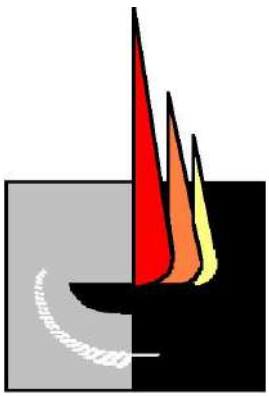
Universidad Nacional de La Pampa

Facultad de Ingeniería



Ingeniería en Sistemas

Práctica Profesional Supervisada

Proyecto:

"Diseño e implementación de sistema web para gestión y monitoreo de semáforos y cámaras IP"

Alumna:

SANCHEZ, María Belén

Tutor Académico:

Dr. BECKER, Pablo Javier

Tutor Institucional:

A.P. ALESSO, Leandro

General Pico, La Pampa, 2017.

Índice

Descripción general

El proyecto que se desarrolla a continuación corresponde a la Práctica Profesional Supervisada (PPS), según lo estipulado en la resolución N° 158/16 del Consejo Directivo de la Facultad de Ingeniería.

La Práctica Profesional Supervisada fue llevada a cabo en la empresa "Arbit Ingeniería", siendo actualmente, una de las empresas incubadas en INCUBATEC -Incubadora de Empresas de Base Tecnológica- situada en calle 21 n° 1354, primer piso, en la localidad de General Pico, La Pampa. La misma, se caracteriza por realizar actividades relacionadas al diseño y desarrollo de equipos tecnológicos específicos, automatización de procesos, control y monitoreo de dispositivos, telemetría, entre otras cosas.

El proyecto surge a través de una solicitud realizada por el municipio de la localidad de General Pico a la empresa 'Arbit Ingeniería' que consiste en desarrollar un sistema web capaz de gestionar y monitorear los semáforos y las cámaras IP que se encuentran bajo su control y supervisión. Los motivos causales del desarrollo del proyecto son, entre otras, las siguientes problemáticas abordadas por el municipio de la ciudad:

- Registro de semáforos y cámaras IP -con sus respectivas características- en listas realizadas en planillas de cálculo, almacenadas en una única PC sin conexión remota alguna, limitando su acceso y dificultando su rápida actualización.
- Acceso limitado a las cámaras IP. Para extraer la información contenida allí, es necesario ubicarse dentro del radio de cobertura de la cámara, es decir, a unos pocos metros de proximidad. Esto conlleva a que el personal deba trasladarse -junto con dispositivos de almacenamiento- hasta la ubicación de las mismas, consumiendo tiempo y gastos de transporte.
- Detección de fallos en semáforos sólo detectados visualmente -por transeúnte notificador de la causa o personal municipal en supervisión-. El tiempo transcurrido desde la generación de la falla y su detección, hasta la reparación y posterior puesta en funcionamiento, es considerable. Esta situación puede ser causal de un accidente de tránsito.

Introducción

En el desarrollo del informe, para cumplimentar el objetivo propuesto de desarrollar un sitio web para gestión y monitoreo de semáforos y cámaras IP, no solo se abordarán cuestiones vinculadas al software propiamente dicho, definiendo las tecnologías y los procedimientos empleados, sino que también, se hará especial hincapié en cuestiones referidas al hardware, dado que la funcionalidad del sistema depende en gran parte de los dispositivos que sean utilizados.

A nivel de hardware se especificarán dos secciones: una de ellas referidas al hardware de simulación –definiendo los componentes que fueron utilizados para la etapa previa de simulación, antes de la implantación-, y la otra, al hardware de implantación -definiendo los componentes que serán utilizados en la puesta en marcha del sistema-.

HARDWARE DE SIMULACIÓN

Causas de simulación

Obtener el estado de los semáforos de la ciudad, como función principal del sitio, significaría la necesidad de construir un dispositivo electrónico que sea capaz de reconocer si un semáforo se encuentra funcionando normalmente o si bien presenta algún tipo de fallo. La empresa Arbit tuvo la responsabilidad de la creación del mismo.

El sitio debía ser desarrollado en paralelo a la creación del dispositivo para que el producto total pueda ser entregado en tiempo y forma al cliente. Debido a tal causa, fue necesaria la simulación de los equipos. Por un lado, se debió crear un prototipo que simulase la operatoria de los semáforos y por otro, era necesario contar con una cámara IP particular que se asemeje al funcionamiento de las instaladas y utilizadas por el municipio de la ciudad ya que no podíamos contar con las cámaras municipales.

Simulación de semáforos

En este caso particular, se procedió a la simulación de dos semáforos, los cuales funcionarían en una intersección de dos vías –un semáforo por vía-. Para simular la funcionalidad de los semáforos, la empresa Arbit desarrolló un prototipo que consiste en una placa de prueba o *protoboard* compuesta por el microcontrolador PIC18F97J60 de la marca Microchip, compatible con el estándar IEEE 802.3 para la conectividad Ethernet. La placa cuenta, además, con 3 interruptores -pulsadores o *switches*- por semáforo para simular las funcionalidades del mismo -encendido alternativo de luz verde, amarilla y roja, apagado e intermitencia-.

La comunicación entre los interruptores y el microprocesador está dada por cables puente los cuales hacen circular la corriente de un lado a otro.

La placa cuenta con 11 resistencias de valores entre 220 ohm y 10k ohm cada una. La mayoría de las resistencias colocadas en el prototipo son de 10k ohm y cumplen la funcionalidad ‘Pool Down’ [ver *Figura 1.1*] ayudando a definir un estado lógico correcto para el microprocesador, es decir, si el interruptor (botón) no está pulsado, por defecto, la resistencia ayuda a mantener un estado lógico que identifique el nulo tránsito de corriente. La causa de la colocación de estas resistencias en el prototipo es la interferencia producida en el ambiente, por ejemplo, aquella que producen los tubos fluorescentes u otros elementos, lo que puede alterar el estado lógico del circuito. Por lo tanto, el microcontrolador puede tomar un falso estado y el posterior procesamiento de la señal será incorrecto.

Las otras resistencias –de 220 ohm- solo cumplen la funcionalidad de disminuir la intensidad de la corriente. Un ejemplo son las resistencias que se encuentran en el paso de corriente hacia el LED, evitando la quemadura del mismo.

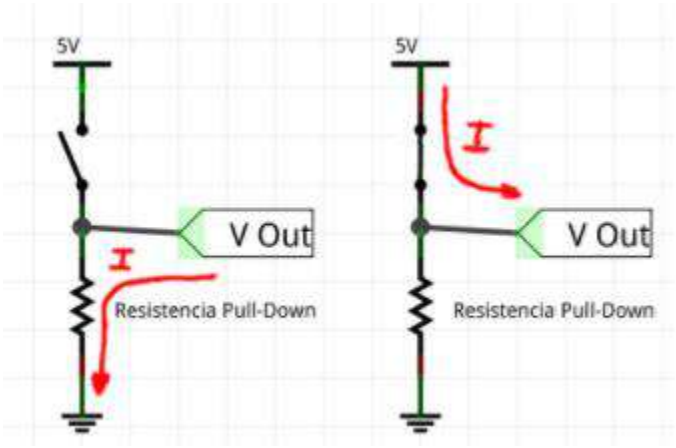


Figura 1.1 – Resistencias Pool Down

Para reconocer si el equipo se encuentra encendido o apagado, se conectó un diodo emisor de luz –LED- a la placa.

El prototipo es suministrado de energía eléctrica a través de un cable USB, que tendrá que estar conectado a una fuente de energía externa.

En las *Figuras 1.2* y *1.3* puede observarse el prototipo de simulación de semáforos realizado por la empresa Arbit junto a sus componentes.

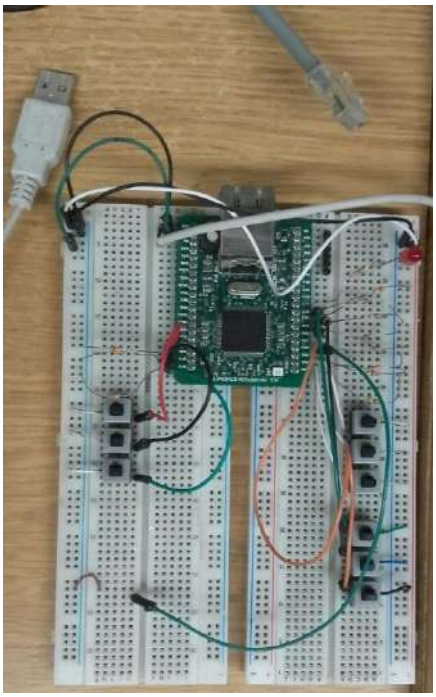


Figura 1.2 – Componentes del Prototipo

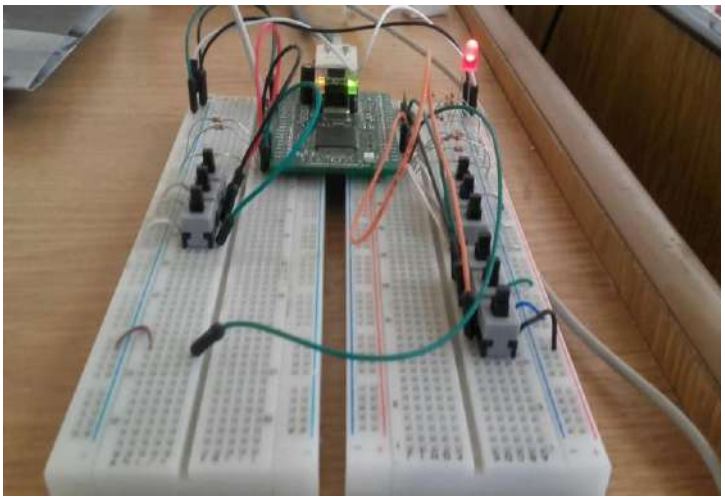


Figura 1.3 – Prototipo encendido y conectado a la red local de la empresa

Para poder reconocer los diferentes estados en los que puede transitar cada semáforo representado en el prototipo, la empresa Arbit desarrolló un firmware -software capaz de comunicarse directamente con el hardware -. El mismo se encuentra íntegramente desarrollado en el lenguaje de programación C cuyo desarrollo fue basado en la gestión de tareas y máquinas de estado.

Por un lado, se creó una librería con las variables de estado y el prototipo de las funciones a utilizar (ver *Figura 1.4*), y por el otro, se describieron tales funciones (ver *Figura 1.5*).

Se utilizó la herramienta MPLAB X IDE v3.40 con el compilador C18 V3.47 para el desarrollo del firmware y el programa provisto por Microchip MPLAB IPE v3.45 con el programador Pickit 3 para grabar o almacenar el mismo en el microcontrolador del prototipo.

