cada uno de ellos significaría una entra al puerto GPIO del Raspberry Pi en alguno de los casos se tendría que hacer la electrónica necesaria para poder obtener el nivel de 3.3v que necesita la Raspberry Pi para poder leer los datos de los instrumentos.

Figura 5.3. Muestra la aplicación del lector de barra para una biblioteca para control de inventario.

Los datos mandados al Raspberry Pi tendiendo un Modem USB para conexión a internet mostrando los procesos de proyecto en la Figura 5.4, mandará los datos a un servicio web donde almacenara los datos automáticamente.

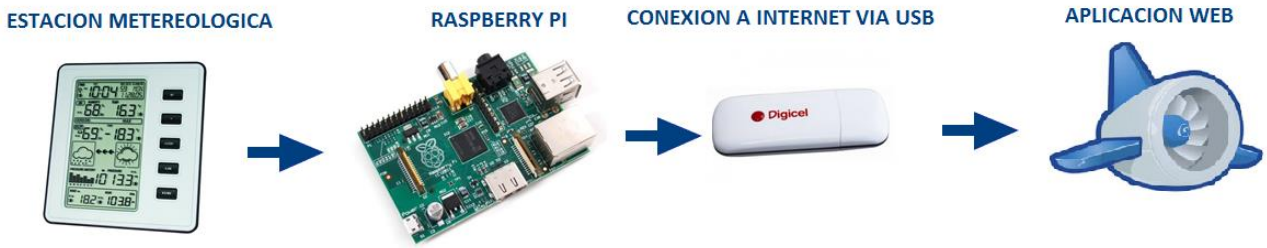


Figura 5.4. Muestra la aplicación del Raspberry Pi como una estación meteorológica.

5.1.3CONTROL DE ENERGÍA EN RESIDENCIAS Y SUBESTACIONES

Dotar a la Raspberry Pi de un shield para la lectura y medición de valores de medición eléctrica tomando datos de hora, fecha, voltaje, corriente, potencia, etc. Teniendo una aplicación web siempre con Google App Engine donde mostrara los valores en tiempo real, así como los valores al mes y al año. La figura 5.5 muestra la posible adaptación del shield en la Raspberry Pi.

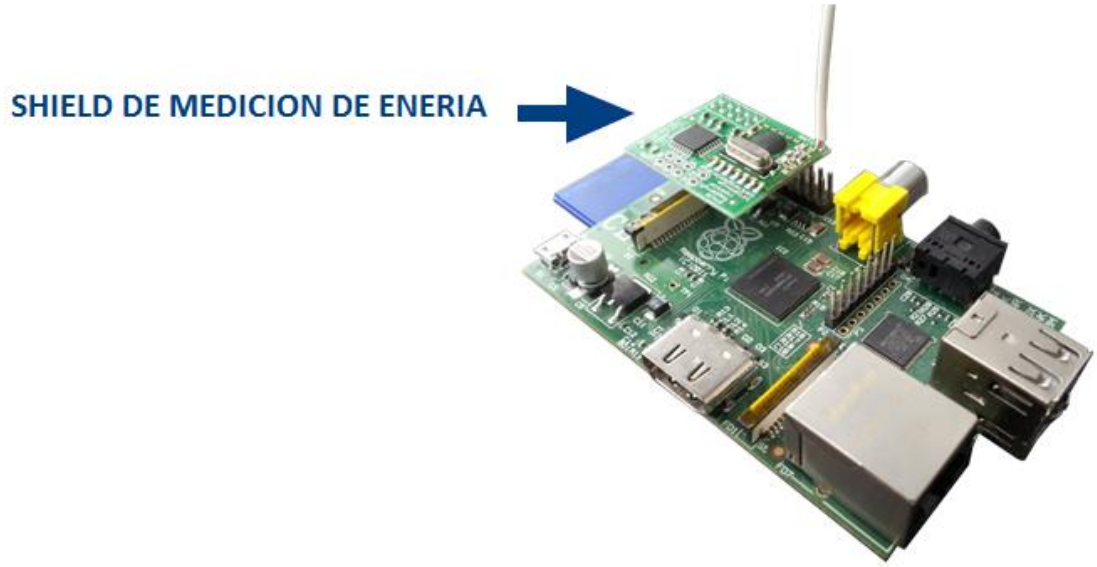


Figura 5.5. Muestra la aplicación del lector de barra para una biblioteca para control de inventario.

5.2 CONCLUSIONES TECNICAS.

Dentro de todo el trabajo de graduación se llegó a la culminación de la interfaz inalámbrica para un lector de código de barra. La tarea comenzó desde Proyecto de Ingeniería, donde la idea de un lector de barra inalámbrico a bajo precio podía ser una realidad, mediante la aplicación del router Dragino y un lector de barra comercial. Durante ese proceso se tuvo la limitante de que el lector de barra su comunicación necesitaba de una computadora para encender y emular a RS232 para comunicarse con el router Dragino. Se hicieron una serie de circuitos para solucionar ese problema, el resultado no fue el esperado, por lo que se tomó en cuenta utilizar un SCB. El Raspberry Pi fue uno de los más indicados para solucionar el problema de conectividad y dar un mejor soporte de comunicación con el router Dragino.

En la etapa de Hardware, las ventajas del router Dragino utilizando Raspberry Pi fueron: la comunicación entre los dos dispositivos fue mediante la comunicación UART, entre los dos dispositivos su valor de voltaje en las entradas de los puertos es de 3.3V. El entorno gráfico del Raspberry Pi fue más amigable, esto gracias a la conexión HDMI y VGA. Se utilizó un LCD para poder mostrar los datos y requerimientos que se necesitaban para hacer las peticiones de los préstamos.

La interfaz inalámbrica utilizada fue totalmente con el router Dragino. En la actualidad existen routers con capacidad USB, estos se utilizan para impresoras, dispositivos de comunicación Wi-Fi y como transferencia de archivos (Discos duros externos). La mayoría de ellos tienen su marca registrada y no tienen posibilidad de cambiar su firmware. Por lo que encontrar un router con capacidad de conexión USB para un lector de barra es muy difícil. Hacer un shield USB para el Dragino podría ser una buena alternativa para independizar al router del Raspberry Pi.

La Raspberry Pi pudo haber sido implementada en toda la aplicación sin necesidad del router Dragino. La Raspberry Pi se puede conectar a un adaptador Wi-Fi, mediante los puertos USB directamente desde la Raspberry Pi. Esto hubiera eliminado al router Dragino y bajar el precio de la aplicación, así como también el espacio físico que ocupa el router Dragino.

La aplicación Web de Google App Engine fue de gran ayuda para el almacenamiento de la base de datos y la aplicación del sitio Web. Google App Engine entre sus ventajas se encuentran: tiene la capacidad de trabajar con varios tipos de programación entre ellos Python, el cual se ha venido trabajando en el curso de este trabajo de graduación. La aplicación Web tiene su propio dominio, el cual lo hace una aplicación atractiva. No hay necesidad de preocuparse por el Hardware, ya que todo está en la nube. Su implementación es muy rápida y se generan copias de seguridad, esto reduce grandemente los gastos servidores locales y personal de manteniendo el sitio web. En comparación, si la aplicación Web estuviera instalada en los servidores de la Universidad de El Salvador, tendría varias diferencias con un servicio como Google App Engine entre ellos: al no tener energía eléctrica o servicio de internet en donde se encuentra el servidor Web, los usuarios dentro y fuera de la Universidad no podrían ver la aplicación. Otra diferencia es que necesitaría personal para mantenimiento, seguridad y administración de la aplicación, Google App Engine tiene varias certificaciones (SAS 70 Tipo II) y auditorias (SSAE16) para controlar el modo de seguridad de sus clientes.

5.3 CONCLUSIONES FINANCIERAS.

El costo total del sistema fue de $190.02, este precio se desglosa en la Tabla 5.1a y Tabla5.1b de este apartado, mientras que si se hace el mismo diseño con una computadora personal que agregue un servidor local y que haga las mismas características de la interfaz desarrollada en este trabajo de gradación.

RUBRO	COSTO
ROUTER DRAGINO	$ 65.56
RASPBERRY PI	$ 62.46
LECTORA METOLOGIC	$ 5.00
CAJA DE ACRILICO	$ 17.00
CIRCUITERIA AUXILIAR	$ 10.00
MANO DE OBRA	$ 30.00
TOTAL	**$ 190.02**

a)

RUBRO	COSTO
COMPUTADORA PERSONAL COMPLETA	$ 500.00
LECTOR DE BARRA	$ 5.00
INSTALACION DE TOMAS E INTERNET	$ 30.00
INSTALACION DE UN SERVIDOR LOCAL	$ 100.00
TOTAL	**$ 635.00**

b)

Tabla 5.1. Se muestran las comparaciones entre la interfaz desarrollada en el trabajo de graduación y el mismo diseño con una computadora con punto de red.

Los precios presentados en la Tabla 5.1a pueden variar de acuerdo al tipo de envío, lugar de cotización, volumen de compra etc.

El costo de este dispositivo se compara con otras opciones posibles para el manejo de información como son dispositivos inalámbricos existentes en el mercado, y la posibilidad de ubicar un punto de red inalámbrico con una computadora.

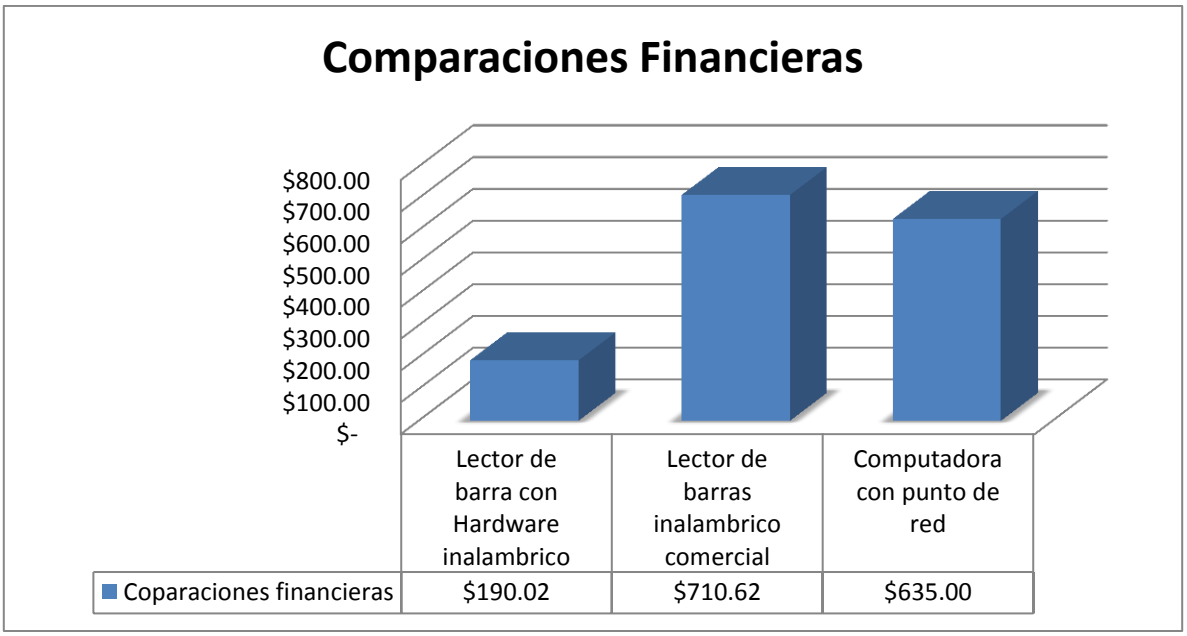


Figura 5.6. Muestra la Comparación de costos.

De la evaluación de los costos mostrados en la Figura 5.6, se concluye que es posible generar un dispositivo funcional que a menor costo comparado con las otras posibilidades. Es un 73.26% menor que un dispositivo similar comercial, y es un 70.08% menor en costo que una computadora con un punto de red.

Por lo que una inversión de en un sistema similar al mostrado representa una buena opción de inversión a corto plazo.

RECOMENDACIONES

Dentro del trabajo de graduación surgieron algunas recomendaciones sobre que se debería de hacer en algunos casos, dentro de los cuales se mencionan alguno de ellos a continuación:

VERIFICACIÓN DE ERRORES

En la parte del posteo en el capítulo 3. El encargado de laboratorio no puede saber si lo que se envió realmente está guardado en la base de datos y el préstamo fue efectivo o no. Por lo que se recomienda crear una forma de programación el cual el encargado de laboratorio pueda visualizar si el préstamo fue efectivo, esto se hará por medio de mensajes de error que podrá mandar la aplicación web, por ejemplo mostraremos algunos posibles errores y sus significados.

ERROR 0: Perdida de comunicación Wi-Fi.
ERROR 1: Base de datos llena.
ERROR 2: La lectura fue exitosa.
ERROR 3: El estudiante este saldo rojo.
ERROR 4: Si llega el mensaje hasta el servidor pero GAE no responde.

La Figura 5.7 muestra una lectura que hace el router Dragino cuando no está conectado con una red Wi-Fi, este problema hace que la interfaz desarrollada no muestre este tipo de errores por lo que es de gran ayuda la realización de la verificación de errores como es este caso. En la Figura el comando curl regresa a Dragino la frase:

"Couldn´t resolve host "lectorbarrasdragino.appspot.com""

Esta variable puede indicar al Raspberry Pi y mandar un mensaje a la pantalla LCD que muestre para el caso ERROR: 0, que significa perdida de comunicación WI-FI.

```
escriba una Materia
la materia es
TEL115
escriba un Articulo
el articulo es
1205026068020070
Pego en Nuevo Prestamo
curl: (6) Couldn't resolve host 'lectorbarrasdragino.appspot.com'
escriba un Articulo
el articulo es
fin
Pego en Nuevo Prestamo
curl: (6) Couldn't resolve host 'lectorbarrasdragino.appspot.com'
Escriba una accion:
```

Figura 5.7. Muestra al Mensaje que manda Dragino mostrada en la variable curl.