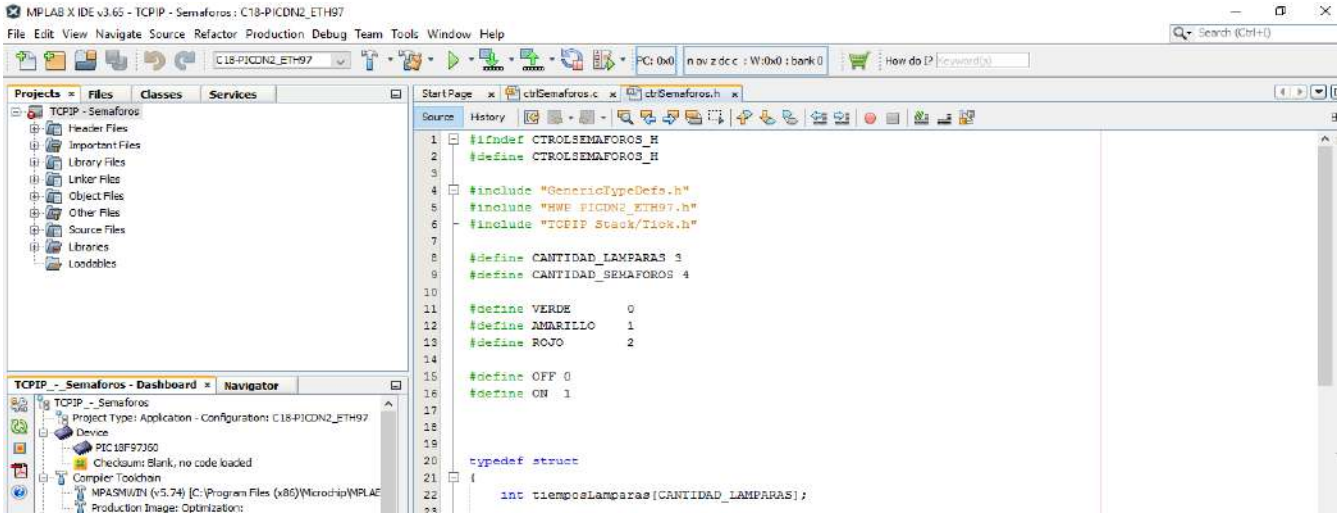


Figura 1.4 – Librería 'ctrlSemaforos.h' en MPLAB

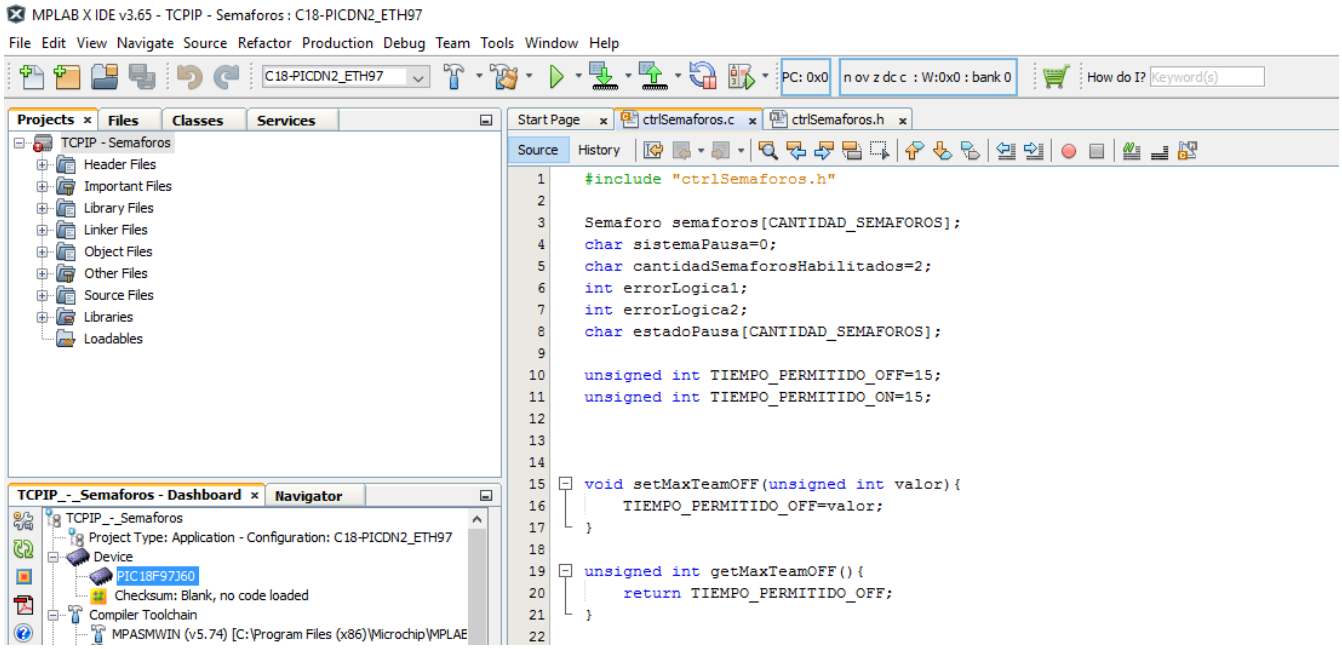


Figura 1.5 - Programa 'ctrlSemaforos.c' en MPLAB

A fin de minimizar recursos, se evitó la instalación de un sistema operativo dentro del microprocesador. Por lo tanto, se debió realizar la configuración específica de los protocolos tanto de red (IP), como de transporte (TCP) y de aplicación (SNMP, HTTP) que se utilizarán para la comunicación con el sistema web.

Como el estado del equipo es reconocido por el sistema web a través de la comunicación SNMP, fue necesario definir los OID (Identificadores de Objeto) que utilizará el protocolo y corresponder cada uno de ellos con una variable específica que luego pueda ser consultada para conocer el estado de un objeto requerido.

La marca del microprocesador –Microchip- provee librerías que implementan los protocolos TCP/IP y SNMP. Por lo tanto, la gran parte de la comunicación fue resuelta.

En complemento, se debió armar la tabla MIB –Management Information Base- para poder consultar los estados de las variables a través de la comunicación por el protocolo SNMP. La misma es un tipo de base de datos que contiene información jerarquizada en la cual se definen variables con un identificador único (OID). Éstas son utilizadas por el protocolo en función de supervisar y controlar los componentes del sistema.

Para realizar diferentes consultas SNMP al equipo – GET, WALK, SET, entre otras - se utilizó el programa de código libre, MIB Browser -desarrollado por la compañía "iReasoning, Inc."- en conjunto con el archivo MIB definido anteriormente (Ver *Figura 1.6* y *Figura 1.7*).

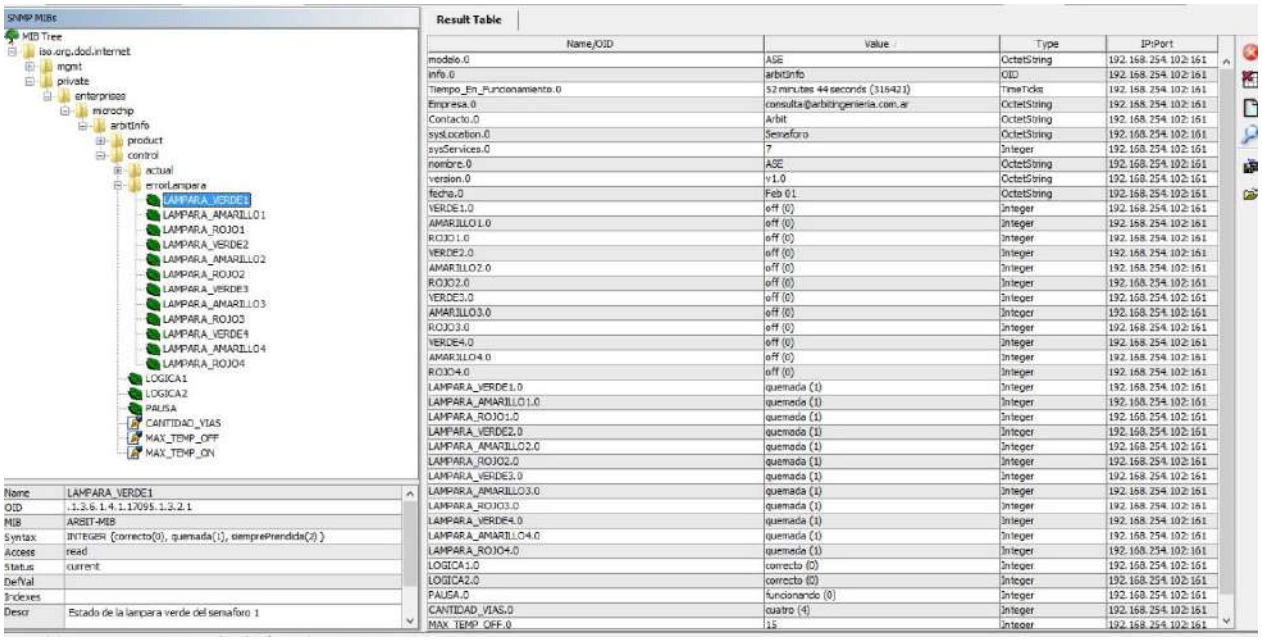


Figura 1.6- Consulta SNMP WALK al OID Principal

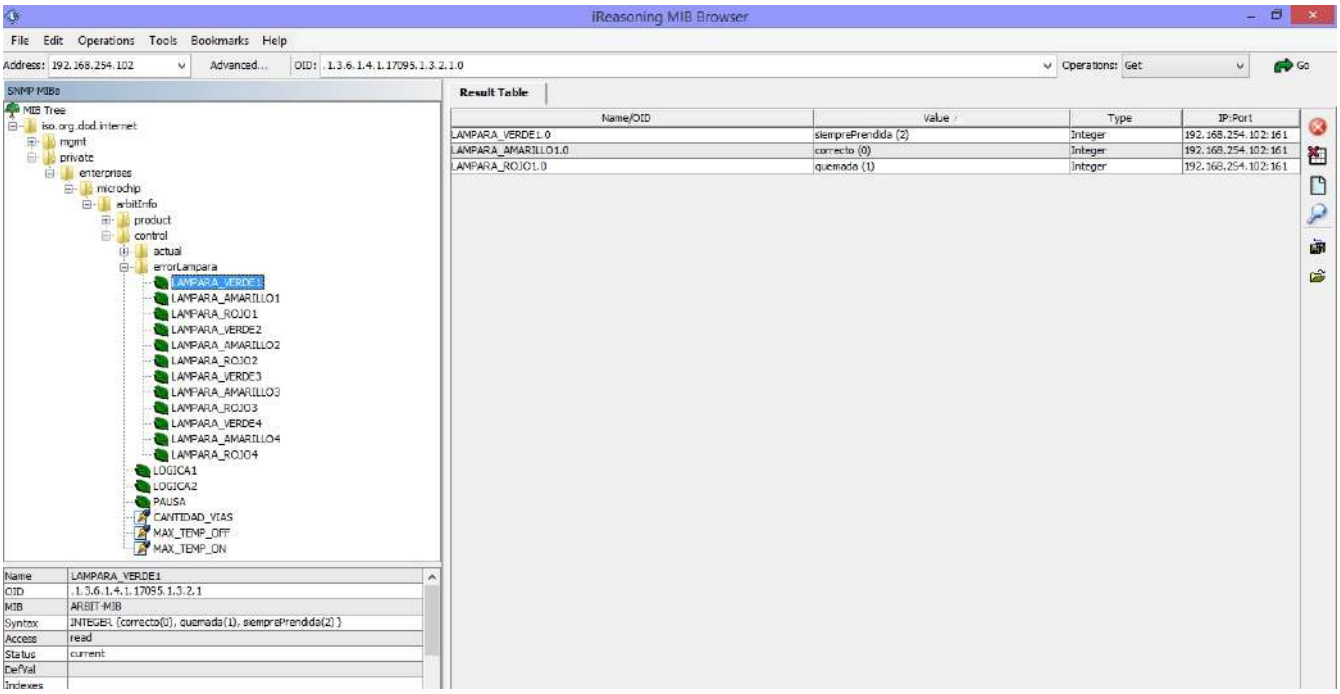


Figura 1.7- Consulta SNMP GET a un semáforo del prototipo

Por último y en complemento a lo ya descripto, la empresa también desarrolló una pequeña página web, la cual se encuentra embebida en el equipo, con el objetivo de visualizar de manera gráfica y en tiempo real los estados de los semáforos representados por el prototipo. Principalmente, esta característica fue desarrollada para facilitar el testeo del sistema (ver *Figura 1.8*).

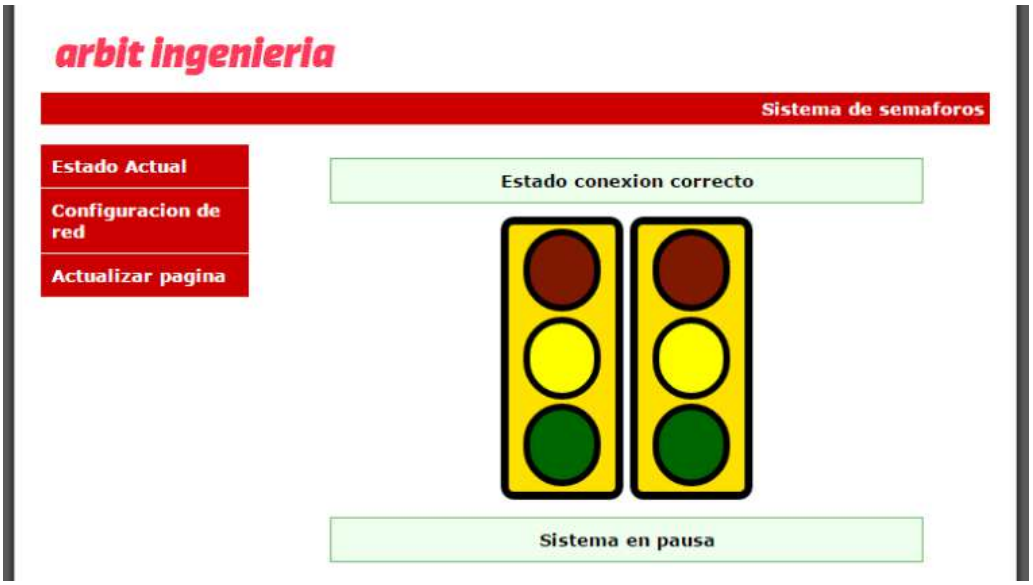


Figura 1.8- Sitio web incorporado en Microprocesador

Simulación de Cámaras IP

Ante la causa de no poder contar con el acceso a cámaras IP municipales, debió simularse la funcionalidad de las mismas. Para ello, se utilizó una cámara de carácter propio que cuenta con las siguientes características:

- Marca: Vivotek
- Modelo: IB8369A
- Descripción: aplicación al aire libre, imagen de 2 Mpixeles, resolución 1080p FULLHD, transición de video de 2 MPixeles a 30 cuadros por segundo con la compresión H.264.
- Imagen ilustrativa:

La cámara debió ser configurada correctamente, a través del sitio embebido en la misma, para que los videos que fueran registrados allí puedan ser almacenados directamente en una carpeta del servidor -encargado de proveer el servicio del sitio web- (Ver *Figuras 2.1* y *2.2*). Esa conexión entre el servidor y las cámaras para la transferencia de archivos fue establecida a través del protocolo FTP (Protocolo de Transferencia de Archivos).

Una vez almacenados los archivos en el servidor, los mismos serán ordenados por fecha de emisión. Un script ejecutado allí cada un minuto será el responsable de tal ordenamiento. Esta acción permite que luego sea posible descargar los videos de manera ordenada y amigable desde la plataforma web. Tal procedimiento será desarrollado con mayor énfasis en instancias avanzadas del informe.

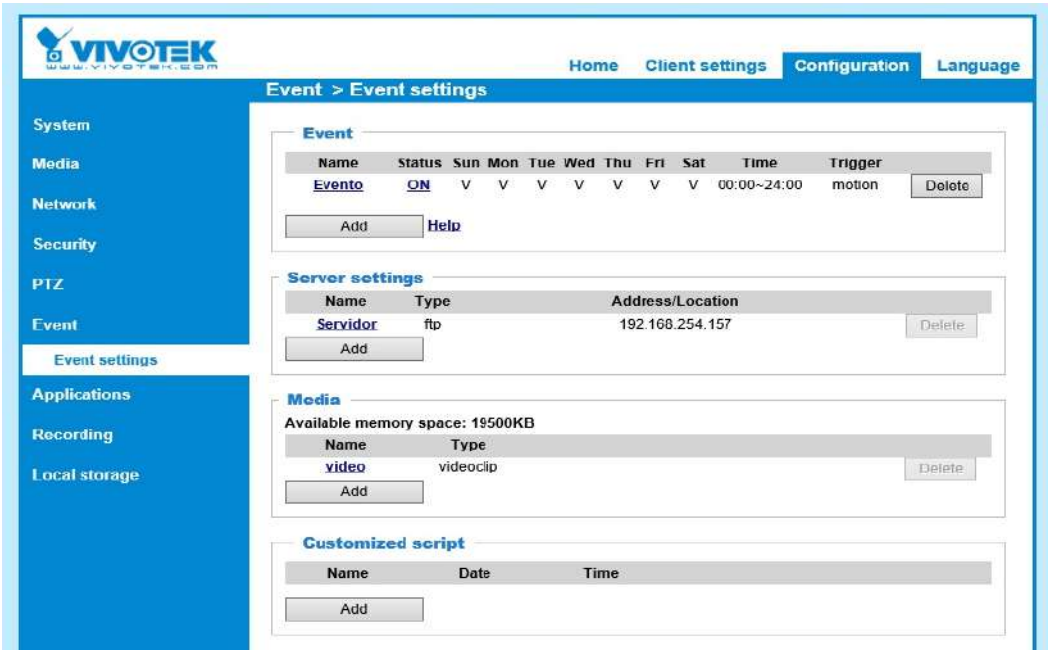


Figura 2.1 – Configuración de eventos en Cámara IP

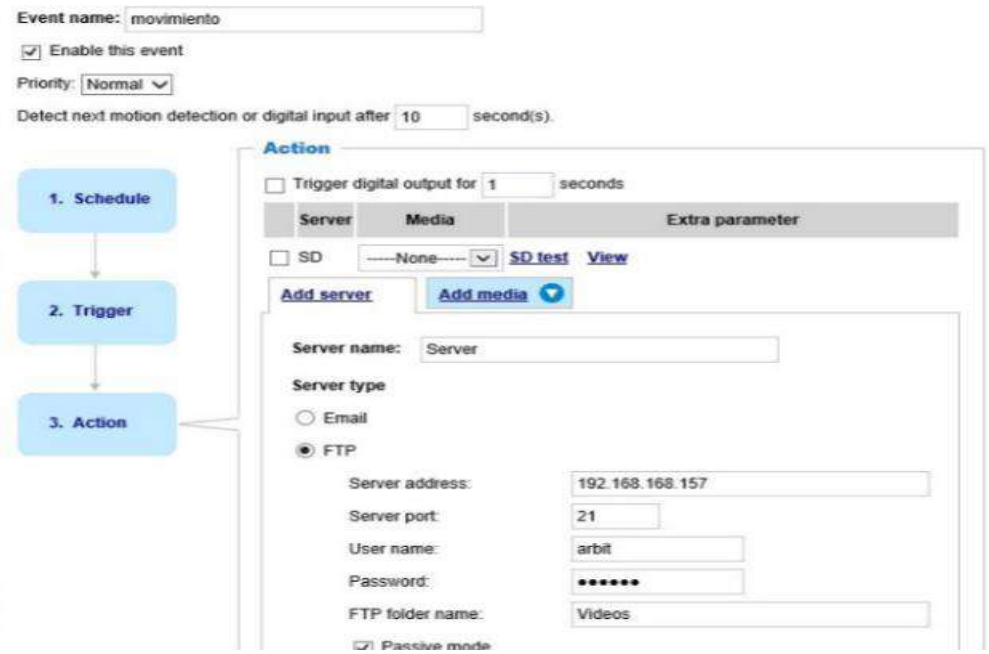


Figura 2.2 - Configuración envíos de videos al servidor

SITIO WEB

Requerimientos del Sistema

Luego de reiteradas entrevistas con los funcionarios encargados del Departamento de Señalización de Tránsito del municipio, ha sido posible detallar los siguientes requerimientos funcionales y no funcionales del sistema web solicitado.