RASPBERRY PI WIFI

Debido a que actualmente la parte de la comunicación Wi-Fi es directamente por el Dragino. Este se puede minimizar dotando al Raspberry Pi de comunicación Wi-Fi por medio de un adaptador Wi-Fi, esto eliminaría la función del Dragino de dotar de comunicación inalámbrica. Como la Figura 5.8.

Figura 5.8. Muestra al Raspberry Pi con conectividad Wi-Fi.

Al utilizar la Raspberry se mejoraría la eficiencia ya que no hay la necesidad de mandar el mensaje a otro dispositivo, sino que se controlaría desde un solo código. El espacio físico utilizado reduciría grandemente ya que el espacio del router Dragino se eliminaría.

ANEXO A

A.1.0 CONCEPTOS BÁSICOS.

A1.1.1. WORLD WIDE WEB.

El *World Wide Web* (*wwww* por sus siglas en inglés) es una colección de documentos Hyper Text Markup Lenguage (HTML por sus siglas en ingles), y es la base de casi todas las páginas web. Los vínculos entre páginas se llaman *Hyper Links* o simplemente vinculos y son los que dan a la Internet su característica de red. La red fue inventada en la década de 1990 y actualmente cuenta con alrededor de 644 millones de sitios web activos.

A1.1.2. INTRODUCCIÓN AL HTML.

HTML es el corazón de la red. Está compuesto de: contenido, que es lo que se muestra; etiquetas, que dan forma al contenido, referencias a otros documentos, que pueden ser imágenes, videos y *links* o enlaces a otras páginas.

Además, de etiquetas y atributos, HTML define el término elemento para referirse a las partes que componen los documentos HTML. Aunque en ocasiones se habla de forma indistinta de "elementos" y "etiquetas", en realidad un elemento HTML es mucho más que una etiqueta, ya que está formado por: Una etiqueta de apertura, cero o más atributos, texto encerrado por la etiqueta, una etiqueta de cierre.

El texto encerrado por la etiqueta es opcional, ya que algunas etiquetas de HTML no pueden encerrar ningún texto. El siguiente esquema muestra un elemento HTML, formado por una etiqueta , atributos y contenidos de texto:

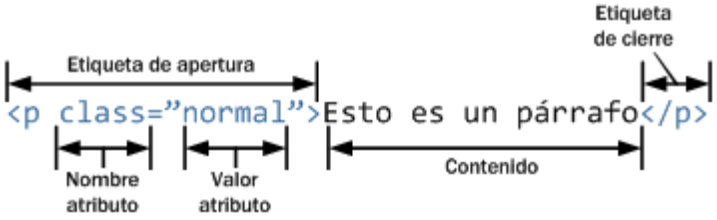


Figura.A1. 1. Esquema de las partes que componen un elemento HTML.

La estructura mostrada en la figura A1.1 es un elemento HTML, el elemento comienza con una etiqueta de apertura ( ), contiene el atributo, para el ejemplo *class="normal"* [1], dispone de un contenido de texto en le figura A1.1 “Esto es un párrafo” y finaliza con una etiqueta de cierre ( ).

Por tanto, si una página web tiene dos párrafos de texto, la página contiene dos elementos y cuatro etiquetas: dos etiquetas de apertura y dos etiquetas de cierre. De todas formas, aunque estrictamente no son lo mismo, es habitual intercambiar las palabras "elemento" y "etiqueta".

Por otra parte, el lenguaje HTML clasifica a todos los elementos en dos grupos: elementos en línea (inline elements en inglés) y elementos de bloque (block elements en inglés).

La principal diferencia entre los dos tipos de elementos es la forma en la que ocupan el espacio disponible en la página. Los elementos de bloque siempre empiezan en una nueva línea y ocupan todo el espacio disponible hasta el final de la línea, aunque sus contenidos no lleguen hasta el final de la línea. Por su parte, los elementos en línea sólo ocupan el espacio necesario para mostrar sus contenidos.

La mayoría de elementos de bloque pueden contener en su interior elementos en línea y otros elementos de bloque. Los elementos en línea sólo pueden contener texto u otros elementos en línea. En otras palabras, un elemento de bloque no puede aparecer dentro de un elemento en línea. En cambio, un elemento en línea puede aparecer dentro de un elemento de bloque y dentro de otro elemento en línea.

Los elementos en línea definidos por HTML son: `a, abbr, acronym, b, basefont, bdo, big, br, cite, code, dfn, em, font, i, img, input, kbd, label, q, s, samp, select, small, span, strike, strong, sub, sup, textarea, tt, u, var.`

Los elementos de bloque definidos por HTML son*:* `address, blockquote, center, dir, div, dl, fieldset, form, h1, h2, h3, h4, h5, h6, hr, isindex, menu, noframes, nos-cript, ol, p, pre, table, ul.`

Los siguientes elementos también se considera que son de bloque: `dd, dt, frame-set, li, tbody, td, tfoot, th, thead, tr.`

Los siguientes elementos pueden ser en línea y de bloque según las circunstancias: `button, del, iframe, ins, map, object, script.`

A1.1.3. HTTP.

Hyper Text Transfer Protocol (HTTP de las siglas en inglés), es el protocolo principal de la red. Es lo que los navegadores utilizan para comunicarse con los servidores. Una solicitud desde el navegador a una URL se hace como se muestra en la figura A1.2.

```
Htt4p://www.example.com/foo
```

Figura.A1. 2. Solicitud a una URL desde un navegador.

Comienza con una línea de solicitud como la que se muestra en la Figura A1.3.

```
GET /foo HTTP/1.1
```

Figura.A1. 3. Primera Línea de solicitud en HTTP.

HTTP es un protocolo de texto simple, y este texto se envía por la red exactamente como se mostró en la figura A1.3. Esta tiene 3 partes principales: el método (GET), la ruta (/foo) y la versión (HTTP/1.1)

Ahora que el servidor ha visto la petición HTTP, en vía una respuesta. La respuesta básica a la petición del ejemplo se mira como en la figura A1.4:

```
HTTP/1.1 200 OK
```

Figura.A1. 4. Primera Línea de solicitud en HTTP.

Esto es llamado línea de estado y es análoga a la línea de petición mostrada anteriormente: primero la versión (HTTP/1.1), seguido del código de estado (200) y por último la frase de respuesta (OK).

A1.1.4. APLICACIONES WEB.

El propósito de un servidor web es responder a las solicitudes HTTP. Hay dos tipos principales en las que se puede clasificar las respuestas de los servidores: estáticas y dinámicas.

El contenido se considera estático si es un archivo pre establecido que el servidor simplemente regresa. Un ejemplo son los archivos de imágenes. Más interesantes son las respuestas dinámicas, que son respuestas que se construyen en el momento por un programa que corre en el servidor. Casi todos los contenidos en línea hoy en día son dinámicos. El programa que construye estas respuestas dinámicas es llamado una aplicación web.

Una aplicación web es un programa que genera contenido. Este programa, corre en el servidor, se comunica con HTTP, y genera contenido dinámico solicitado por el buscador.

A1.2. GOOGLE APP ENGINE.

App Engine es un servicio de alojamiento web que presta *Google* de forma gratuita hasta determinadas cuotas, este servicio permite ejecutar aplicaciones sobre la infraestructura de *Google*. Si no se cuenta con un dominio propio, *Google* proporciona uno con la estructura Mostrada en la figura A1.5,

```
midominio.appspot.com
```

Figura.A1. 5. Estructura del dominio gratuito que proporciona Google.

También permite implementar un dominio propio a través de *Google Apps*. Por el momento las cuentas gratuitas tienen un límite de un gigabyte de almacenamiento permanente y la suficiente cantidad de ancho de banda y CPU para cinco millones de visitas mensuales, y si la aplicación supera estas cuotas, se pueden comprar cuotas adicionales.

Actualmente las aplicaciones *Google App Engine* se implementan mediante los lenguajes de programación *Python, Java y Go.*

El servicio fue lanzado el 7 de abril del 2008 como un servicio de *cloud* pero a diferencia de otros servicios en la nube como *Amazon Web Services* [2] o *Azure Services Platform [3]* de *Microsoft*, el servicio ofrecido por *Google* es un servicio de Plataforma como Servicio y no de Infraestructura como Servicio. [4]

El uso de la infraestructura de servicio de *Google App Engine* es completamente gratuito hasta un *Gigabyte* de almacenamiento y cinco millones de visitas mensuales. Si esos límites son superados entonces se tiene que pagar por más recursos a *Google* a unos precios bastante accesibles. Además se puede usar un dominio propio para la URL de la aplicación o bien se puede usar un subdominio de appspot.com ofrecido de manera gratuita por *Google* al estilo de *Heroku*.[5]

GAE soporta de manera oficial los lenguajes de programación *Python y Java* de manera estable y en modo de *beta testing* en lenguaje de programación Go creado por ellos mismos. Al soportar *Java*, es posible además utilizar cualquier lenguaje JVM o lo que es lo mismo, cualquier lenguaje que pueda ejecutarse sobre una máquina virtual de *Java*, aunque eso sí, con serias limitaciones. Entre dichos lenguajes se encuentran: *Groovy, JRuby, Scala, PHP, Clojure, Perl*.

A1.2.1. RESTRICCIONES.

- Las aplicaciones solo tienen permisos de lectura a los archivos del sistema de archivos. Para almacenar datos y archivos en modo lectura y escritura es necesario utilizar un sistema de archivos virtual sobre el *DataStore*.
- Solo se puede ejecutar código a través de consultas HTTP
- Las aplicaciones Java solo pueden usar el conjunto considerado seguro de clases del JRE estándar.
- Las aplicaciones no pueden crear nuevos hilos de ejecución
- Los usuarios de Python pueden subir módulos para su uso en la plataforma pero no aquellos que están completamente desarrollados en C o Pyrex
- El soporte para SSL solo está disponible par dominios *.appspot.com
- Un proceso iniciado en el servicio para responder a una consulta no puede durar más de treinta segundos
- No soporta sesiones persistentes, solo sesiones replicadas a las que además se les aplican ciertos límites.
- No se pueden abrir *sockets*, por lo tanto, no se puede usar *Twisted.*

A1.2.2. EL ENTORNO DE DESARROLLO.

El SDK (*Software Develope Kit)* de *Google App Engine* viene con un entorno de desarrollo que puede ser ejecutado en local en nuestra máquina antes de subir los cambios a la nube y que simula completamente el entorno del *App Engine*. El SDK puede ser descargado de la página de descargas del proyecto.[6]

Para ejecutar el servidor del entorno de desarrollo se debe ejecutar el *script* `dev_appserver.py` y como parámetro el *path* o ruta al directorio que contiene la aplicación. Por ejemplo, como se ve en la figura A1.6 donde se hace desde una consola de un sistema tipo UNIX.

```
$ cd apps/

$ cd google_appengine/

$ python dev_appserver.py directoriodelaaplicacion/
```

Figura.A1. 6. Línea para correr la aplicación como servidor local, utilizando el SDK de GAE en Ubuntu.

A1.2.3. EL ENTORNO DE EJECUCIÓN DE PYTHON.

App Engine ejecuta las aplicaciones mediante un intérprete de *Python* 2.5.2 cargado en un entorno "*sandboxed*". Toda aplicación programada para correr en la nube de *Google* debe incorporar un archivo de configuración llamado `app.yaml` donde se configuran diferentes aspectos de la aplicación y el entorno de ejecución.

El entorno de ejecución admite módulos de terceros programados íntegramente en *Python* y no deben incluir ninguna extensión C o cualquier otro código susceptible de ser compilado. El entorno incluye la librería estándar de *Python* a excepción de algunos módulos que han sido desactivados por seguridad como por ejemplo la escritura de disco o los *sockets*.

Otros módulos de *Python* han sido reemplazados o personalizados para hacerlos compatibles con *App Engine*. Un ejemplo de archivo de configuración `app.yaml` se muestra en la Figura A1.7. Para más una descripción detallada del archivo app.yaml remitirse al capítulo 4 del presente documento.

```
application: lectorbarrasdragino

version: 1

runtime: python27

api_version: 1

threadsafe: true

libraries:

- name: jinja2

  version: latest

handlers:

- url: /.*

  script: lectorbarrasdragino.app

builtins:

- remote_api: on
```

Figura.A1. 7. Estructura Básica del archivo de configuración app.yaml.

A1.2.4. CONSOLA DE ADMINISTRACIÓN DE LA APLICACIÓN.

La Consola del administrador de *Google App Engine* ofrece acceso completo a la versión pública de la aplicación. Para acceder a la Consola, se utiliza el siguiente enlace en un navegador web: https://appengine.google.com/

Accede a la cuenta de *Google* o crea una con una dirección de correo electrónico y contraseña.