Requerimientos Funcionales

- Alta, baja y modificación de semáforos
  - Datos:
    - Ubicación:
      - Calles de intersección, Latitud, Longitud (referencia en un mapa)
    - Identificación:
      - Dirección IP
        - Verificación de conectividad
    - Estado Actual:
      - Funciona / No Funciona / Intermitente
      - Errores posibles:
        - Error de conexión
        - Intermitencia fuera de rango
      - Descripción.
    - Datos Generales:
      - Mantenimiento a cargo de Municipio u otra entidad
      - Tipo de ópticas: LED o Incandescente
      - Nro. de ópticas
      - Nro. de Vías
      - Control: Si o No
    - Horarios:
      - Hora de Inicio
      - Hora de Fin
    - Infraestructura
      - Nro. de cuerpos
      - Burletes: Si o No
      - Nro. de cabezales
      - Nro. de Viseras
      - Nro. de columnas
      - Nro. de Pescantes
      - Nro. de Soportes

        - Ubicación bases: Si o No
    - Controlador
        - Tipo de Controlador: Electrónico / Otro
        - Marca: CySe o Tacuar
        - Modelo
        - GPS: Si o No
        - Voltios: 220 o 24
        - Fecha de colocación de Batería
        - Gabinete: Si o No
    - Historial de Estados Fallidos
        - Estado, Fallo, Fecha y Descripción del error o vuelta en funcionamiento
- Alta baja y modificación de cámaras IP
    - Datos:
        - Ubicación:
            - Calles de intersección, Latitud, Longitud (referencia en un mapa)
        - Identificación:
            - Dirección IP
                - Verificación de conectividad
        - Estado Actual:
            - Funciona / No Funciona
            - Error posible:
                - Error de conexión
            - Descripción
        - Datos Generales:
            - Marca
            - Modelo
            - Capacidad total de memoria
            - Capacidad actual de memoria
            - Fecha de Instalación
        - Historial de Estados Fallidos
            - Estado, Fallo, Fecha y Descripción del error o vuelta en funcionamiento del equipo
- Monitoreo por minuto de estado actual de cámaras
- Monitoreo por minuto de estado actual de semáforos
- Ubicación geográfica de semáforos y cámaras reconocidos en un mapa
- Muestra de detalle respecto al estado actual de los equipos sobre el mapa

- Lista de semáforos
    - Filtro semáforos de la lista por calles de intersección, dirección IP, estado o descripción
- Lista de cámaras
    - Filtro cámaras de la lista por calles de intersección, dirección IP, estado o descripción
- Visualización de archivos de videos ordenados por fecha de emisión y actualizados por minuto
    - Carpeta contenedora de archivos identificada por fecha
    - Archivos identificados por la dirección IP de la cámara origen y las calles de la intersección a la que pertenece la misma
- Descarga/Edición/Borrado de archivos de video
- Gestión de Usuarios y Grupos de Usuarios
    - Restricción de acciones según usuario registrado:
        - Usuario Empresa Arbit: Completo acceso
        - Usuario Administrador Municipal: Completo acceso exceptuando control de grupos de usuarios
        - Usuario Administrador semáforos: Sólo ABM y Monitoreo de Semáforos
        - Usuario Administrador de Cámaras: Sólo ABM y Monitoreo de Cámaras

En la *Figura 3.1* se observa la planilla de cálculo con la que cuenta el municipio para la registración de los semáforos y sus características. La misma fue utilizada para determinar los requerimientos funcionales de los mismos.

Figura 3.1 – Planilla de Cálculo: Registración de semáforos municipales.

Requerimientos No Funcionales

- Sistema amigable al usuario. Los operadores del sitio deben en corto lapso de tiempo familiarizarse con las funcionalidades del mismo.
- Adecuada interfaz gráfica. Debe ser posible reconocer rápidamente cada elemento. Los textos contenidos deben tener buen contraste con el fondo para su fácil lectura.
- Fluida navegación por el sitio.
- Primer entregable del sitio con tiempo máximo de un mes. El municipio de la ciudad desea en ese tiempo estimado comprobar la viabilidad del sistema para definir su continuidad.
- Sitio seguro. El sitio solo puede ser accedido por los usuarios registrados, es decir, identificados por el nombre usuario y contraseña correspondiente.
- Escalabilidad. El diseño del sistema debe contemplar futuras proyecciones. Debe estar diseñado para continuar mejorándolo, adaptando nuevas funcionalidades en corto plazo.
- Manual y video explicativo para el usuario.

Tecnologías Utilizadas

Sistema Operativo

Al momento de elegir un sistema operativo para el servidor que contendrá el sitio, fue necesario comparar información de varios de ellos para identificar cual era el más adaptable a las necesidades del sitio.

Luego de comparar diferentes sistemas operativos, se observó que los más adecuados a las necesidades eran los siguientes:

- CentOS
- Ubuntu Server

La causa de tal elección se debió a que ambos eran de código abierto y se podrían ajustar correctamente a las necesidades del proyecto. Dado que el sitio no requería de grandes complejidades en cuestiones de seguridad -ya que se trata de un sistema de uso interno- o en cuestiones de compatibilidad con diferentes dispositivos y protocolos – ya que solo se requería que soporte algunos protocolos como FTP, SNMP, HTTP, entre otros y se logre comunicar correctamente tanto con los semáforos como con las cámaras IP-. Además, ya se contaba con la disposición de los instaladores correspondientes para ambos sistemas operativos, lo que beneficiaría al pronto comienzo de desarrollo del proyecto.

Cuadro comparativo:

CentOS	Ubuntu Server Edition
Basado en Red Hat Linux Enterprise.	Basado en Debian.
Actualizado con menos frecuencia.	Actualizado con frecuencia.
Más estable y seguro debido a las actualizaciones infrecuentes.	Los paquetes actualizados pueden ser inestables y no seguros, lo cual es poco probable ya que se someten a pruebas vigorosas antes de ser lanzados a la versión oficial.
No hay suficientes tutoriales y tiene una base de usuarios más pequeña.	Documentación rica, comunidad activa y variedad de tutoriales disponibles en línea.
Paquetes .rpm y gestor de paquetes ‘yum’.	Paquetes .deb y el gestor de paquetes 'apt-get'.

Si bien ambos sistemas operativos reunían los requisitos necesarios para el proyecto, se decidió proceder a instalar y utilizar ***Ubuntu Server*** en su versión 16.04.3 LTS (64 bits) ya que era el más adecuado dado a los propios conocimientos y experiencias anteriores sobre la gestión de paquetes, instaladores, herramientas de programación, entre otras. Además, los sistemas operativos basados en Debian son los sistemas que comúnmente hemos manejado a lo largo de la carrera universitaria, por lo que ello aportaba mayor seguridad a la hora de implantar el servidor.

Máquina virtual

Se decidió instalar el sistema operativo en una máquina virtual para permitir la portabilidad del servidor. Es decir, una vez que el sistema se realice en su totalidad y funcione correctamente, exportaremos la máquina virtual del servidor y la colocaremos en la máquina del municipio desde donde se ejecutará el sistema.

En la máquina física que se utilizó para crear el sitio, se encontraba instalado el sistema operativo Windows 7 de 64 bits, por lo que fue necesario instalar el programa ***Oracle Virtual Box*** en su versión 5.1.8 para poder ejecutar allí la máquina virtual con el sistema operativo Ubuntu Server.

Algunas características de la máquina física son:

- Memoria RAM: 4Gb
- Disco: 500 Gb
- Procesador: AMD A6 6400k 4.1 Ghz.
- Placa de red: Wireless LAN y LAN PCI Express TP-LINK

Algunas características de la máquina virtual son:

- Memoria RAM: 2 Gb
- Disco: 15 Gb
- Red: Conexión configurada como adaptador puente, asociada a la placa física utilizada, que puede ser tanto placa de red inalámbrica como alámbrica (contando con ambas opciones).

Operador de archivos

El sistema operativo elegido no provee interfaz gráfica, por lo tanto, solo puede operarse a través de la ejecución de líneas de comando. Dado a esta característica se ha escogido utilizar la herramienta ***WinSCP*** en su versión 5.9.5 que ha sido instalada en el S.O Windows (sistema operativo base de PC física) para lograr acceder al servidor vía SSH –medio de transferencia seguro- y permitir efectuar operaciones básicas de archivos tales como crear, modificar, localizar, eliminar, mover y descargar, entre otras. Además, el sistema fue utilizado para transferir archivos entre ambos sistemas operativos ya que provee facilidad y seguridad para tal operación.

Servidor

El servidor elegido para montar el sitio fue el servidor HTTP ***Apache*** en su versión 2.4.18 ya que es un servidor web de código abierto, con buena base informativa -foros, sitios, entre otros-, extensible y multiplataforma –funciona en sistemas Windows, Macintosh y Unix-. Por lo tanto, se puede asegurar que funciona perfectamente en el sistema operativo elegido: Ubuntu Server.

Luego de la instalación del servidor, fue necesario reconocer los siguientes archivos y sus funcionalidades para su posterior modificación:

- Ubicación del archivo de configuración Apache, *apache2.conf*, para configurar aspectos del sitio que queremos se vean reflejados en el acceso al servidor (ver *Figura 4.1*).

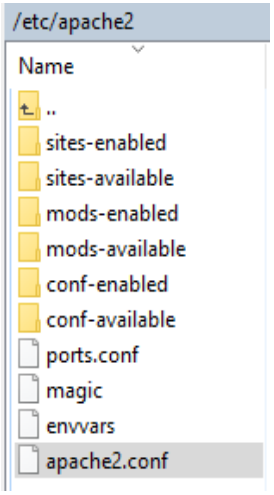


Figura 4.1 – Archivo de configuración Apache

- Localización del directorio raíz del servidor, para poder almacenar allí, por cuestiones de organización, los archivos estáticos del sitio y además configurar el fichero en el cual se inicia la interacción con el sistema web –página web por defecto denominada *index.html*-. (ver *Figura 4.2*).

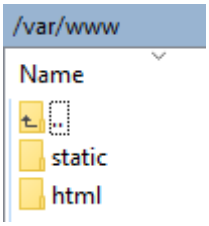


Figura 4.2 – Directorio raíz del servidor

Base de Datos

El Gestor de Base de Datos instalado en el servidor fue ***MySQL*** en su versión 5.7.19. Los motivos de la elección fueron, entre otros, que se trataba de una base de datos *Open Sourse* -código abierto-, con muy buen rendimiento ya que es veloz a la hora de ejecutar operaciones, adaptable a cualquier dispositivo al ocupar un mínimo de espacio en lo que respecta su instalación y ejecución, además, soporta gran variedad de sistemas operativos.

Por otra parte, se contaba con gran conocimiento y experiencia sobre la utilización de tal base de datos ya que ha sido utilizada a lo largo de la carrera universitaria.

Framework

Como aspecto fundamental a la hora de diseñar el sitio, se debió establecer cuál sería el marco de trabajo y los lenguajes de programación que se iban a utilizar.

Evaluando tanto los requerimientos funcionales como los requerimientos no funcionales del proyecto, poniendo énfasis en la propia experiencia con diferentes lenguajes de programación y *frameworks* se ha decidido utilizar el *framework* ***Django*** en su versión 1.9.2 el cual se basa en el lenguaje de programación Python.

Al ser un *framewok* que no ha sido presentado y utilizado en la carrera universitaria, se consideró necesario describir en el presente informe su funcionamiento y la organización interna que provee para el desarrollo de un sitio.

DJANGO

Descripción general

Django es un *framework* web de alto nivel, escrito en el lenguaje de programación Python, que ayuda a obtener un desarrollo rápido y un diseño pragmático a la hora de crear un sitio.

Resuelve una buena parte de los problemas del desarrollo web de tal manera que permite realizar mayor hincapié en *qué* es lo que se debe hacer que en *cómo* se debe hacer. Es gratuito y de código abierto.

La causa de posibilitar el rápido y eficiente desarrollo de un sitio hace referencia a la utilización de las herramientas o estructuras ya definidas –modelos, vistas y plantillas- que provee por defecto el administrador de Django para satisfacer completamente ciertas funcionalidades común a todo sitio web, como lo es, entre otras cosas, la gestión de usuarios o bien, para resolver rápidamente nuevas funcionalidades, adaptando las herramientas provistas por defecto a los requerimientos solicitados. Es decir, es posible desarrollar páginas web que utilizan la estructura por defecto creada por el administrador de Django y que luego pueden ser adaptadas a los requerimientos o bien crear páginas independientes al administrador, con un completo desarrollo propio.