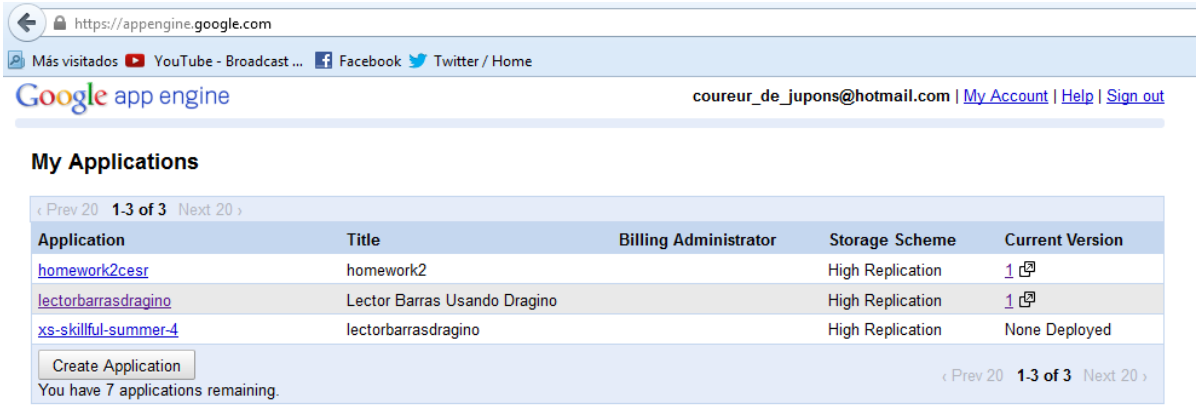


Figura.A1.8. Vista de la consola de administración de aplicaciones de GAE.

Luego de acceder se mostrara una página como la mostrada en la Figura A1.8 que muestra un ejemplo de una consola de administración de la aplicación, se observan en la Figura una tabla con el título *My Applications*, con un botón en la base para crear otra aplicación. Es de resaltar que solo se pueden crear 10 aplicaciones un una cuenta gratuita. En la Figura A1.8 se aprecian las columnas entre las principales esta *Application* que representa el identificador de la aplicación, *Title* que es un nombre descriptivo para la aplicación, además otras opciones como el administrador de facturación, si se activó esa opción al crear la aplicación, el esquema de almacenamiento y por último la versión de la aplicación.

La Consola del administrador sirve para:

- Crear una nueva aplicación y configurar un subdominio appspot.com gratuito o el nombre de dominio de nivel superior que se desee.
- Invitar a otras personas a que sean desarrolladores de la aplicación para que puedan acceder a la Consola y subir nuevas versiones del código.
- Ver datos de acceso y registros de errores, además de analizar el tráfico.
- Examinar el almacén de datos de tu aplicación y administrar los índices.
- Administrar las colas de tareas para permitir la pausa, la purga y la eliminación de colas.
- Administrar tareas concretas de una cola de tareas para permitir la visualización, eliminación o ejecución de tareas individuales de forma inmediata.
- Probar nuevas versiones de tu aplicación y cambiar la versión que ven tus usuarios.

A1.2. 5. FACTURACIÓN.

LÍMITES DE LA FACTURACIÓN.

Cada aplicación en *App Engine* puede consumir un cierto nivel de recursos informáticos gratuitos, controlado por un conjunto de límites. Los desarrolladores que necesitan recursos por encima de estos límites gratuitos pueden cambiar a una aplicación de pago a través de *Google Checkout* para establecer un presupuesto diario de recursos. Cuando se pasa a una aplicación de pago se gastara un mínimo de $2.10/por semana. Esto le permite adquirir recursos adicionales cuando sea necesario. *App Engine* siempre será libre para empezar, y después de haber creado una aplicación de pago, todo el uso de los límites libres permanecerá libre.

A continuación se presentan los límites fuera de los cuales la aplicación deja de ser gratuita, solo para los parámetros más importantes.

CANAL.

Llamadas al canal de la API: El número total de veces que la aplicación accede al servicio del Canal.

Creación de Canales: El número de canales creados por la aplicación.

Horas de canal solicitadas: El número de horas de canal de tiempo de conexión solicitada por la aplicación.

Los datos enviados Canal: La cantidad de datos enviados a través del servicio de canal. Esto también cuenta para la cuota de ancho de banda saliente. La tabla A1.1 resume los límites por uso de canal de *Google Apps*.

Recurso	Limite Gratuito Por defecto		Limite de Facturacion por Defecto	
	Limite Diario	Taza Maxima	Limite Diario	Taza Maxima
Llamadas al canal de la API	657,000 calls	3,000 calls/minute	91,995,495 calls	32,000 calls/minute
Creacion de Canales	100 channels	6 creations/minute	Según Presupuesto	60 creations/minute
Horas de Canal Solicitadas	200 hours	12 hours requested/minute	Según Presupuesto	120 hours requested/minute
Datos enviados por Canal	Arriba de la cuota de ancho de banda de salida	22 MB/minute	2 GB	740 MB/minute

Tabla.A1.1. Límites de Facturación por el uso de los canales en GAE.

CÓDIGO Y DATOS ESTÁTICOS.

Se refiere al espacio de almacenamiento utilizado por todo el código del programa y los datos estáticos. Esto combina espacio utilizado por todas las versiones. Por ejemplo, si usted mantiene dos versiones de la aplicación, cada uno con 100 Mb, se trata de 200Mb. El tamaño total de código y almacena los archivos estáticos se enumeran en la *DashBoard* de la aplicación. Aplicaciones libres sólo podrá subir hasta 1 GB de código y datos estáticos. Las aplicaciones de pago pueden subir más, pero se le cobrará $ 0.13 por GB al mes por cualquier código y almacenamiento de datos estática que sea superior a 1 GB.

BASE DE DATOS (DATASTORE).

Se refiere a los datos almacenados por la aplicación en el almacén de datos, la cola de tareas y en la *Blobstore*. La tabla A1.2 resume los límites de facturación y gratuidad por el uso del *Data Store* de *App Engine*.

Número de índices: El número de índices de almacén de datos que existen para la aplicación. Esto incluye los índices que se han creado en el pasado y ya no aparecen en la configuración de la aplicación pero no se han borrado con el comando AppCfg de *vacuum_indexes*.

Las operaciones de escritura: El número total de operaciones de escritura almacén de datos.

Operaciones de lectura: El número total de almacén de datos de las operaciones de lectura.

Operaciones pequeñas: El número total de *datastore* pequeñas operaciones.

Resource	Limites Diarios Por Defecto	Limites de Facturacion por Defecto
Capacidad de Alamcenamiento de Datos	1 GB **Nota: Limite total. Sin limite Diario**	1 GB gratis, sin Limite
Numero de Indices	200 **Nota:Limite Total. Sin Limite Diario**	200
Operaciones de Escritura	50,000	Ilimitada
Operaciones de Lectura	50,000	Ilimitada
Pequeñas Operaciones	50,000	Ilimitada

Tabla.A1.2. Límites de Facturación por el uso de del data store en GAE.

SOLICITUDES.

Ancho de banda de salida: La cantidad de datos enviados por la aplicación en respuesta a las solicitudes. La tabla A1.3 resume los límites de la cantidad de solicitudes para una aplicación gratuita y una con facturación.

Esto incluye:

- Datos entregados en respuesta a las dos peticiones seguras y solicitudes no seguras de los servidores de la aplicación, servidores de archivos, estáticos o el *Blobstore*.
- Datos enviados en mensajes de correo electrónico.
- Datos enviados a través de XMPP o por el canal de la API.
- Datos salientes peticiones HTTP enviados por el servicio de obtención de URL.

Ancho de banda entrante: La cantidad de datos recibidos por la aplicación de peticiones. Cada solicitud HTTP entrante no puede ser mayor de 32 MB.

Esto incluye:

- Datos recibidos por la aplicación en las solicitudes de seguras y solicitudes no seguras.
- Las subidas a la *Blobstore*.
- Los datos recibidos en respuesta a las peticiones HTTP por el servicio de obtención de URL.

Resource	Limite Gratuito por Defecto		Limite de Facturacion por Defecto	
	Limite Diario	Taza Maxima	Limite Diario	Taza Maxima
Ancho de Banda de Salida(incluye HTTPS)	1 GB	56 MB/minute	1 GB free; 14,400 GB maximo	10 GB/minute
Ancho de Banda de Entrada (incluye HTTPS)	1 GB; 14,400 GB maximo	56 MB/minute	Ninguno	Ninguno

Tabla.A1.3. Límites de Facturación por el uso del Ancho de Banda en GAE.

PRECIOS.

La Tabla A1.4 resume los precios de las características más importantes a cancelar una vez se supera la gratuidad de la aplicación.

Recurso	Unidad	Costo por Unidad
Ancho de Banda de Salida	Gigabytes	$0.12
Almacenamiento de Datos (Blobstore)	Gigabytes per month	$0.13
Almacenamiento de Datos (Datastore)	Gigabytes per month	$0.24
Canal	Canal Abierto	$0.0001 ($0.01/100 channels)
SSL Virtual IPs (VIPs)	Virtual IP per month	$39.00

Tabla.A1.4. Listado de Precios por unidad que cobra GAE al exceder los límites gratuitos.

La tabla A1.5 resume el costo de las operaciones con base de datos una vez se supera la gratuidad de la aplicación.

Operación	Costo
Escritura	$0.10 por 100k de operación
Lectura	$0.07 por 100k de operación
Pequeña Operación	$0.01 por 100k operación

Tabla.A1.5. Listado de Costos por las operaciones en el Data Store de GAE al exceder los límites gratuitos.

Para una información más detallada sobre los precios se puede visitar la página de Facturación oficial de Google [7], así como también la página sobre las cuotas diarias gratuitas y facturables [8].

A1.3. PROGRAMACIÓN DE UNA APLICACIÓN WEB BASADO EN GAE.

A1.3.1. ESTRUCTURA BÁSICA PYTHON.

Las aplicaciones GAE tendrán la estructura básica mostrada en la Figura A1.9

```
import webapp2

def escape_html(s):

      codigo python

def rot_trece(h):

      codigo python

form =”””

class MainPage(webapp2.RequestHandler):

      codigo python

app = webapp2.WSGIApplication([('/',MainPage)],debug = True)
```

Figura.A1. 9. Estructura básica del código Python para una aplicación GAE.

Se comienza con la importación de librerías, se deben de incluir todas las librerías necesarias para el correcto funcionamiento del programa, si se utilizan plantillas deben de incluirse las respectivas librerías, se si hace uso de los comandos de tiempo en Python, por ejemplo, también debe incluirse las librerías para manejar tiempo de Python, la mayoría de librerías de Python están disponibles.

Luego se define la clase principal que contendrá el código de la aplicación, esta debe llevar como parámetro un objeto `webbapp2.RequesHandler` incluido en la librería que se importó el cual es un manejador genérico para Google. Dentro de la clase se definen las funciones necesarias para que la aplicación haga su trabajo, estas pueden ser varias de acuerdo a las necesidades que la aplicación vaya a realizar, pero siempre debe de incluir por lo menos la función get(self).

La última línea siempre se debe la mostrada en la Figura A1.10.

```
app = webapp2.WSGIApplication([('/', MainPage)], debug=True)
```

Figura.A1. 10. Línea de mapeo de URL para terminar una aplicación GAE.

Llamada línea de mapeo de URL. En el ejemplo mostrado en la Figura A1.10, esta solo tiene una URL '/', la cual nos dirige la a aplicación al manejador *MainPage*.

Podemos incluir dentro del código lenguaje HTML utilizando triple comillas como se muestra en la Figura A1.11.

```
import webapp2

def escape_html(s):

      codigo python

def rot_trece(h):

      codigo python

form = """

<form method="post">

      ENTER some text to ROT13:

      <br>

4     <br>

      <textarea cols="50" rows="5" name="text">%(text)s</textarea>

      <br>

      <input type="submit">

</form>

"""

class MainPage(webapp2.RequestHandler):

      codigo python

app = webapp2.WSGIApplication([('/',MainPage)],debug = True)
```

Figura.A1. 11. Estructura de una aplicación GAE con código HTML incrustado.

La variable *form* en este caso dirige a un código en HTML que incluye un *input* y un formulario. Pero lo más recomendable es utilizar plantillas para esto. Ya que si no el código para una aplicación con múltiples páginas web, será demasiado grande y el uso de plantillas simplifica las cosas.

A1.3.2. PLANTILLAS.

Para mantener separado el código *Python* y el código HTML de las páginas web, GAE hace uso de plantillas, las plantillas son librerías que poseen rutinas que direccionan el código a otros scripts que contienen el código HTML de las páginas web, esto es muy útil cuando se trabaja con múltiples páginas web. *AppEngine* incluye un motor de plantillas que pueden ayudarle a hacer precisamente eso Jinja2.

Jinja2 es un potente motor de plantillas basado en el sistema de plantillas de Django. La idea es separar la lógica de presentación, y obtener un código de proceso limpio y bien definido. El proceso básico para que funcione en *AppEngine* es el siguiente.

Paso 1: Modificar 'app.yaml'

Se abre app.yaml desde el directorio del proyecto *AppEngine* en el editor de su elección, para modificar un par de líneas:

1) Se cambia el atributo de tiempo de ejecución para python27.

2) Añadir *multi-hilo: true*, y

3) Añadir la biblioteca Jinja2 en la configuración. Esto es lo que tu app.yaml podría parecer después de que haya terminado:

Paso 2: Crear un directorio en la carpeta del proyecto denominado plantillas. Dentro de ese directorio se colocan todos los archivos HTML a utilizar. La estructura de estos archivos HTML es la clásica para este lenguaje. Los nombres de estos archivos deben de recordarse y ser consistentes con su utilización ya que se hará referencia a ellos en el código de proceso Python.

Paso 3: Hay que modificar el script Python para usar Jinja2, hay que eliminar todo el código HTML de este script principal que solo contendrá código Python. Además se debe importar el módulo Jinja2, y crear un "ambiente" por medio de la cual podemos acceder a las funciones de plantillas. Esto es lo que la parte superior de la secuencia de comandos podría ser como se muestra en la Figura A1.12.

```
import os

import webapp2

import jinja2

jinja_environment = jinja2.Environment(autoescape=True,

loader=jinja2.FileSystemLoader(os.path.join(os.path.dirname(__file__), '

plantillas')))
```

Figura.A1. 12. Secuencia de comandos para importar Jinja2.