Con motivo de brindar una definición global sobre el marco de trabajo que provee Django, es posible definir lo siguiente:

Django permite crear proyectos donde cada uno de ellos describe la configuración total de un sitio web. Un proyecto puede contener una o más aplicaciones que pueden verse como diferentes módulos o funcionalidades. Una aplicación puede estar a su vez en múltiples proyectos, lo que permite la reutilización de código. Estas aplicaciones, al igual que el proyecto, tienen una estructura organizacional predefinida por el *framework* donde cada uno de los eslabones que define esa estructura tiene una funcionalidad específica.

En el siguiente apartado, se describirán los elementos principales en los que se basa la estructura mencionada.

Principales elementos estructurales

Django respeta el patrón de diseño conocido como MVC: Modelo–Vista–Controlador.

- El modelo, en Django, hace referencia al modelo propiamente dicho.
- La Vista, en Django, refiere a las plantillas (*templates*).
- El controlador, en Django, refiere a la Vista.

✓ El Modelo

El modelo define los datos que serán almacenados en la Base de Datos. Por ende, es una estructura elemental en la que se basa Django para definir automáticamente las sub-siguientes estructuras.

Django define un Mapeador Objeto-Relacional (ORM) para interactuar con los datos de la base sin necesidad de utilizar sentencias SQL. Solo se utilizan funciones Python predefinidas por el mapeador para lograr tal interacción.

Por lo tanto, el modelo se conforma por clases contenedoras de variables y funciones. Las variables describen cada uno de los datos a través de la definición de parámetros como lo son el tipo, la nomenclatura, la unicidad, la longitud, las opciones de datos, entre otros. Las funciones definen la forma de acceso y manipulación a tales variables. Esta estructura permite indicar y controlar el comportamiento de los datos.

- ✓ La Vista

La vista se presenta en forma de funciones en Python y definen la lógica de cada aplicación. Las funciones pueden ser creadas con el fin de procesar los datos del modelo para luego responder a las peticiones que realiza el usuario desde la interacción con la plantillas o páginas web o bien, con el fin de definir previamente, los datos del modelo que serán visualizados.

El ORM de Django - nombrado anteriormente- permite realizar las consultas a la Base de Datos que necesitan las funciones de la vista para el procesamiento de la información.

- ✓ La Plantilla

Las plantillas son las encargadas de definir el estilo de presentación de los datos. Una plantilla está conformada básicamente por código HTML a la cual se le adhiere otras etiquetas propias de Django para interactuar con el modelo y las funciones de la vista configuradas en la aplicación. Además, para el diseño y procesamiento de los datos es posible adherirle también código XML, CSS, Javascript, CSV, entre otros.

Otro elemento estructural importante, que se encuentra fuera del ABM descrito anteriormente y es necesario nombrar ya que fue utilizado para el diseño de ciertas páginas del sitio, es:

- ✓ El Administrador

En el administrador o archivo *admin.py* de cada aplicación, es posible determinar qué datos del modelo serán visualizados y cómo serán representados en las plantillas por defecto que provee el administrador de Django.

Herramientas de programación

Con motivo de contar con una amigable herramienta de programación fueron mayormente utilizadas dos herramientas: *Kate* en su versión 3.13 (ejecutado desde S.O Ubuntu remoto) y *Notepad* ++ en su versión 7.3 (ejecutado desde S.O Windows de máquina física utilizando el gestor de archivos WinSCP). También, en un principio, se utilizó la herramienta *Atom* en su versión 1.9 (ejecutado desde S.O Ubuntu remoto a través del protocolo SSH) con diccionario de Python como ayuda sintáctica al momento de crear o modificar los diferentes archivos de configuración de Django.

Google Maps API

Para poder cumplir con el requerimiento funcional de crear un mapa donde allí se localicen las cámaras IP y los semáforos, se ha decidido utilizar la API Google Maps gratuita que provee la empresa Google. El único requerimiento para lograr embeber tal herramienta en el sitio fue crear una clave de autenticación que otorgue el permiso de utilización del código brindado por la API (Ver *Figura 5.1*).

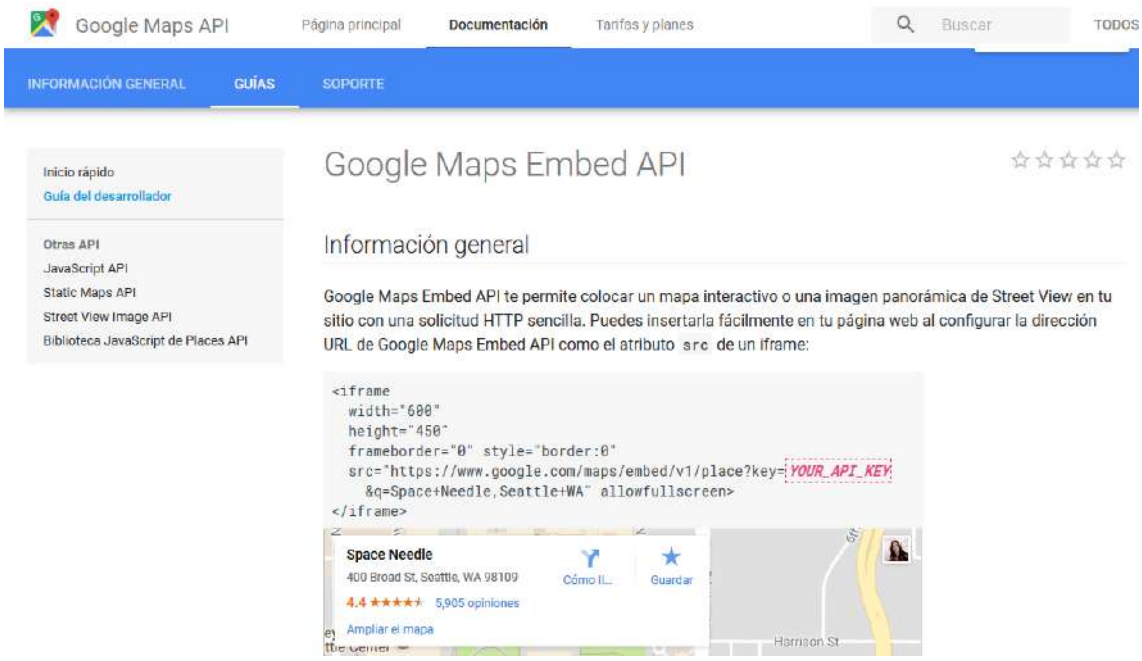


Figura 5.1 – Google Maps API

Implementación del sitio

En este apartado se describirán detalles específicos sobre la implementación del sitio. Se definirán las rutas de localización tanto del *framework* instalado con los paquetes necesarios como del proyecto creado para el sitio. Además, se identificarán los archivos creados y modificados en los diferentes módulos del sistema para lograr los objetivos deseados.

Instalación y Localización

Para proceder a instalar Django primero se debió instalar el lenguaje de programación en el que está basado el mismo, y como se describió anteriormente, este lenguaje es Python.

Una vez instalado Python en su versión 2.7 y un sistema de gestión de paquetes de Python, denominado *Pip* de versión 8.1.2 -el cual facilitó la instalación de los restantes paquetes vinculados al lenguaje como lo fueron *pyexcelerator*, *pyping*, *simplejson*, *easysnmp*, entre otros – se procedió a instalar Django, ejecutando la sentencia *python setup.py install* una vez situados dentro del directorio del instalador.

Creación del Proyecto

Una vez instalado el *framework* Django, se creó el proyecto denominado *municipio* que contiene las configuraciones generales del sitio. Fue creado dentro del directorio */opt/* del servidor -se eligió ese directorio por cuestión de fácil acceso, pero cualquiera pudo haber sido el mismo para almacenar el proyecto-.

Crear el proyecto significó ejecutar por consola, en la ruta del directorio propuesto, la sentencia: *django-admin startproject municipio*. Luego de la ejecución, Django crea por defecto las siguientes carpetas, subcarpetas y archivos dentro del directorio base –lo que se consideró anteriormente como la estructura organizacional que provee Django-:

/opt/
    municipio/
        manage.py
        municipio/
            __init__.py
            settings.py
            urls.py
            wsgi.py
            apps.py

Estos archivos refieren a:

- El primer directorio denominado *municipio/* es sólo una carpeta contenedora del proyecto que puede renombrarse en cualquier momento, ya que no modificará nada en el procesamiento del sitio.
- *manage.py:* Una utilidad de línea de comandos que te permite interactuar con el proyecto de varias maneras. Más adelante se ejemplificará su uso en el proyecto.
- El directorio *municipio/* interno es el paquete Python para el proyecto. Su nombre es el nombre de paquete Python que se necesita usar para importar cualquier elemento dentro del mismo (por Ej. *municipio.urls*).
- *municipio/__init__.py*: Un archivo vacío que le dice a Python que este directorio debe considerarse un paquete Python.
- *municipio/settings.py*: contiene la configuración del proyecto Django.
- *municipio/urls.py*: contiene la declaración de las URL del proyecto Django; como si fuere una “tabla de contenidos” del sitio.

- *municipio/wsgi.py*: Punto de entrada para servir el proyecto mediante servidores web compatibles con WSGI.

Los archivos definidos hasta el momento que debieron ser adaptados para el desarrollo del sitio fueron los siguientes:

- *municipio/wsgi.py*: se ejecuta la aplicación *Interfaz de Puerta de enlace de Servidor Web (WSGI)* , la cual cumple la función de una interfaz simple y universal entre el servidor web y el *framework* para el lenguaje de programación Python. Para ello, en el archivo fue necesario definir la ruta donde se encuentra el contenido del proyecto en el sistema operativo y asociar a la variable de entorno de Django *DJANGO_SETTINGS_MODULE* los componentes del archivo de configuración del proyecto -*settings.py*-. Más adelante, se describirá cómo es la configuración inversa, es decir, la que se realiza en el servidor apache para que el mismo asocie sus entradas con el proyecto Django.
- *municipio/urls.py*: Se definen dentro del módulo *urlpatterns* cada url del proyecto, para que las mismas sean reconocidas por el servidor.
- *Municipio/settings.py*: se configuran los valores predeterminados de las variables estáticas pertenecientes al proyecto en cuestión. Algunos de los aspectos más relevantes de configuración son, entre otros:
    - Definición de las aplicaciones vinculadas al proyecto
    - Definición de la conexión a la base de datos
    - Configuración de *templates* o plantillas del proyecto
    - Localización de elementos estáticos del proyecto (CSS, JS, IMG, etc.)
    - Localización del directorio base del proyecto
    - Localización de la aplicación WSGI

Configuración del servidor

Para que el servidor apache pueda reconocer y ejecutar el sitio creado en Django, fue necesario configurar los siguientes archivos del servidor -nombrados anteriormente- de la siguiente manera:

- Modificación del archivo *apache2.conf* procediendo a configurar las asociaciones de las rutas válidas del servidor con el proyecto Django a través de la Interfaz de Puerta de enlace de Servidor Web (WSGI).