Hay un par de cosas a tener en cuenta aquí: el objeto `jinja2.Environment` que estamos creando toma un argumento *`Loader`* que es un directorio desde el que se puede cargar plantillas, por lo que hay que importar el módulo os para poder usar las funciones de manipulación de rutas para obtener la ruta a el directorio de plantillas. Note también que hemos utilizado el argumento *`Autoescape = True`* - esto hace Jinja2 salte todas las cadenas maliciosas automáticamente, es decir cadenas con código HTML que modifiquen el código de alguna forma.

Para usar una plantilla en uno de los manipuladores de nuestra aplicación, se usa algo como se muestra en la Figura A1.13:

```
class MainPage(webapp2.RequestHandler):

    def get(self):

        template_values = {

            'name': 'SomeGuy',

            'verb': 'extremely enjoy'

        }

        template = jinja_environment.get_template('index.html')

        self.response.out.write(template.render(template_values))
```

Figura.A1. 13. Estructura de una aplicación GAE utilizando plantillas con Jinja2.

Primero creamos un diccionario con los pares `name / verb` para la plantilla, que son los mismos nombres de los campos incluidos en la plantilla. Luego se recupera un objeto de plantilla mediante `jinja_environment.get_template`, y finalmente se hace y escribe el código HTML resultante usando `template.render (template_values)`. Podemos incluir también código Python dentro del código HTML por medio de Jinja2. Para hacer esto debemos hacer dentro de las plantillas HTML: `{% código python%}`

Las variables se envían desde las plantillas HTML o que se reciben desde el código Python van encerradas con doble llave: `{{una_variable}}`

Hay que tener en cuenta que hay que instalar este módulo antes para ello una forma es como se muestra en la Figura A1.14.

```
$sudo apt-get install python-setuptools

$easy_install Jinja2
```

Figura.A1. 14. Comando para instalar el módulo Jinja2 de GAE desde consola de Linux.

En vista a usar plantillas se debe añadir un fichero que las contenga y especificarlo en el código este debe estar en el mismo directorio del proyecto si no dará error.

A1.3.3. EL DATASTORE.

Cuando usamos *Google App Engine*, no tenemos acceso a una base de datos relacional tradicional como MySQL, Oracle o Postgres. Nuestros datos se almacenan en el *Google Datastore* que usa un enfoque jerárquico orientado a objetos al estar basado en otra tecnología de Google, el Google *Bigtable* que es un sistema distribuido de almacenamiento de datos estructurados.

La forma de describir un Modelo en el *Datastore* se muestra en la Figura A1.15. Primero se inicia importando la librería db del *google apps engine*, para seguir con la creación de una clase de contendrá las entidades del modelo.

```
from google.appengine.ext import db

class Person(db.Model):

    name = db.StringProperty()

    password = db.StringProperty()
```

Figura.A1. 15. Construcción de un modelo en el DataStore de GAE.

La introducción de un nuevo registro en el *Datastore* se realiza como se muestra en la Figura A1.15. Como se aprecia en la Figura se utiliza el método put del modelo.

```
newperson = Person(name="Perico",
password="62f3b966a2ba0903d5b9da1440eef6c7")

newperson.put()
```

Figura.A1. 16. Introducción de un nuevo registró en el DataStrore de GAE.

Consultar al modelo de datos es realmente sencillo, la Figura A1.17 muestra diferentes solicitudes de consulta al DataStore.

```
query = db.Qurery(Person)

query = query.filter('name =', name)

query = query.filter('password = ', password)

results = que.fetch(limit=1)
```

Figura.A1. 17. Consultas al DataStore de GAE.

Al igual que con las bases de datos relacionales, *Google Datastore* nos provee de una interfaz de consulta a través de una cadena GQL, como se muestra en la Figura A1.18.

```
query = db.GqlQuery("SELECT * FROM Person WHERE name = :name "
                    "AND password = :pwd ",
                    name="Perico",
                    password="62f3b966a2ba0903d5b9da1440eef6c7")
```

Figura.A1.18. Consulta con cadena GQL en GAE.

Hay que recordar que GQL no puede ejecutar JOINS en las consultas SELECT.

Se tiene acceso al *DataStore* desde el *Dashboard* de la aplicación de donde podemos hacer manipulaciones a las bases de datos desde el entorno grafico de la aplicación. La figura A1.19 muestra una captura de cómo se ven las diferentes entidades desde la consola de administración del DataStore.

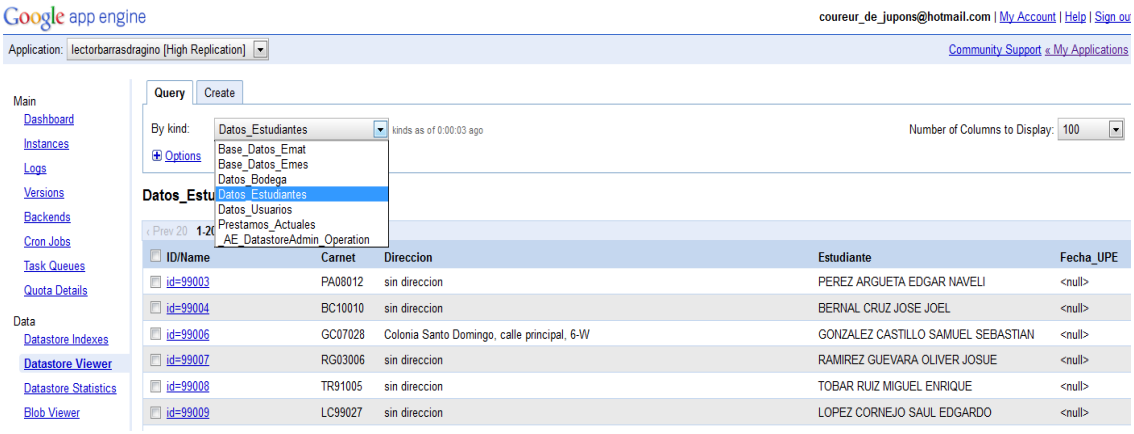


Figura.A1. 19. Apariencia del DataStore Viewver de GAE.

A1.3.4. MEMCACHE.

El uso de la memoria caché consiste en acelerar las consultas comunes del almacén de datos. Si varias solicitudes realizan la misma consulta con los mismos parámetros y no es necesario que los cambios de los resultados aparezcan en el sitio web inmediatamente, la aplicación puede almacenar los resultados en la caché del servicio *Memcache*. Las solicitudes sucesivas pueden comprobar la memoria caché y realizar solo la consulta del almacén de datos si los resultados no existen o han caducado. Los datos de sesión, las preferencias del usuario y cualquier otra consulta que se realice en la mayoría de las páginas de un sitio son buenos candidatos para almacenar en caché.

Memcache puede resultar útil para otros valores temporales. No obstante, si estás pensando si almacenar un valor únicamente en *Memcache* y no has considerado otro almacenamiento permanente, hay que asegurarse que la aplicación funciona de forma aceptable cuando el valor no esté disponible de repente. Los valores pueden caducar en *Memcache* en cualquier momento y pueden caducar antes del límite de tiempo de vencimiento establecido para el valor. Por ejemplo, si la repentina ausencia de los datos de sesión de un usuario causa fallos en el funcionamiento de la sesión, los datos deberían almacenarse probablemente en el almacén de datos además de hacerlo en el servicio *Memcache*.

De forma predeterminada, los valores almacenados en el servicio *Memcache* se conservan el mayor tiempo posible. Los valores se pueden expulsar de la memoria caché cuando se añade a esta un valor nuevo y la memoria caché tiene poco espacio. Cuando los valores se expulsan por presión de la memoria, los valores que menos se han utilizado recientemente son los primeros en salir. La aplicación puede proporcionar una hora de vencimiento cuando se almacena un valor, como un número de segundos relacionado con el momento en que se ha añadido el valor o como una hora *Epoch Unix* absoluta en el futuro (un número de segundos desde la medianoche del 1 de enero de 1970). El valor se expulsará en este preciso momento, aunque se pueda expulsar por otras razones. En circunstancias poco frecuentes, los valores también pueden desaparecer de la memoria caché antes del vencimiento por razones distintas a la presión de la memoria. Cuando el servicio *Memcache* es resistente a los fallos del servidor, los valores de *Memcache* no se guardan en disco, de modo que un fallo del servicio puede dar lugar a que los valores no estén disponibles.

En general, una aplicación no debería esperar que un valor almacenado en caché esté siempre disponible.

El tamaño máximo de un valor almacenado en la memoria cache del *App Engine* es de un megabyte. El uso de la memoria cache es muy sencillo, y un ejemplo se muestra en la Figura A1.20.

```
from google.appengine.api import memcache

# Añadir un valor si no existe en la cache con una caducidad de una hora.

memcache.add(key="talla_usuario_11987", value="XXL", time=3600)
```

Figura.A1. 20. Uso del Memcache en GAE.

ANEXO B

B.1 LECTORES DE BARRA.

Un Lector de barra es aquel que por medio de un láser lee un código de barras y emite el número que muestra el código de barras, no la imagen.
Existen diferentes tipos de escáner o lectores entre algunos tipos están:

- Lectores tipo pluma o lápiz.
- Lectores de ranura o slot.
- Lectores laser tipo pistola.
- Lectores tipo rastrillo o ccd.
- Lectores ccd de proximidad.
- lectores laser fijos omnidireccionales.
- lectores autónomos.
- lectores de códigos de barras de 2d.

B.1.1 LECTORES TIPO PLUMA O LÁPIZ.

Son básicamente lectores tipo pluma montados en una caja. La lectura se realiza al deslizar una tarjeta o documento con el código de barras impreso cerca de uno de sus extremos por la ranura del lector. La probabilidad de leer el código en la primera oportunidad es más grande con este tipo de unidades que las de tipo pluma, pero el código debe estar alineado apropiadamente y colocado cerca del borde de la tarjeta o documento. Este se muestra en la Figura B2.1.

Figura B2.1 muestra la forma del lector tipo lápiz.

B.1.2 LECTORES DE RANURA O SLOT.

Son lectores de contacto que emplean un foto detector CCD (Dispositivo de Carga Acoplada) formado por una fila de LEDs que emite múltiples fuentes de luz y forma un dispositivo similar al encontrado en las cámaras de video. Se requiere hacer contacto físico con el código, pero a diferencia del tipo plumo no hay movimiento que degrade la imagen al escanearla. Este se muestra en la FiguraB2.2.

Figura B2.2 Muestra el lector de ranura.

B.1.3 LECTORES TIPO RASTRILLO O CCD.

El escaneo es completamente electrónico, como si se tomase una fotografía al código. No se requiere hacer contacto físico con el código pero debe hacerse a corta distancia. Tiene problemas de lectura en superficies curvas o irregulares. Este se muestra en la Figura B2.3.

Figura B2.3 Muestra el lector de rastrillo o CCD.

B.1.4 LECTORES CCD DE PROXIMIDAD.

Requieren poca distancia del lector al objeto pero tienen mejor performance que los CCD debido a su potente luz láser. Mejores resultados en superficies curvas o irregulares.

B.1.4.1 LECTORES LASER DE PROXIMIDAD.

Usan un mecanismo activador el escáner para prevenir la lectura accidental de otros códigos dentro de su distancia de trabajo. Un espejo rotatorio u oscilatorio dentro del equipo mueve el haz de un lado a otro a través del código de barras, de modo que no se requiere movimiento por parte del operador, éste solo debe apuntar y disparar.

Por lo general pueden leer códigos estropeados o mal impresos, en superficies irregulares o de difícil acceso, como el interior de una caja. Más resistentes y aptos para ambientes más hostiles.

El lector puede estar alejado de 2 a 20 cm del código, pero existen algunos lectores especiales que pueden leer a una distancia de hasta 30 cm, 1,5 metros y hasta 5 metros.

B.1.4.1.2 LECTORES LASER TIPO PISTOLA.

Son básicamente lo mismo que el tipo anterior, pero montados en una base. La ventana de lectura se coloca frente al código a leer (generalmente se orientan hacia abajo) y la lectura se dispara al pasar el artículo que contiene el código frente al lector y activarse un censor especial. Esta configuración de lector se encuentra frecuentemente en bibliotecas ya que libera las manos del operador para que pueda pasar el libro frente al lector. También se utiliza en sistemas automáticos de fábricas y almacenes, donde el lector se coloca sobre una banda transportadora y lee el código de los artículos que pasan frente a él. Este se muestra en la Figura B2.4

Figura B2.4 Muestra el lector tipo pistola.

A.1.4.1.3 LECTORES LASER FIJOS OMNIDIRECCIONALES.

Se encuentran normalmente en las cajas registradoras de supermercados. El haz de laser se hace pasar por un arreglo de espejos que generan un patrón omnidireccional, otorgando así la posibilidad de pasar el código en cualquier dirección. Los productos a leer se deben poder manipular y pasar a mano frente al lector. Recomendados cuando se requiere una alta tasa de lectura. Este se muestra en la Figura B2.5

Figura B2.5 Muestra el lector de Omnidireccionales.

B.1.4.1.4 LECTORES AUTÓNOMOS.

No requieren atención, se usan en aplicaciones automatizadas o de cinta transportadora. Varían en velocidad de lectura según la producción y la orientación requerida de los códigos de barras, línea única, multilínea y omnidireccional. Este se muestra en la Figura B2.6

Figura B2.6 Muestra el lector Autónomo.

B.1.5 LECTORES DE CÓDIGOS DE BARRAS DE 2D.

Leen códigos en dos dimensiones como PDF, DATAMATRIX y MAXICODE, ya mostrados en los tipos de código de barras existentes. Este se muestra en la Figura B2.7

La estructura básica de un código de barras consiste de zona de inicio y terminó en la que se incluye: un patrón de inicio, uno o más caracteres de datos, opcionalmente unos o dos caracteres de verificación y patrón de término. La información es leída por dispositivos ópticos los cuales envían la información a una computadora como si la información hubiese sido tecleada.