- Modificar archivo index.html para que el mismo redireccione a la página inicio del administrador de Django configurada previamente como */syc/admin.* (Ver *Figura 6.1*)

index.html

```
1  <HTML>
2    <HEAD>
3        <!-- Send users to the new location. -->
4        <TITLE>redirect</TITLE>
5        <META HTTP-EQUIV="refresh"
6          CONTENT="0;URL=/syc/admin">
7    </HEAD>
8    <BODY>
9        <CENTER>
10           <a href="/syc/admin"> SYC | Arbit Ingenieria </a>
11       </CENTER>
12   </BODY>
13 </HTML>
```

Figura 6.1 – Archivo Index.html del servidor

Creación de Superusuario

Para lograr un primer acceso al sitio fue necesario crear una cuenta de superusuario dentro del proyecto Django. El comando ejecutado dentro de la carpeta del proyecto en cuestión para realizar tal operación fue: *python manage.py createsuperuser.*

Luego de la ejecución del comando, el sistema pretende como entrada obligatoria la configuración del nombre de usuario, la contraseña del mismo y el e-mail – de forma opcional-.

Una vez configurado el proyecto del sitio con la creación del superusuario, el administrador de Django permite el acceso a sus páginas web predefinidas. En el siguiente apartado se describirán las características funcionales de las páginas web creadas por defecto.

Funcionalidades provistas por el Administrador

Al crear un proyecto en Django, ya contamos con muchas herramientas que no son necesarias crearlas desde el comienzo en un sitio web, como lo es por ejemplo el gestor de usuarios. Django provee toda una estructura para ello, la cual comprende tanto una lógica funcional como plantillas adecuadas para cada funcionalidad. Permite con ello crear, *loguear*, confirmar cierre de sesión, cambiar contraseña, configurar permisos, generar grupos y registrar las actividades o acciones recientes de los usuarios en el sitio (Ver *Figura 7.1*).

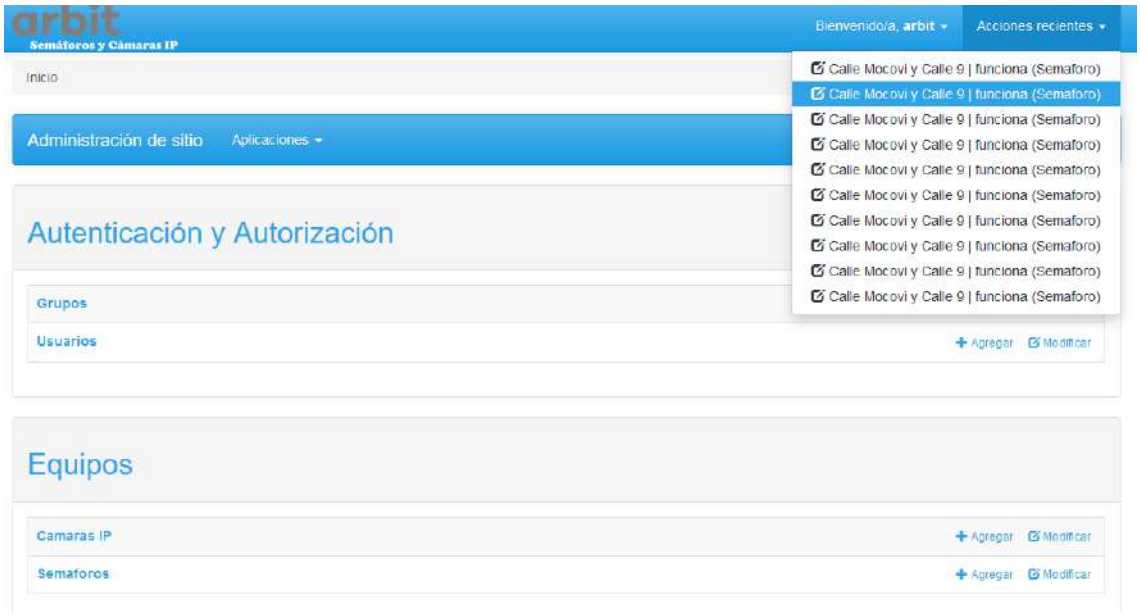


Figura 7.1 – Acciones recientes del usuario

Además, también provee por defecto, la carga, modificación y eliminación de registros de cada modelo de aplicación del sitio, así como también ver el historial de los mismos (Ver *Figura 7.2*).

Figura 7.2 – Historial de un registro

Por otro lado, brinda la posibilidad de ejecutar acciones que involucren varios ítems del listado de registros de los diferentes modelos de las aplicaciones y realizar búsquedas por cada atributo de los modelos definidos, entre muchas otras funcionalidades (Ver *Figura 7.3*).

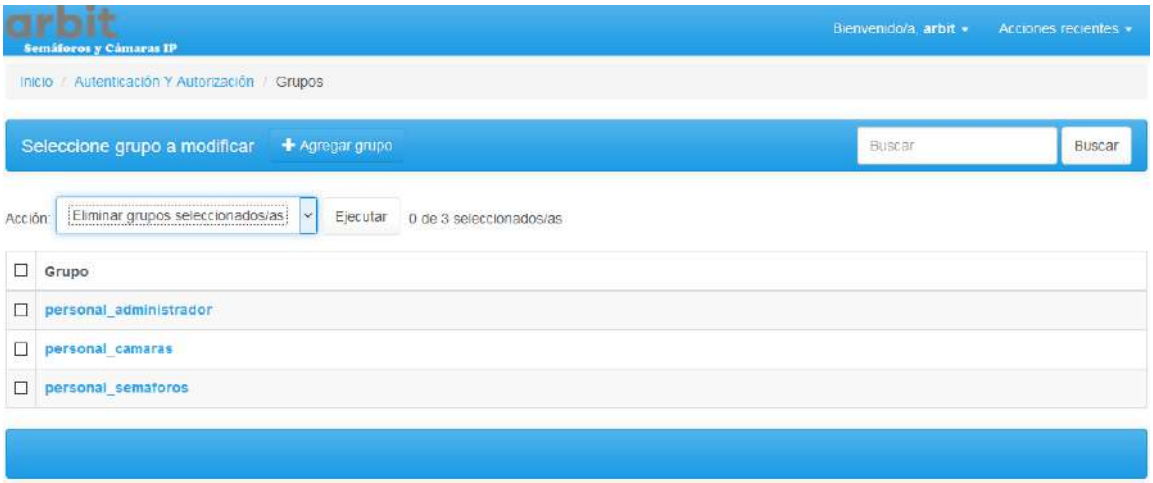


Figura 7.3 – Gestión y Búsqueda de registros

Creación de Formularios

En una sección del sitio, como una de las funciones principales, se debió lograr la gestión tanto de las cámaras IP como de los Semáforos. Tal gestión debió involucrar la creación de formularios para lograr las actividades de carga, modificación y eliminación de los equipos. Por entonces, fue necesario crear una aplicación del proyecto a la que se denominó *equipos*. La acción fue llevada a cabo ejecutando el siguiente comando en consola una vez posicionados en el directorio del proyecto:
python manage.py startapp equipos

Esta acción creó el directorio *equipos*, con la siguiente estructura por defecto:

equipos/
- *__init__.py*
- *models.py*
- *admin.py*
- *views.py*
- *apps.py*
- *tests.py*

Algunos archivos explicitados aquí no fueron definidos anteriormente, por lo tanto, son definidos debajo:

- ***models.py***: se declaran las dos clases del modelo. Una clase hace referencia a la estructura que tiene el formulario de Semáforos y otra a la estructura que tiene el formulario de Cámaras IP.
- ***admin.py*****:** se define el orden y la visualización de los elementos del modelo en las plantillas creadas por el administrador de Django.
- ***views.py***: se declaran las funciones de la vista que procesarán la información solicitada en la interacción sitio-usuario.
- ***test.py***: se definen las pruebas necesarias para verificar la correcta funcionalidad del sitio. En este caso particular, tal archivo no ha sido generado.

La aplicación *equipos* fue creada con el objetivo de describir tanto el modelo del formulario de semáforos como el modelo del formulario de cámaras IP, para que luego puedan crearse las correspondientes tablas en la base de datos. Por lo tanto, el primer paso que se realizó luego de crear la aplicación, fue definir el modelo de la misma dentro del archivo *models.py*.

Luego de crear correctamente el modelo, se debió ejecutar una serie de comandos para migrar la estructura del modelo a la base de datos configurada ya que como se ha dicho anteriormente, Django crea una base de datos con tablas y atributos basándose en el modelo definido. La serie de comandos ejecutados desde el directorio del proyecto, donde se encuentra el archivo manage.py, comprende:

- *python manage.py makemigrations* -para crear las migraciones correspondientes de acuerdo a la creación o cambio del modelo. Luego de ejeutar el comando, Django crea una carpeta dentro del directorio de la aplicación, la cual contiene archivos generados que registran dichas migraciones.
- *python manage.py migrate* -para crear las tablas en la base de dato o bien aplicar tales cambios a la misma.

Una vez creada la base de datos del proyecto con las tablas adaptadas al modelo realizado, se procedió a desarrollar cómo se verán esos datos en los *templates* del sitio: en qué orden y de qué manera. Para ello, fue necesario editar el archivo *admin.py* de la aplicación.

Hasta el momento, la palabra *templates* hace referencia a las plantillas creadas por el administrador de Django, que conforman, en este caso, la mayoría de las páginas web del sitio.

Para que la aplicación pueda ser ejecutada correctamente, se debió editar tanto el archivo *apps.py* de la aplicación, creando la clase de configuración de la aplicación, como así también agregar la aplicación al conjunto de aplicaciones definidas en el archivo *settings.py* del proyecto.

Con lo descripto anteriormente, ya se encuentran en funcionamiento los formularios para la gestión de Semáforos y Cámaras IP. Pero aún no cumplen con todos los requerimientos funcionales solicitados. Para ello, fue necesario adaptar las plantillas creadas por defecto, adhiriendo nuevas funcionalidades.

Adaptación de Formularios

Búsqueda y Localización en el Mapa

Para lograr ver geográficamente la localización de cada equipo en los formularios fue necesario incorporar a cada plantilla código *javascript.* En la misma, se embeben las variables y funciones provistas por el script que aportada la API de Google Maps adaptándolas a las necesidades del sitio.

La API permitió contar con la visualización de un mapa centrado en la ciudad de General Pico con un marcador dentro del mapa localizado en una determinada intersección que tiene como imagen un ícono del elemento al que se está haciendo referencia, ya sea un semáforo o una cámara - según corresponda-.

Determinar la intersección en la que se ubica el objeto puede realizarse de dos maneras: Una manera es definir las calles de la intersección en el campo de búsqueda agregado a los formularios por tal motivo, y otra es definirla realizando un *click* sobre el mapa. En ambos casos, la información que se extrae y es almacenada para la localización del objeto son las coordenadas geográficas del mismo, es decir, su latitud y longitud.

Otra funcionalidad incorporada en el mapa fue la de visualizar una pequeña ventana informativa sobre el estado del equipo en cuestión, al realizar *click* sobre el ícono del objeto. La ventana informativa contiene los datos de IP, dirección, estado, tipo de fallo -si existiere- y demás características propias de cada objeto (ver *Figura 8.1*).

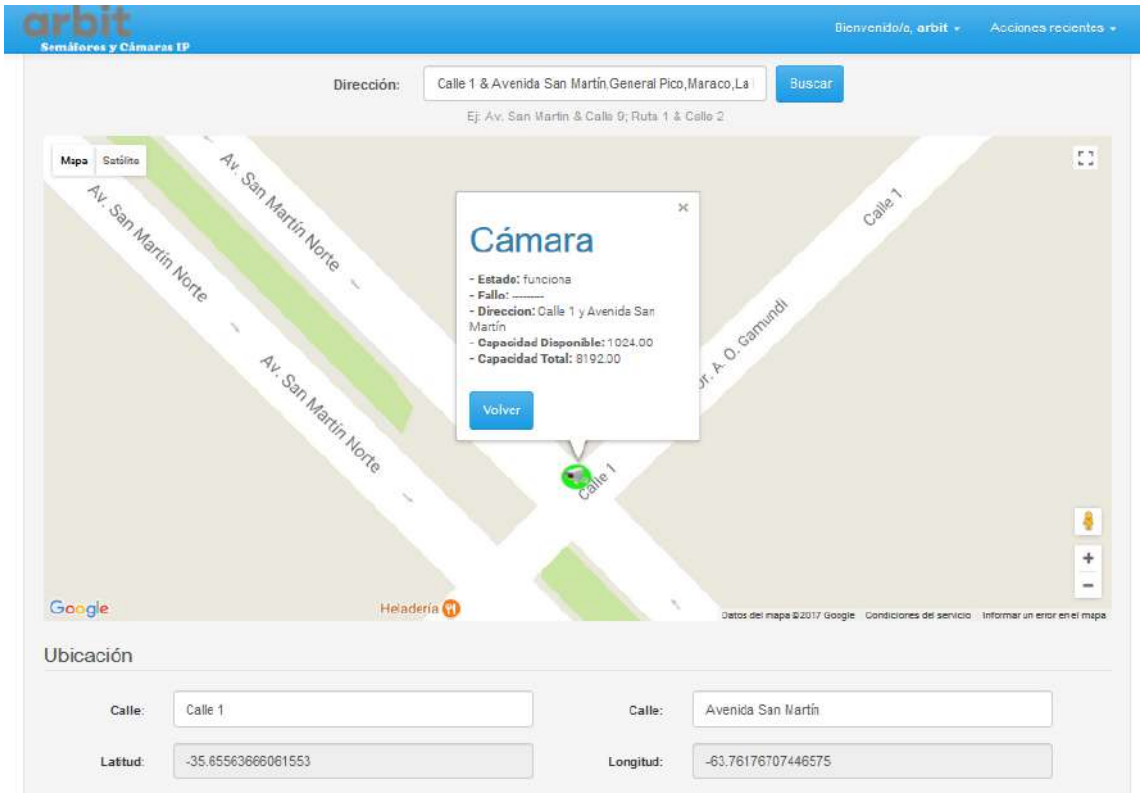


Figura 8.1 – Búsqueda y Localización en el mapa

Verificación de Conexión al Equipo

Otra vez, en ambos formularios, fue necesario embeber código *javascript* para que sea posible captar el evento del usuario al solicitar la verificación de la IP definida.

El evento es iniciado presionando la tecla *Enter* en el campo de solicitud de la dirección IP –campo obligatorio en ambos formularios- al tratarse de una identificación única del equipo. Con el fin de procesar esa verificación fue necesario crear una función en la vista de la aplicación *equipos*, a la que se denominó *Ping*. La misma recibe como parámetro la dirección IP que se requiere procesar para determinar si se encuentra o no alcanzable. Para determinar el resultado, y como lo explicita el nombre de la función, se envía un ping a esa dirección y si la dirección responde, significa que es alcanzable o en otro caso, no lo es. La función retorna un mensaje que se procesa en JS. El resultado obtenido será observado por el usuario percibiendo un color (verde o rojo según el resultado) sobre el campo de la dirección IP del formulario (Ver *Figura 9.1*).

- Como nota relevante a lo descripto anteriormente, en el código JavaScript embebido en las plantillas, se debió utilizar el método *getJSON()* provisto por la librería JQuery, para llamar a las funciones de la vista. Es decir, se utilizó el formato que provee JSON (*JavaScript Object Notation*) para el intercambio de datos realizados entre las plantillas y la vista. JSON provee un formato de texto con estructura jerárquica, por lo que es posible definir cualquier tipo de estructura para el intercambio de datos.

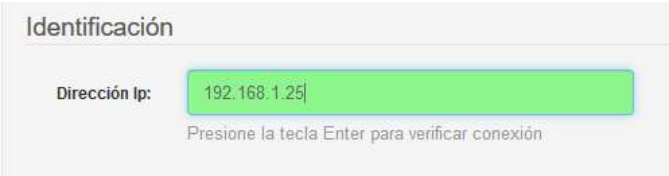


Figura 9.1 – Conexión Exitosa

Configuración de Vías para Semáforos

Como las vías sólo son definidas en el registro de Semáforos -en la plantilla que define el formulario para la gestión de los mismos-, al momento de guardar el semáforo que se quiere crear o modificar, se debe *setear* en el dispositivo la cantidad de vías configuradas en el campo *vías*. Para ejecutar tal requerimiento, fue necesario crear una nueva función en la vista de la aplicación *equipos* a la cual se denominó *set_vias*. La misma es ejecutada desde el código JavaScript embebido en la plantilla y su procesamiento es idéntico a la función *Ping* definida anteriormente, solo que su funcionalidad es diferente.

Estados, Fallos e Historial de Estados

Obtener el estado actual de los equipos significó la creación de dos scripts: uno para identificar el estado actual de cada cámara IP y otro para identificar el estado actual de cada semáforo. Ambos scripts se vinculan con los equipos mencionados a través de su IP y evalúan los parámetros requeridos para descifrar el estado de los mismos e identificar sus fallos, si los hubiere.

Con el fin de obtener información actualizada referida a los estados de los equipos en el sitio, fue necesario ejecutar cada un minuto los scripts en el servidor. La tarea programada genera que se actualice la base de datos automáticamente, si un cambio detectado en los equipos lo sugiere. Con los datos procesados en cada ejecución de los scripts, es posible obtener información actualizada sobre el estado de los equipos en un lapso muy pequeño de tiempo.

Para ejecutar cada un minuto los scripts fue necesario utilizar la herramienta de programación de tareas del S.O Ubuntu Server denominada *Cron*.

En el siguiente apartado, serán definidos ambos scripts:

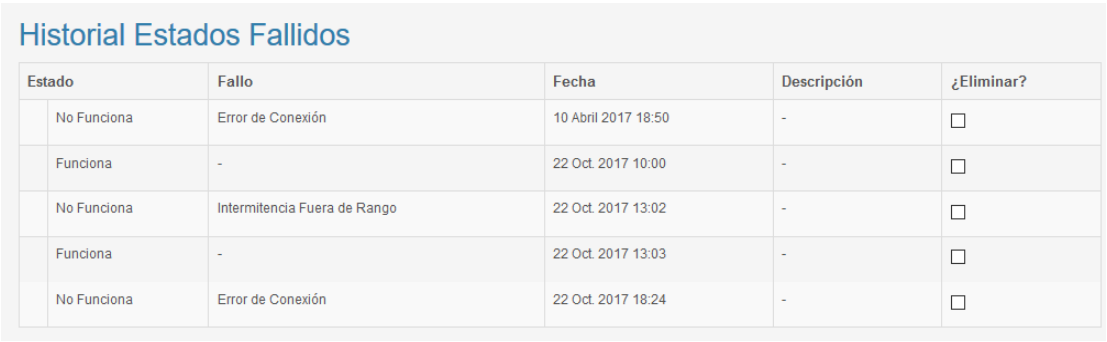
- *ConsultaEstadosSemaforos*
    - El script informa sobre el estado de los semáforos. Para entender tal funcionalidad, deben ser comprendidos los diferentes estados en los que puede transitar un semáforo. Estos estados son:
        - Funciona: estado en el que el semáforo se encuentra funcionando normalmente. Enciende luz verde, amarilla y roja de manera síncrona.
        - Intermitente: Estado intermitente del semáforo fuera de las horas activas del mismo. Cada semáforo tendrá configurado el horario en el que se considerará activo. Si existe un estado de intermitencia fuera de ese horario, se definirá al semáforo como ‘intermitente’, en lo contrario, se define un estado de fallo.
        - No Funciona: estado en el que el semáforo puede presentar fallos por alguno de los siguientes motivos:
            - Semáforo intermitente dentro de sus horas activas, es decir, dentro del horario en el que el mismo debería estar funcionando correctamente. Esta situación deriva de alguna falla previa del equipo, como pueden serlo, en este caso evaluado, la detección de una lámpara quemada o de un error lógico.

- Semáforo sin conexión. No puede establecerse la conexión con el equipo. Por ende, se considera un estado de fallo definido como error de conexión.

- *ConsultaEstadosCamaras*
  - Las cámaras, tal como lo indican los requerimientos funcionales, pueden tener dos estados: Funciona y No Funciona. El único caso en el que se detecta el no funcionamiento es cuando no es posible conectarse a la cámara. Es decir, no se puede establecer una conexión con la misma.

Con la definición y ejecución de los scripts, además de establecerse el estado actual de cada equipo y su fallo - si lo tuviere- (Ver *Figura 10.1*) se registra y almacena en la base de datos cada cambio de estado. De esta manera, se obtiene un historial de estados para cada equipo (Ver *Figura 10.2*). La función principal del historial es dar a conocer los diferentes estados que ha transitado cada equipo, es decir, cuándo falló y cuándo retomó su funcionamiento. Además, gracias a los datos brindados en cada historial, es posible el análisis y la determinación de conclusiones a nivel de toma de decisiones.

Figura 10.1 – Estado de equipo



Estado	Fallo	Fecha	Descripción	¿Eliminar?
No Funciona	Error de Conexión	10 Abril 2017 18:50	-	☐
Funciona	-	22 Oct. 2017 10:00	-	☐
No Funciona	Intermitencia Fuera de Rango	22 Oct. 2017 13:02	-	☐
Funciona	-	22 Oct. 2017 13:03	-	☐
No Funciona	Error de Conexión	22 Oct. 2017 18:24	-	☐

Figura 10.2 – Historial de Estado de equipo

La información que puede extraerse de tales datos para la toma de decisiones futuras pueden ser, entre otras, la cantidad de veces en un período de tiempo que un equipo dejó de funcionar y por qué causa; el tiempo que conllevó la reparación y vuelta en marcha de un equipo, es decir, el tiempo transcurrido desde que se generó un fallo en el equipo hasta la resolución y vuelta en marcha del mismo; los fallos más comunes de los equipos; entre otras.