Figura B2.7 Muestra el lector EN 2D.

B.2 ESCANER MS9520.

El lector o escáner de barra utilizado en el este proyecto será el MS9520 Voyager[A1], ofrece una lectura de todos los códigos de barras 1D que están en el mercado[A2]. La serie de escáneres lineales de código de barras de mano Voyager tiene un formato unificado con funciones que los convierten en la referencia de la industria en cuanto a valor y rendimiento, por lo que en nuestro proyecto la utilización de este escáner MS9520 lo hace muy eficiente y de mucho ayuda.

Este escáner está equipado con la función de activación automática por infrarrojos y descodifica todos los códigos de barras 1D estándar. El escáner MS9520 además de ser efectivo tiene una forma de diseño muy buena lo cual lo hace especial, se puede ver la representación del escáner en la Figura B2.8.

Figura B2.8 Muestra el lector de barra MS9520 Voyager.

B.2.1 APLICACIONES DEL MS9520.

El escáner es de múltiples aplicaciones por lo que la forma de verlo es muy extensa en los siguientes ejemplos se darán algunas aplicaciones:

- ***PUNTO DE VENTA***: Es al parecer una de las más usadas por los escáner utilizadas en tiendas, comercios y farmacias, supermercados, etc. Donde se requieren leer los códigos de barras de los productos que se van a cobrar al cliente. Estos van ligados con inventarios, al leer los códigos de barras de los productos en el almacén.
- ***BIBLIOTECAS***: Para leer los códigos en los libros, carnet, entrada de personal, forma de préstamos, etc.
- ***CONTROL DE ACCESO:*** Los códigos de barras de gafetes y credenciales, carnet, documentos únicos de identidad, entradas a cine, conciertos, conferencias, etc.

B.2.2 INTERVALO DE ACTIVACIÓN IR.

El valor por defecto de láser ó modo de barrido del escáner se muestra en la Figura B2.9, donde se observa el barrido máximo que tiene el escáner, esto servirá de mucha ayuda dependiendo la aplicación del escáner, para el caso del proyecto se hará una aplicación en el espacio pequeño por lo que no necesita tanta movilidad de un lugar a otro.Cualquier movimiento detectado por el IR en el área de activación hará que el escáner al se encienden automáticamente el láser, la preparación del escáner para obtener el conocimiento de códigos de barras, decodificación y transmisión.

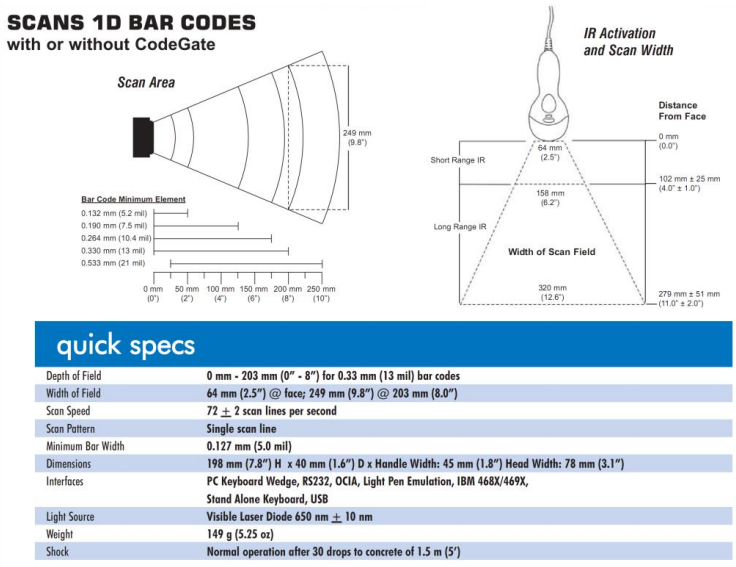


Figura B2.9 Muestra el lector de barra MS9520 Voyager.

B.2.3 ESPECIFICACIONES TÉCNICAS.

En la tabla B.2.1 se muestra las especificaciones del escáner a ocupar.

OPERATIVAS	
Fuente de luz	Diodo láser visible 650 nm ± 10 nm
Indicadores visuales	Verde = listo para escanear; Rojo = lectura correcta; Amarillo = escaneo automático
Interfaces del sistema principal	USB, RS232, teclado en cuña, IBM 46xx (RS485), OCIA, emulación láser de lápiz óptico, emulación (lector) de lápiz luminoso
MECANICAS	
Dimensiones (L x An x Al)	198 mm x 78 mm x 56 mm (7,8″ x 3,1″ x 2,2″)
Peso	149 g (5,3 oz)
ELECTRICAS	
Tensión de entrada	5 VCC + 0,25 V
Energía operativa (típica)	825 mW (165 mA a 5 V)
Energía de reserva (típica)	600 mW (120 mA a 5 V)
Transformadores de CC	Clase 2: 5,2 VCC a 1 A
Clase del láser	Clase 1: IEC60825-1, EN60825-1
Compatibilidad electromagnética	FCC Parte 15, ICES-003, EN55022 Clase B
MEDIOAMBIENTALES	
Temperatura de funcionamiento	0 °C a 40 °C (32 °F a 104 °F)
Temperatura de conservación	-40 °C a 60 °C (-40 °F a 140 °F)
Humedad	5% a 95% de humedad relativa, sin condensación
Caída	Diseñado para soportar caídas al suelo desde 1,5 m (5 pies)
Sellado medioambiental	Sellado para resistir contaminantes de partículas del aire
Niveles de luz	4842 lux (450 candelas/pie)
RENDIMIENTO DE ESCANEO	
Patrón de escaneo	Línea de escaneo simple
Velocidad de escaneo	72 líneas de escaneo por segundo

Tabla B2.1 muestra las características del lector de barras.

B.2.4 EMPLEO DEL LECTOR DE BARRAS MS9520.

El Voyager MS9520 es el lector laser más vendido en el mundo. Una de sus características más importantes es que el lector se puede utilizar de dos modos:

MODO FIJO.

Se pone en su base, y funciona de modo automático, es decir, al acercar cualquier objeto o un código de barras, gracias a su sensor Infrarrojo, el rayo láser se encenderá automáticamente, leerá el código de barras presentado y lo transmitirá inmediatamente a la computadora. Después de leer, el rayo láser permanece encendido esperando leer otro código de barras. Esta es una gran ventaja, ya que permite al usuario utilizar las dos manos sin tener que tocar o manipular el lector. Este se muestra en la Figura B2.10.

Figura B2.10 Muestra el lector de barra MS9520 Voyager.

MODO MANUAL.

En este modo, el usuario toma el lector de la base y lo acerca a cualquier código de barras, y automáticamente lo leerá y lo transmitirá a la computadora. No es necesario oprimir ningún botón o gatillo, haciendo la operación más sencilla para el usuario. Este se muestra en la Figura B2.11.

Figura B2.11 Muestra el lector de barra MS9520 Voyager.

B.3 ROUTER DRAGINO.

Figura B3.1 Muestra el Router Dragino.

La Figura B3.1 muestra el Dragino en forma física este permite incorporar Linux en su MCU proyecto. Es un costo, Linux motherboard hardware abierto bajo para micro controladores. Se ejecuta Linux, usando OpenWRT , y tiene plena capacidad Ethernet y WiFi 802.11b / g. El objetivo de la Dragino es resolver el problema de conectividad y mejorar en gran medida los productos de micro controladores como el Arduino. Dragino tiene una potente CPU 180MHz, flash de 8 MB, 32 MB de RAM, Ethernet, Wifi y se ejecuta firmware OpenWRT. Las aplicaciones incluyen control remoto de robots, registro de datos, aplicaciones web para la presentación de datos, redes de malla a través de WiFi y muchos más. La primera placa es Dragino llamada Dragino MS12.

El Dragino cuenta con las siguientes partes como lo muestra la Figura B3.2:

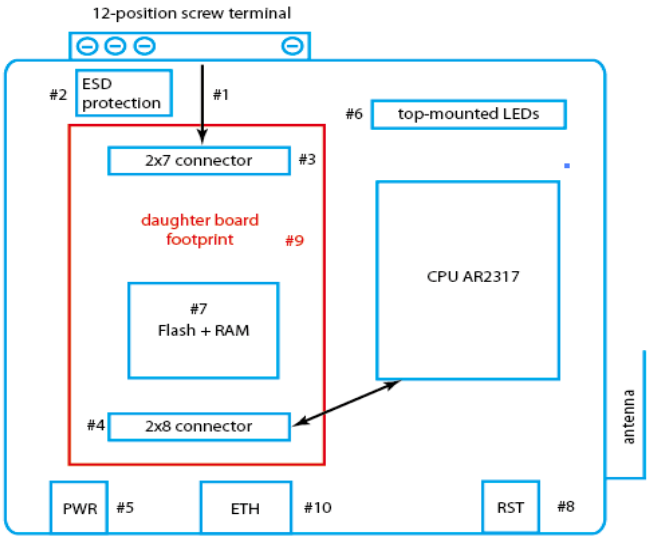


Figura B3.2 Muestra la forma en bloques sobre el Dragino.

A continuación se muestran las partes del bloque del Dragino:

1: Puente entre el terminal de tornillo de 12 posiciones y un conector 2x7.
2: los componentes de protección ESD .
3: 2 x 7 conectores.
4: GPIO (SPI), UART, LED externo, COLD_RST_L, 3.3v/5v/raw.
5: Puerto de alimentación.
6: LEDs (WiFi, rotura del corazón, LAN, SYS, PWR).
7: 8M flash + 16M SDRAM.
8: RST.
9: La hija huella Consejo.
10: Puerto Ethernet.

El apartado #4 el cual contiene los GPIO y UART es el que conectaremos al Raspberry Pi para la comunicación serial. A continuación encontraremos algunos de las características con las que cuenta el Dragino mostrada en la Figura B2.14.

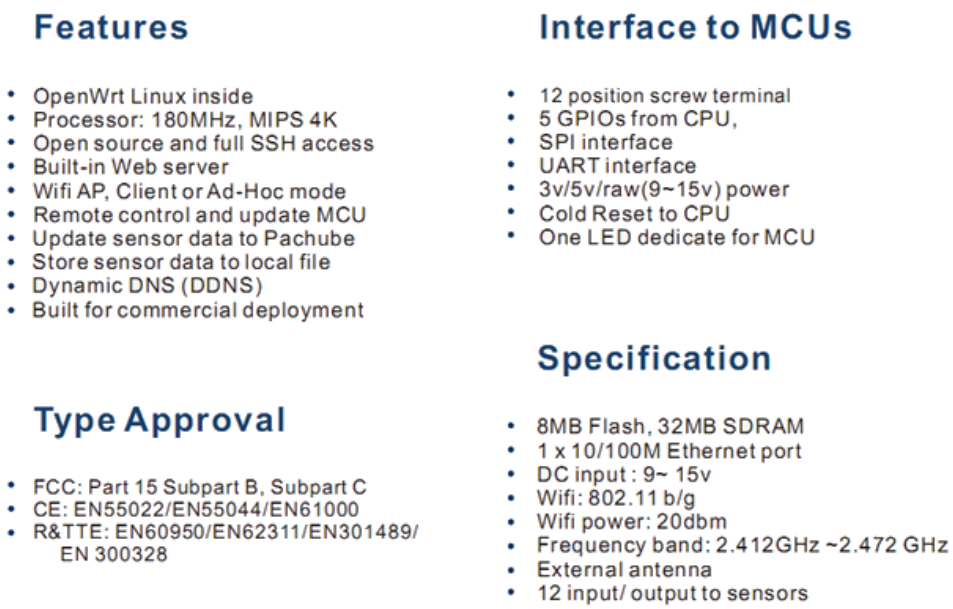


Figura B3.3 Muestra las características y especificaciones del Dragino.

B3.1 DESCARGAR SOFTWARE.

Para descargar la aplicación AP51-Flash vamos al siguiente sitio y lo descargamos:

http://svn.dragino.com/tools/

Figura B3.4. Sitio Web para descargar la aplicación ap51-flash.

Para obtener los archivos del firmware Dragino vamos al siguiente sitio web:

http://www.dragino.com/downloads/index.php?dir=MS12/firmware/

Figura B3.5 Sitio Web para descargar Firmware Dragino.

Y seleccionamos la última versión de firmware estable en este caso la 2.0:

http://www.dragino.com/downloads/index.php?dir=MS12/firmware/2.0-beta-1/

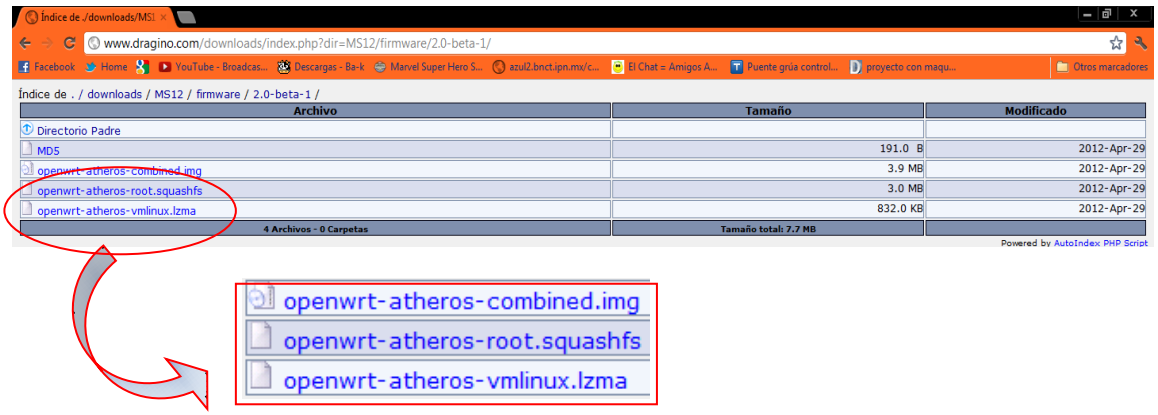


Figura B3.6. Sitio Web para descargar Firmware Dragino.

B.3.2 PREPARANDO EQUIPO PARA LA INSTALACION.

PASO 1: Mover los archivos descargados a la carpeta personal.

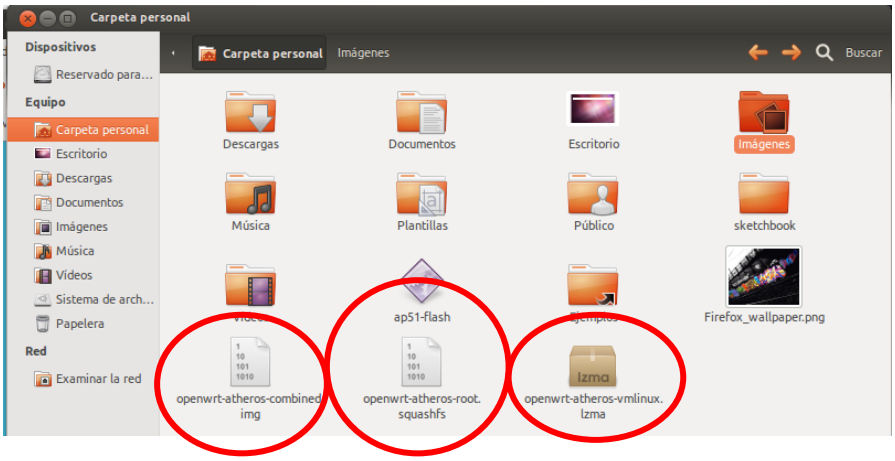


Figura B3.7. Archivos descargados en la carpeta personal.

PASO 2: Se debe de abrir la consola de comandos de Linux, e iniciar como súper usuario, a continuación se escribe lo que se muestra en la Figura B3.8.

```
sudo su
```

Figura B3.8. Archivos descargados en la carpeta personal.

PASO 3: Se le da el permiso de ejecución a la aplicación ap51-flash, como se muestra en la Figura B3.9:

```
chmod +x ap51-flash
```

Figura B3.9. Archivos descargados en la carpeta personal.

PASO 4: Luego de ello el paso siguiente es deshabilitar la conexión Wi-Fi de la computadora.

PASO 5: Conectar el cable RJ-45 al puerto ETHERNET del Dragino, como lo muestra la Figura B3.9

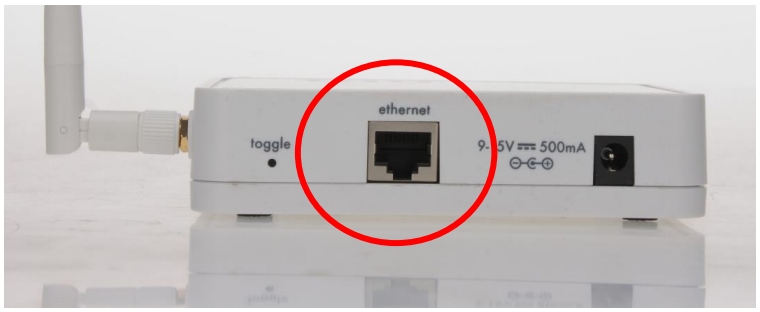


Figura B3.9. Puerto Ethernet de Dragino.

PASO 6: Conectar el Dragino al puerto Ethernet de la computadora sin conectarlo a la fuente de poder.

B.3.3 INICIANDO LA INSTALACION DEL FIRMWARE DRAGINO.

Se configura la IP a la tarjeta eth0 de la máquina, como lo muestra la Figura B3.10:

```
ifconfig eth0 192.168.255.2 up
```

Figura B3.10. Puerto Ethernet de Dragino.

La Figura B3.11 ejecuta el siguiente comando, todo eso se debe de hacer con privilegios de súper usuario:

```
./ap51-flash eth0 openwrt-atheros-root.squashfs openwrt-atheros-vmlinux.lzma
```

Figura B3.11. Puerto Ethernet de Dragino.

Linux buscará el Dragino en el Puerto Ethernet para hacer la instalación. Mostrará el siguiente mensaje ver Figura B3.12:

```
Device detection in progress. . . . . .
```

Figura B3.12. Puerto Ethernet de Dragino.

Seguido de ello se conectara el Dragino a su fuente de poder. AP51-flash detectara automáticamente el Dragino. Y comenzara a realizar la instalación, como se muestra en la Figura B3.13:

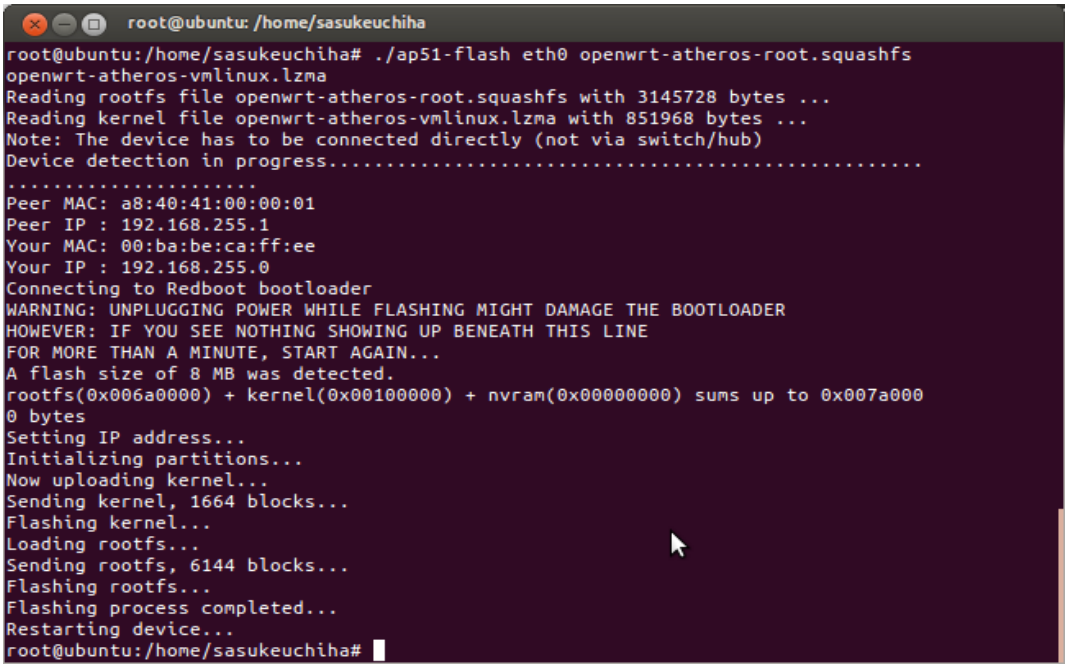


Figura B3.13. Captura del proceso de instalación del firmware Dragino.

Al terminar el proceso debemos esperar a que el LED de corazón comience a parpadear, esto puede tomar varios minutos.

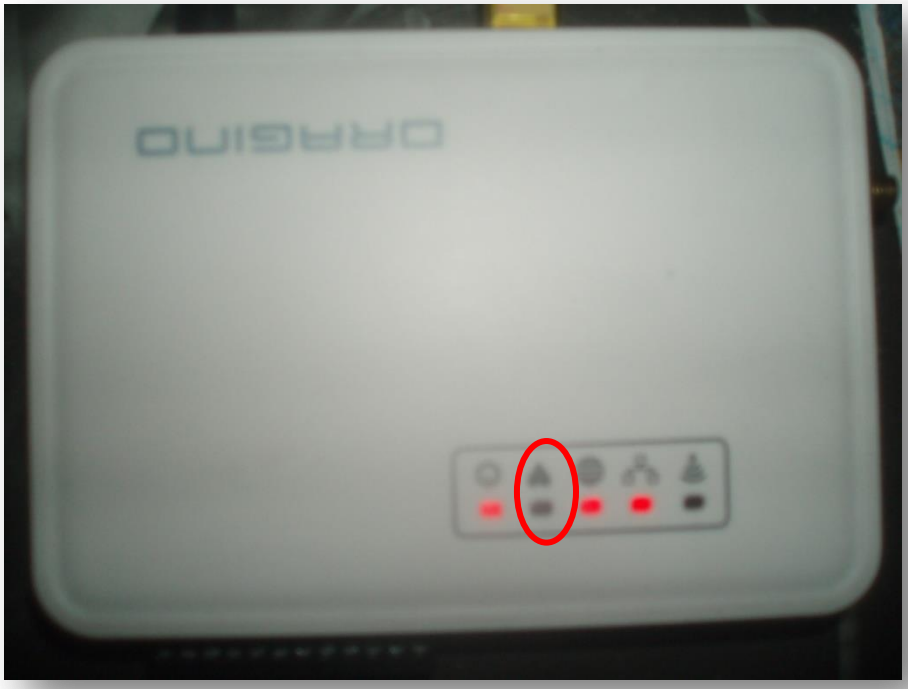


Figura B3.14. Ubicación del LED que indicara que el proceso se terminó con éxito.

La dirección IP proporcionada por Dragino es 19.168.255.1

B.3.4 ACCEDIENDO AL ROUTER CON FIRMWARE DRAGINO.

Al escribir en el navegador la dirección IP podremos ver la siguiente interfaz de configuración:

Figura B3.15. Interfaz Gráfica para configuración del router.

Para acceder al router se hace mediante el siguiente comando, como se muestra en la Figura B3.16, mediante la comunicación ssh[A2] :

```
ssh 192.168.255.1
```

Figura B3.16. Interfaz Gráfica para configuración del router.

La forma gráfica se muestra en la Figura B3.17 para la comunicación SSH visto desde el Dragino a la vez la dirección de root@dragino-562fff:~# muestra que ya está ejecutando OpenWRT y listo para dar instrucciones.

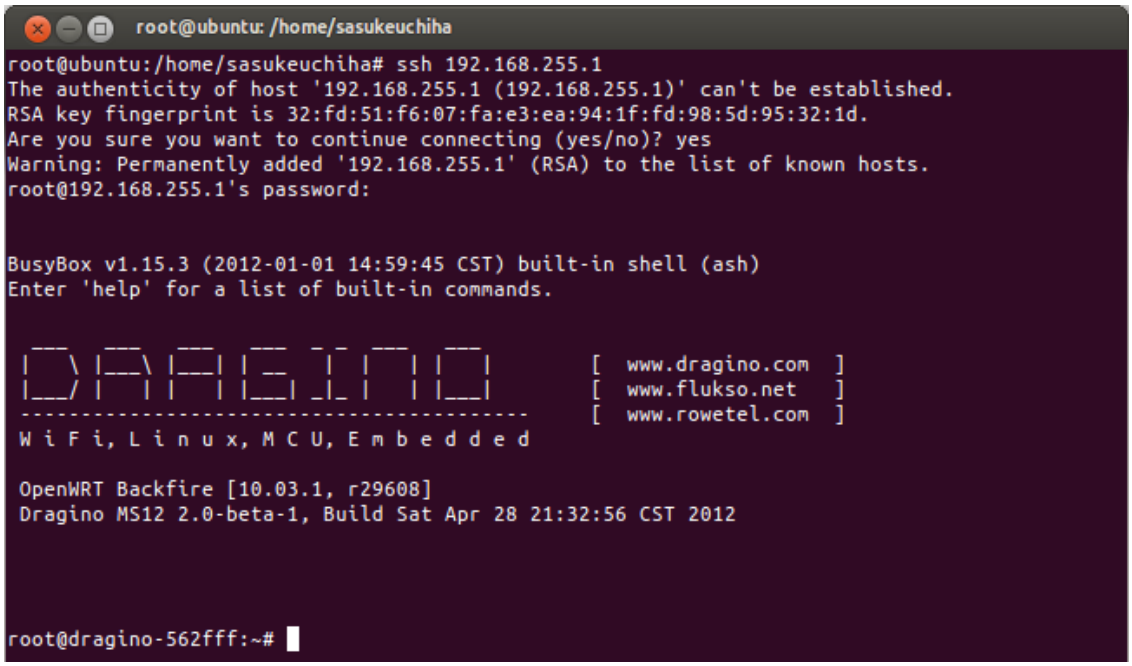


Figura B3.17. Interfaz Gráfica para configuración del router.

B.4 INTEGRADO MAX 232.

El MAX232 es un circuito integrado de Maxim que convierte las señales de un puerto serie RS-232 a señales compatibles con los niveles TTL de circuitos lógicos. El MAX232 sirve como interfaz de transmisión y recepción para las señales RX, TX, CTS y RTS.

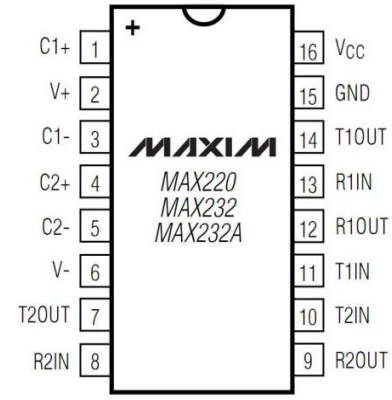
El circuito integrado tiene salidas para manejar niveles de voltaje del RS-232 (aprox. ± 7.5 V) que las produce a partir de un voltaje de alimentación de + 5 V utilizando multiplicadores de voltaje internamente en el MAX232 con la adición de condensadores externos. Esto es de mucha utilidad para la implementación de puertos serie RS-232 en dispositivos que tengan una alimentación simple de + 5 V.

Las entradas de recepción de RS-232 (las cuales pueden llegar a ± 25 V), se convierten al nivel estándar de 5 V de la lógica TTL. estos receptores tienen un umbral típico de 1.3 V, y una histéresis de 0.5 V.

La versión MAX232A es compatible con la original MAX232, y tiene la mejora de trabajar con mayores velocidades de transferencia de información (mayor tasa de baudios), lo que reduce el tamaño de los condensadores externos utilizados por el multiplicador de voltaje, – 0.1 µF en lugar del 1.0 µF usado para la comunicación con el Dragino, la Figura B4.1, mediante el carnet IEEE se puede introducir a la página del MAXIM y hacer pedidos gratis[A3]. Este circuito fue enviado de esta manera.

Figura B4.1 Muestra el integrado MAX 232.

El circuito que se ocupó para la conexión se extrajo del datasheet de la empresa MAXIM se muestra en la Figura B4.2, se hace una serie de arreglos de condensadores para tener los valores de voltajes que se quieren a la salida, para el caso se ocupó el MAXIM232A , lo cual se requirió de valores de capacitancia de 0.1uF.



CAPACITANCE (μF)					
DEVICE	C1	C2	C3	C4	C5
MAX220	0.047	0.33	0.33	0.33	0.33
MAX232	1.0	1.0	1.0	1.0	1.0
MAX232A	0.1	0.1	0.1	0.1	0.1

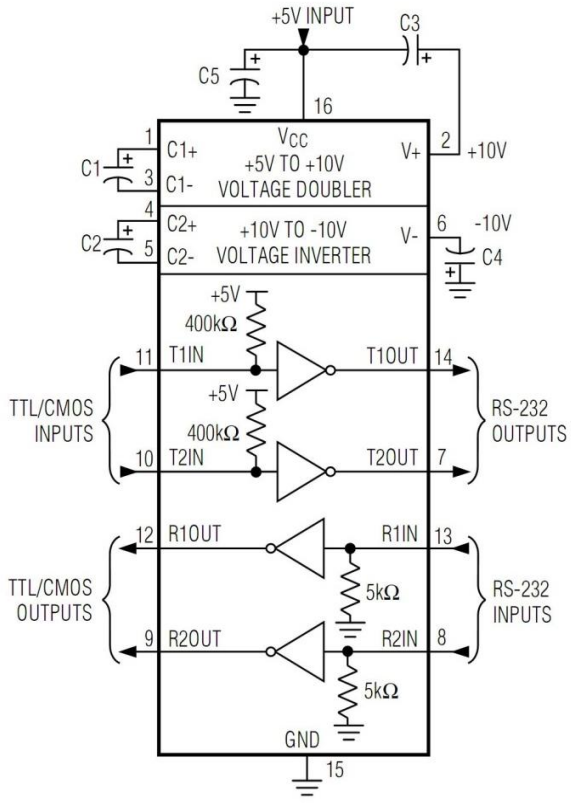


Figura B4.2 Muestra el circuito MAX 232.

B.5 RASPBERRY PI.

Figura B5.1 Muestra el Raspberry Pi modelo B.

El Raspberry Pi es un ordenador de tamaño de tarjeta de crédito que se conecta a su televisor y un teclado. Es un pequeño ordenador capaz que puede ser utilizado por muchas de las cosas que su PC de escritorio hace, como hojas de cálculo, procesadores de texto y juegos. También reproduce vídeo de alta definición, mostrado en la Figura B.5.1.

El diseño incluye un System-on-a-chip Broadcom BCM2835, que contiene un procesador central (CPU) ARM1176JZF-S a 700 MHz (el firmware incluye unos modos "Turbo" para que el usuario pueda hacerle overclock de hasta 1 GHz sin perder la garantía),9 un procesador gráfico (GPU)VideoCore IV, y 512 MB de memoria RAM aunque originalmente al ser lanzado eran 256 MB. El diseño no incluye un disco duro ó una unidad de estado sólido, ya que usa una tarjeta SD para el almacenamiento permanente; tampoco incluye fuente de alimentación o carcasa. El modelo B se vende a 35 $ y el modelo A a 25 $.

A continuación describimos todas las especificaciones técnicas del fabricante:

MODELO:
Raspberry Pi - Model B
Procesador (con gráfica integrada):
Broadcom BCM2835. Contiene ARM1176JZFS, con unidad de coma flotante, funciona a 700Mhz y un Videocore 4 GPU.
Memoria:
256MB
Características Técnicas de la GPU:
La GPU es capaz de mover contenidos con calidad Bluray, usando H.264 hasta 40MBits/s. Dispone un core 3D con soporte para las librerías OpenGL ES2.0 y OpenVG. Es capaz de decodificar 1080p30 H.264 high-profile.
Dispositivo de Arranque:
Memoria SD card. Tras el arranque inicial de la SD se puede terminar el arranque desde un dispositivo USB.
Conectores:
2x Conectores USB 2.0
Conector Ethernet RJ-45 10/100
Salida de Video Digital HDMI (Audio y Video)
Salida de Video Analógico (S-Video)
Audio Analógico (Conector 3,5mm)
Conector GPIO
Conector de alimentación Micro USB
Lector de memorias SD (Utilizado para arrancar el dispositivo)
Alimentación:
Vía Micro USB 5 Voltios. Casi cualquier dispositivo con alimentación USB puede servir como fuente de alimentación
Consumo Energetico: 700mA(3.5W)
Sistemas Operativos Soportados:
Raspbian “wheezy” (Debian)
ArchLinux
Fedora
QtonPi (QT SDK)
OpenElec
Raspbcm
Android (en desarrollo por usuarios)
Rangos de Temperatura:
LAN9512 de 0°C a 70°C
AP de -40°C a 85°C
Dimensiones:
85.60mm x 53.98mm x 17mm
Pack compuesto por:
1x Dispositivo Raspberry PI
1x Hoja indicando la normativa del dispositivo

Tabla B5.1 muestra las especificaciones Tecnicas del Raspberry Pi.

B.5.1 COMO INICIAR CON RASPBERRY PI .

Antes de encender por primera vez la Raspberry Pi se tiene que instalar una distribución del sistema operativo preparada para 16 bits y copiarla en la tarjeta SD, la cual es como el disco duro de la Raspberry Pi, en se guarda toda la información . En la Web oficial hay unas cuantas distribuciones de sistemas operativos listas para utilizar [A4]. Se puede tener varias tarjetas SD con una distribución por separado en cada SD, solo se tiene que apagar, cambiar de tarjeta y reiniciar las Raspberry Pi. En este trabajo de graduación se usara Debían por ser la más versátil.

B.5.1.1 INATALACION DE IMAGEN RASPIAN.

En la web oficial se muestran las distribuciones que existen, como se dijo se utilizara Debian con el nombre raspian "Wheezy".

PASO 1: Se descarga la imagen de Raspian "wheezy" dando click en Descarga directa como se muestra en la Figura B.5.2 [A5] .

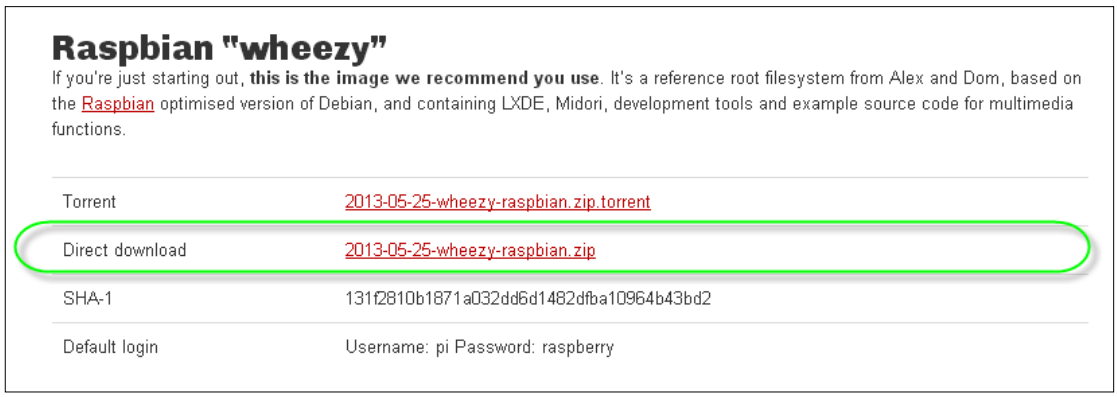


Figura B5.2 Muestra la dirección de imagen de Raspian para Debian.

PASO 2: En cualquier versión de Linux se podrá montar la imagen que se descargó anteriormente. Antes de ello hay que verificar si la clave hash del archivo zip es el mismo que aparece en la página de descargas. El archivo zip en el directorio de inicio (~ /) no debe de quedar de esta manera, y a continuación se debe de ejecutar en el terminal como lo muestra la Figura B.5.3:

```
sha1sum ~/ 2013-05-25-wheezy-raspbian.zip
```

Figura B5.3 Muestra como asignar el número hexagonal al archivo descargado.

Esto imprimirá un número hexagonal largo que debe coincidir con la línea "SHA-1" para la imagen de SD ha descargado.

PASO 3: Extraer la imagen y ver los dispositivos que están montados actualmente en el puerto USB con df-h, la respuesta se muestra en la Figura B5.4.

```
unzip ~/ 2013-05-25-wheezy-raspbian.zip
df -h
/dev/sdd1
```

Figura B5.4 Muestra como descomprimir y ver los dispositivos usb montados.

PASO 4: Ya teniendo el nombre donde está montado la SD, se tiene que desmontarlo para que los archivos no se pueden leer o escribir en la tarjeta SD mientras está copiando sobre la imagen, SD. El comando para desmontar se muestra en la Figura B4.5:

```
umount / dev/sdd1
dd bs=4M if=~/2013-05-25-wheezy-raspbian.img of=/dev/sdd
```

Figura B5.5 Muestra como desmontar y como escribir la imagen.

La segunda línea de la Figura B5.4 da la forma como escribir la imagen en la SD. La entrada da los parámetros de **dd** que se necesitan saber para el caso, **if =** argumento con la ruta de acceso al archivo img., y el "/ dev / sdd" en el archivo de salida **of =** discusión con la derecha nombre del dispositivo. Asegúrese de que el nombre del dispositivo es el nombre del conjunto de la tarjeta SD como se describe anteriormente [A6].

Luego de haber instalado la imagen se reinicia la Raspberry Pi y queda instalado Raspian en el Raspberry Pi y listo para usarse, antes de ello hay que configurar algunas cosas del Raspberry Pi cuando se inicia por primera vez Raspian se tendrán que realizar la configuración de los siguientes pasos:

Pedirá un usuario y una contraseña la cual se muestra en la Figura B5.6:

```
Usuario :pi
Contraeña: raspberry
```

Figura B5.6 Muestra el usuario y contraseña.

Este lo pedirá a la hora de encender el Raspberry Pi, luego de ello se actualizara los paquetes, y se instalara las actualizaciones. Esto lo muestra en la Figura B5.7:

```
sudo apt-get update
sudo apt-get upgrade
```

Figura B5.7 Muestra las actualizaciones de los programas y paquetes de Debian.

Para entrar al entorno grafico se escribirá por línea de comando lo que se muestra en la Figura B5.8.

```
startx
```

Figura B5.8 Muestra como pasar de consola a entorno gráfico.

Hay varias configuraciones más que se podrían instalar, esto dependiendo de la utilización del Raspberry Pi [A7].

B.6 DISEÑO AGREGADO AL HARDWARE.

En vistas que no existía una forma de cómo ver lo que estaba pasando entre la interfaz y el sitio Web, se diseñó una etapa para poder visualizar los pasos que la interfaz pide entre el Dragino y Raspberry Pi. A continuación mostraremos los pasos y los códigos que se utilizaron.

ETAPA DEL HARDWARE.

La mejor forma de visualizar las peticiones como agregar, borrar, carnet, materia y artículos. Se hizo necesario utilizar un LCD de 16*2, como el mostrado en la Figura B6.1.

Figura B6.1 Muestra el LCD de 16x2.

La pantalla LCD se conecta directamente con el Raspberry Pi por medio del circuito que se muestra en la Figura B6.2[B9]. Este circuito es muy barato de hacer ya que solo necesita de un potenciómetro, una resistencia y una tableta para circuitos, con un total de $ 15.00. El circuito creado por en Eagle 6.2.0 [B10]. El circuito Impreso para poder realizar la tableta impresa se muestra en la Figura B6.3a y Figura B6.3b.

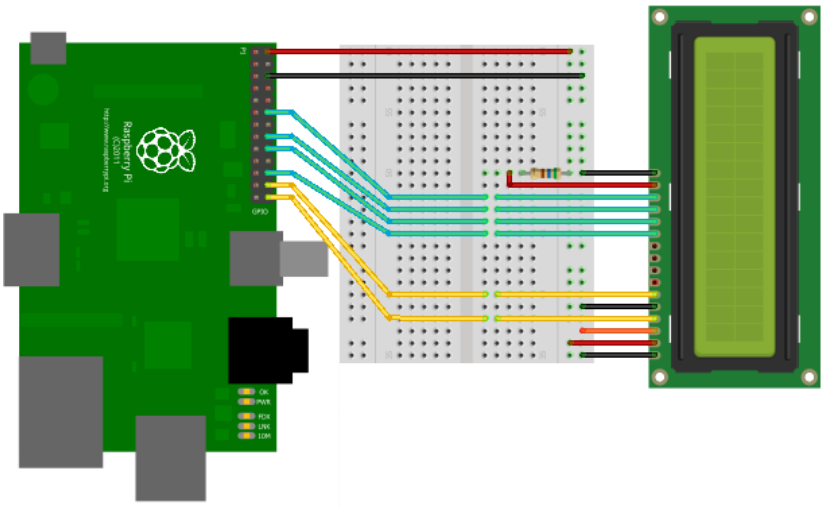


Figura B6.2 Muestra las conexiones que se tienen que hacer con el Raspberry pi.

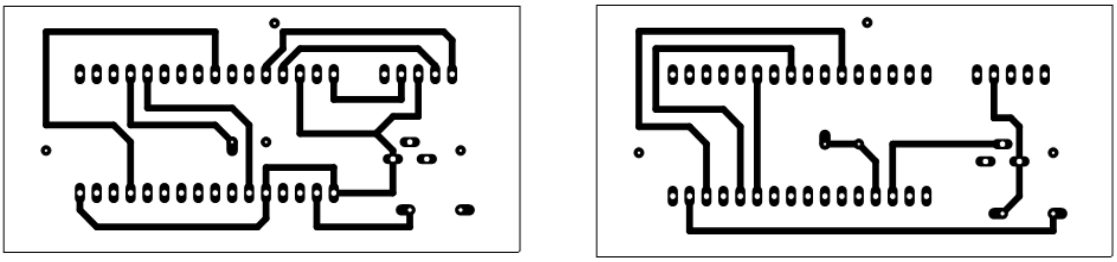


Figura 6B.3 Muestra las pistas para el circuito impreso de frente y atrás.

Ahora bien la forma del código cambio al agregar el LCD en la Figura B6.4 se muestran los cambios realizados [B9]. Los cambios se hicieron para las peticiones que el Dragino necesita, agregar, borra, carnet, materia y artículos. Por lo que estas se muestran en pantalla para visualizar de mejor forma mostrar los préstamos.

```
#!/usr/bin/python

# muestra las librerías a utilizar
import RPi.GPIO as GPIO
import time
import sys
import serial
ser = serial.Serial(port='/dev/ttyAMA0', baudrate=9600, timeout=1)

# Define los números de GPIO hacia LCD

LCD_RS = 7
LCD_E  = 8
LCD_D4 = 25
LCD_D5 = 24
LCD_D6 = 23
LCD_D7 = 18

# Define las constants para el LCD
LCD_WIDTH = 16    # maximos caracteres por linea
LCD_CHR = True
LCD_CMD = False

LCD_LINE_1 = 0x80 # LCD RAM address for the 1st line
LCD_LINE_2 = 0xC0 # LCD RAM address for the 2nd line

# constantes de tiempo.
E_PULSE = 0.00005
E_DELAY = 0.00005

def main():
  # Define el sistema de numeración para el caso es BCM
  # a la vez establece los valores de los puertos GPIO que se utilizaran.
  GPIO.setmode(GPIO.BCM)       # números de BCM GPIO
  GPIO.setup(LCD_E, GPIO.OUT)  # E
  GPIO.setup(LCD_RS, GPIO.OUT) # RS
  GPIO.setup(LCD_D4, GPIO.OUT) # DB4
  GPIO.setup(LCD_D5, GPIO.OUT) # DB5
  GPIO.setup(LCD_D6, GPIO.OUT) # DB6
  GPIO.setup(LCD_D7, GPIO.OUT) # DB7
  # Inicializa las salidas del LCD
  lcd_init()
  #Inicia el programa
  print "Bienvenido al programa de inventario de EIE"
```

```
  a="   Bienvenidos "
  b= " Inventario UES"
  bienve(a,b)
  time.sleep(1)
  print "Introduzca una opcion agregar, borrar o salir"
  c=" Introduzca  una "
  d= "     opcion   "
  bienve(c,d)
  time.sleep(1)
  c="     Agregar  "
  d= "     Borrar   "
  bienve(c,d)
  time.sleep(1)
  #lee si es agregar o borrar
  while True:
   linea = sys.stdin.readline()
   if linea[:-1]== "salir":
        break
   else:
         mando(linea)
          #leo()
# hasta aca el programa lee la opción de agregar, borrar o salir.
# lo siguiente es leer la variable que manda el Dragino.
def leo(a):
   result = ser.read(ser.inWaiting())
        c= result[a:]
   print "esto me manda dragino>>>>>>>>>>>>>>>>>>>>>>>>>>>>",c
   imprime(c)
   ser.flush()
# esta función sirve para mandar el dato leído por el lector de barras y queda en #
espera de una petición del Dragino.
def mando(param):
      ser.write(param)
      print "se le envio esto a Dragino>>>>>>>>>>>>>>>>>>>>>>>>>>>>",param
      a=len(param)
      time.sleep(1)
      # ahora lee lo que Dragino esta pidiendo.
      leo(a)
      ser.flush()

def bienve(a,b):
  lcd_byte(LCD_LINE_1, LCD_CMD)
  lcd_string(a)
  lcd_byte(LCD_LINE_2, LCD_CMD)
  lcd_string(b)
def imprime(a):

  lcd_byte(LCD_LINE_1, LCD_CMD)
  print "ok introduzca"
  lcd_string(" OK.. Introduzca ")
  lcd_byte(LCD_LINE_2, LCD_CMD)
  print "esta manda Dragino>",a
  lcd_string(a)

def lcd_init():
  # inicializar Display
  lcd_byte(0x33,LCD_CMD)
  lcd_byte(0x32,LCD_CMD)
  lcd_byte(0x28,LCD_CMD)
  lcd_byte(0x0C,LCD_CMD)
  lcd_byte(0x06,LCD_CMD)
  lcd_byte(0x01,LCD_CMD)

def lcd_string(message):
  # manda el mensaje hacia el display

  message = message.ljust(LCD_WIDTH," ")

  for i in range(LCD_WIDTH):
    lcd_byte(ord(message[i]),LCD_CHR)
```

```
def lcd_byte(bits, mode):


  GPIO.output(LCD_RS, mode) # RS
  GPIO.output(LCD_D4, False)
  GPIO.output(LCD_D5, False)
  GPIO.output(LCD_D6, False)
  GPIO.output(LCD_D7, False)
  if bits&0x10==0x10:
    GPIO.output(LCD_D4, True)
  if bits&0x20==0x20:
    GPIO.output(LCD_D5, True)
  if bits&0x40==0x40:
    GPIO.output(LCD_D6, True)
  if bits&0x80==0x80:
    GPIO.output(LCD_D7, True)

  # activar pines
  time.sleep(E_DELAY)
  GPIO.output(LCD_E, True)
  time.sleep(E_PULSE)
  GPIO.output(LCD_E, False)
  time.sleep(E_DELAY)

  # bits bajos
  GPIO.output(LCD_D4, False)
  GPIO.output(LCD_D5, False)
  GPIO.output(LCD_D6, False)
  GPIO.output(LCD_D7, False)
  if bits&0x01==0x01:
    GPIO.output(LCD_D4, True)
  if bits&0x02==0x02:
    GPIO.output(LCD_D5, True)
  if bits&0x04==0x04:
    GPIO.output(LCD_D6, True)
  if bits&0x08==0x08:
    GPIO.output(LCD_D7, True)

  # activar pines
  time.sleep(E_DELAY)
  GPIO.output(LCD_E, True)
  time.sleep(E_PULSE)
  GPIO.output(LCD_E, False)
  time.sleep(E_DELAY)

ser.close
if __name__ == '__main__':
  main()
```

Figura B6.4 Código Python encargado de controlar los dispositivos y la pantalla LCD.

El código anterior se inspiró en el código presentado por “spy” en el sitio http://www.raspberrypi-spy.co.uk [B9].

Al unir todas las partes Todo esto se ensamblo en una caja acrílica de 9.3cm x 16.00 cm. Para poder alojar todos los dispositivos. Como lo muestra en la Figura B6.5

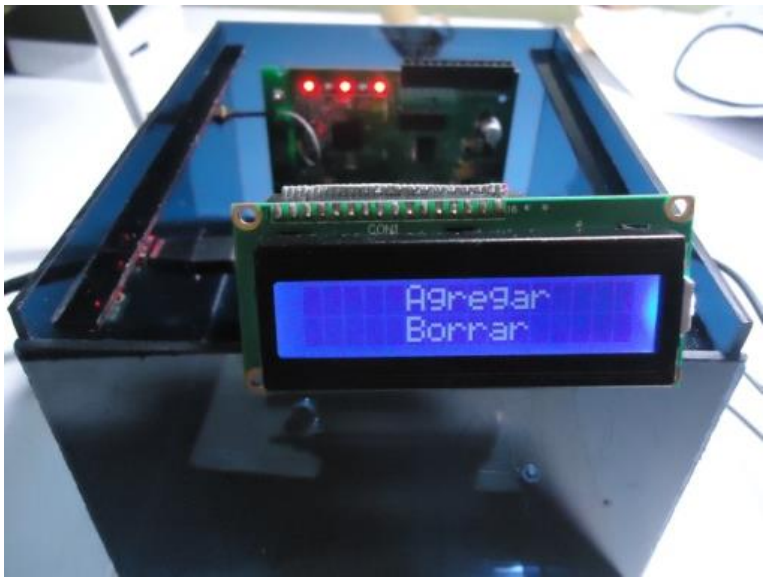


Figura B6.5 Muestra el trabajo realizado para el ensamblaje de todos los componentes.

BIBLIOGRAFIA.

[1] Información específica del Dragino (Revisado por última vez el 26/07/13).
http://www.dragino.com/about/about.html

[2] Guía de configuración para el escáner de la compañía Metrologic, (Revisado por última vez el 26/07/13).
http://downloads.visionid.ie/manuals/MS9535.pdf

[3] Información más amplia UART. (Revisado por última vez el 26/07/13).
http://www.zator.com/Hardware/H2_5_1_1.htm

[4] Video muestra la emulación del circuito con el MAX 232 y conectándolo al Router Dragino. (Revisado por última vez el 26/07/13.
http://www.youtube.com/watch?v=SvvSo-EBuxY

[5] Protocolo USB 2.0, (Revisado por última vez el 26/07/13).
http://www.usb.org/developers/docs/

[6] Ft232 información aplicaciones y mas, (Revisado por última vez el 26/07/13).
http://www.ftdichip.com/Products/ICs/FT232R.htm

[7]Circuitos de prueba con el FT232, (Revisado por última vez el 26/07/13).
http://www.clubse.com.ar/DIEGO/NOTAS/3notas/nota01-1.htm

[8] Información sobre Raspberry PI y puertos utilizados, (Revisado por última vez el 26/07/13).
http://www.raspberrypi.org/faqs (información)
http://deeiivid.files.wordpress.com/2013/02/raspi_html_m3e932f1c.png?w=820 (Puertos)

[9] Puerto serial en el Raspberry pi, (Revisado por última vez el 26/07/13).
http://raspberrypihobbyist.blogspot.com.es/2012/08/raspberry-pi-serial-port.html

[10] información sobre Python (Revisado por última vez el 26/07/13).
http://www.python.org/

[12]Permisos de ejecución para chwn. (Revisado por última vez el 26/07/13).
http://www.cyberciti.biz/faq/how-to-use-chmod-and-chown-command/

[13] Niveles de ejecución para procesos. (Revisado por última vez el 26/07/13).
http://manpages.ubuntu.com/manpages/hardy/es/man8/init.8.html

[14] Inicio de Google Apps Engine, (Revisado por última vez el 26/07/13).
https://cloud.google.com/products/?utm_source

[15]SDk para Google App Engine. (Revisado por última vez el 26/07/13).
https://developers.google.com/appengine/downloads?hl=es[1]

[16] Google Chart Tools . (Revisado por última vez el 26/07/13).
https://developers.google.com/chart/

[17] Instrumentos de una estación Meteorológica. (Revisado por última vez el 26/07/13). http://es.wikipedia.org/wiki/Estaci%C3%B3n_meteorol%C3%B3gica

ANEXO A

[1] Los atributos pueden no aparecer en los elementos HTML, o bien pueden aparecer varios de ellos.

[A2] Para mayor información sobre los servicios de *Amazon Web Services* visitar la página:

http://aws.amazon.com/es/

[A3] Para mayor información sobre los servicios de *Azure Services Platform* visitar la página:

http://www.windowsazure.com/en-us/

[A4] Para mayor información sobre GAE visitar la página:

https://developers.google.com/appengine/

[A5] Para mayor información sobre los servicios de Heroku visitar la página:

https://www.heroku.com/

[A6]Pagina de descarga del SDK

https://developers.google.com/appengine/downloads?hl=es

[A7] Pagina de facturación de GAE

https://developers.google.com/appengine/docs/billing?hl=es

[A8] Página sobre las cuotas gratuitas y facturables

https://developers.google.com/appengine/docs/quotas?hl=es

ANEXO B

[B1]información del escaner

http://www.metrologicmexico.com/productos1/lectores_manuales/ms9520_voyager.php

[B2]Tecnología de códigos de barra

http://www.metrologicmexico.com/contenido1/informacion_tecnica/codigos_de_barras_de_una_dimen.php

[B3] comunicación SSH

http://es.wikipedia.org/wiki/Secure_Shell

[B4]Integrados de MAXIM

http://www.maximintegrated.com/

[B5] Raspberry Pipara Iniciar un sistema operativo.

http://elinux.org/RPi_Easy_SD_Card_Setup

[B6] Sitio de descargas para el Raspberry Pi.

http://www.raspberrypi.org/downloads

[B7] Revista MAgPI muestra la configuración del Raspberry Pi

http://issuu.com/themagpi/docs/the_magpi_issue_2_final?e=4599523/2591447

[B8] Otras configuraciones para el Raspberry.

http://www.alexisabarca.com/2013/02/primeros-pasos-con-una-raspberry-pi/

[B9]Sitio del código e imagen utilizada para el diseño del circuito LCD.

http://www.raspberrypi-spy.co.uk/2012/07/16x2-lcd-module-control-using-python/

[B10] Eagle para crear circuitos impresos.

http://www.cadsoftusa.com/