El procesamiento de información descripto anteriormente aún no se ve reflejado dentro del sitio, pero es una tarea pendiente, que será realizada según la evolución de requerimientos pactados entre el municipio y la empresa.

Actualmente, el formulario de cada equipo presenta su historial de estados con la posibilidad de realizar un procesamiento de los datos de manera externa al sitio.

Creación de Vista de Videos

La página web contenedora del gestor de vídeos de las cámaras IP en el sistema no fue definida dentro de las plantillas por defecto que genera la estructura provista por el administrador de Django para el sitio, sino que fue creada como una página externa al mismo, pero a la cual se accede desde el administrador.

A causa de necesitar resolver tal requerimiento y detectar que el mismo no es más ni menos que la necesidad de crear un simple gestor de archivos que permita a los usuarios del sitio acceder a los archivos de videos de las cámaras que serán situados en un directorio del servidor para luego permitir su descarga, se llegó a la conclusión, antes de comenzar su desarrollo, que esta funcionalidad podría ya estar creada por otros usuarios programadores en la web que utilizan el *framework* Django en sus desarrollos. Luego de ir en búsqueda de tal módulo predefinido, ha sido posible encontrar un sitio, basado en desarrollo colaborativo, que proveía una estructura de aplicación Django junto con una plantilla que incluía los archivos estáticos que la misma utiliza, como lo son: estilos, código *Javascript*, fuentes e imágenes. El sitio al que se hace referencia es GitHub creado por la compañía GitHub, Inc. para alojar proyectos de diferentes usuarios, en el cual un usuario registrado al mismo, denominado Rohith Asrk, desarrolló el módulo gestor de archivos para el *framework* Django y lo alojó en el sitio para que fuera utilizado de manera gratuita por los usuarios de la web que lo requieran.

La aplicación obtenida –*filemanager*- junto a sus especificaciones fueron obtenidas de la siguiente página: *https://github.com/IMGIITRoorkee/django-filemanager* y fue adherida al sitio con normalidad, como si se tratase de cualquier otra aplicación definida, como se hizo con la aplicación *equipos*.

Para que el módulo obtenido se adapte exactamente a los requerimientos solicitados, debió ser modificado. Estas modificaciones tuvieron que ver por un lado con la adaptación de la plantilla predefinida por el módulo a los estilos que fueron definidos para el sitio. Es necesario que el sitio mantenga características y estilos comunes en cada página web que lo conforma para brindar un vínculo visual y organizacional de manera que el usuario identifique fácilmente el sitio en cualquiera de las páginas en que se encuentre ubicado.

Por otro lado, debieron ocultarse funcionalidades provistas por el módulo que no eran necesarias para cumplir con los requerimientos.

Por último, debió adaptarse el idioma, traduciendo mensajes en la vista y elementos de la plantilla desde el idioma inglés al español.

El requerimiento funcional que refiere a visualizar los archivos de videos ordenados por fecha de emisión ha sido cumplido desarrollando un script cuya funcionalidad refiere a organizar cada uno de los videos que se encuentran en el directorio raíz según la fecha actual en la que se realiza el ordenamiento, cumpliendo con la siguiente estructura jerárquica: Directorio Principal / Año / Mes / Fecha /Video (Ver *Figura 11.1*).

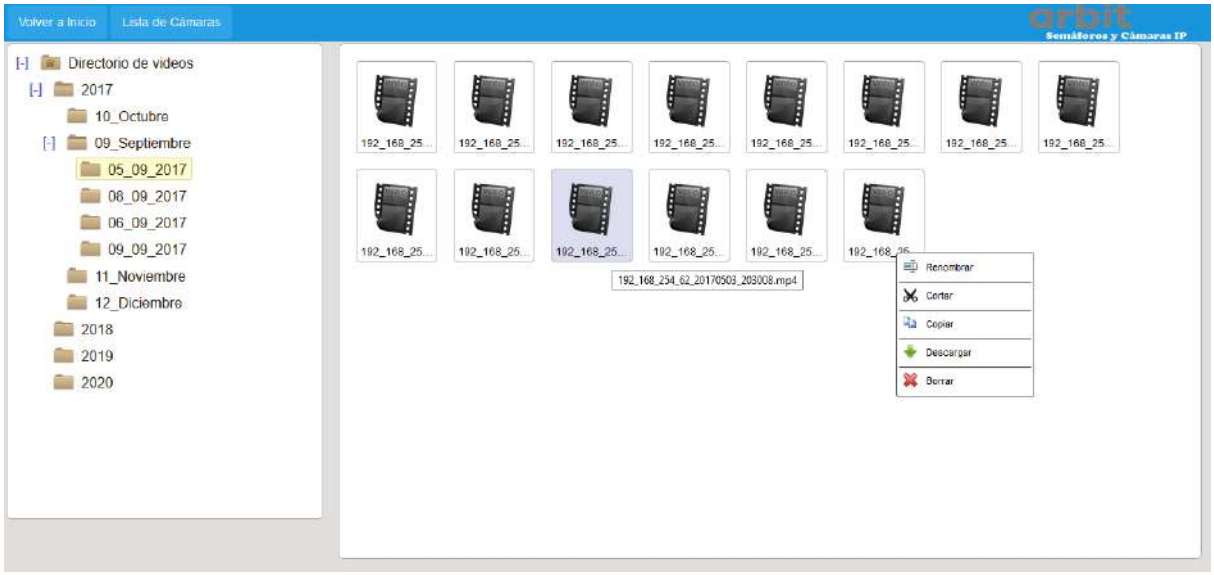


Figura 11.1 – Gestor de Vídeos de Cámaras IP

Al no ser un requerimiento funcional que los videos sean ordenados en tiempo real, se definió ejecutar el script cada un minuto, al igual que el resto de los scripts definidos anteriormente. Para ello, se incorporó el mismo al programador de tareas del sistema operativo Ubuntu Server.

Creación de Vista Operador

Esta vista define el principal requerimiento del sitio cuyo objetivo general refiere a contar con la visualización de un mapa en el cual puedan reconocerse fácilmente, a través de su ubicación, tanto las cámaras como los semáforos que fueron almacenados previamente y el estado en el que se encuentra cada uno de ellos. Además de proveer la visualización de los equipos y sus estados, cuenta con la opción de brindar una descripción detallada de cada equipo la cual define varios de sus atributos como la dirección IP, las calles de la intersección en la que se encuentra ubicado, el estado, el fallo, entre otros.

Los semáforos y las cámaras que serán gestionados pueden encontrarse en puntos geográficamente cercanos unos de otros. Esta situación puede entorpecer la pronta identificación de los mimos, por lo tanto, fue necesaria la creación de filtros para visualizar fácilmente cada equipo y su estado. Los filtros realizados proponen visualizar sólo las cámaras o sólo los semáforos según corresponda.

Crear un mapa con la visualización de los equipos significó incorporar código Javascript a la plantilla *vista_operador* para utilizar las funciones que brinda la API de *Google Maps*. La API permite, además de crear un mapa con centro en la ciudad de General Pico, incorporar elementos al mapa que respondan a eventos como son los marcadores y las ventanas informativas. Los marcadores son utilizados para marcar en el mapa los equipos según su localización y las ventanas informativas fueron utilizadas para describir las características de cada equipo.

Los marcadores que hacen referencia a las cámaras IP son representados por diferentes imágenes adaptadas según el estado del dispositivo: imagen de cámara IP con fondo verde, si la cámara se encuentra funcionando e imagen de cámara IP con fondo rojo, si la misma presenta un fallo. De la misma manera, los marcadores que refieren a los semáforos fueron representados por imágenes adaptadas según su estado: imagen de semáforo con fondo verde si el mismo se encuentra funcionando normalmente, imagen de semáforo con fondo amarillo si el mismo se encuentra en estado intermitente en su período de reposo o pausa, e imagen de semáforo con fondo rojo si el mismo presenta un fallo (Ver *Figura 12.1*).

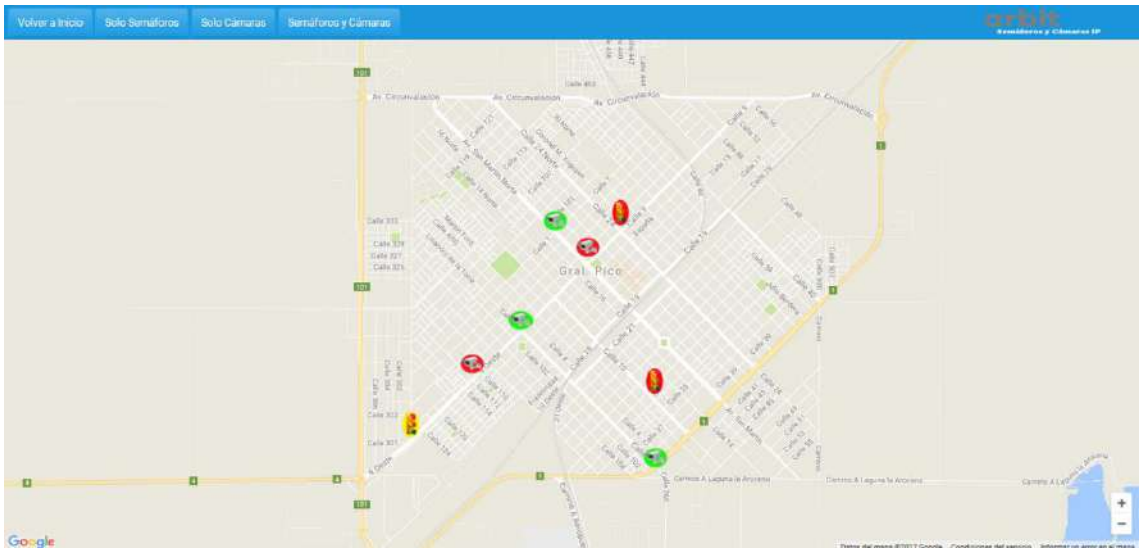


Figura 12.1 – Localización y Estado de Equipos

Cuando los equipos se encuentran funcionando normalmente, son representados por marcadores de fondo verde – o amarillo en caso de semáforo intermitente- que mantienen su posición en el mapa. En cambio, cuando algún equipo presenta un fallo, el marcador de fondo rojo que hace referencia al equipo en el mapa comienza a moverse verticalmente con corta longitud de desplazamiento. Esta característica fue diseñada con el fin de alertar al operador que visualizará la página para que el mismo detecte fácilmente un fallo en los equipos y rápidamente implemente la metodología adecuada para retomar el normal funcionamiento de los mismos.

El operador de la vista puede obtener más información de los equipos presentados como marcadores en el mapa, desplegando una ventana informativa en cada uno de ellos que describe las características principales de los mismos junto a su estado (Ver *Figuras 12.2, 12.3, 12.4 y 12.5*).

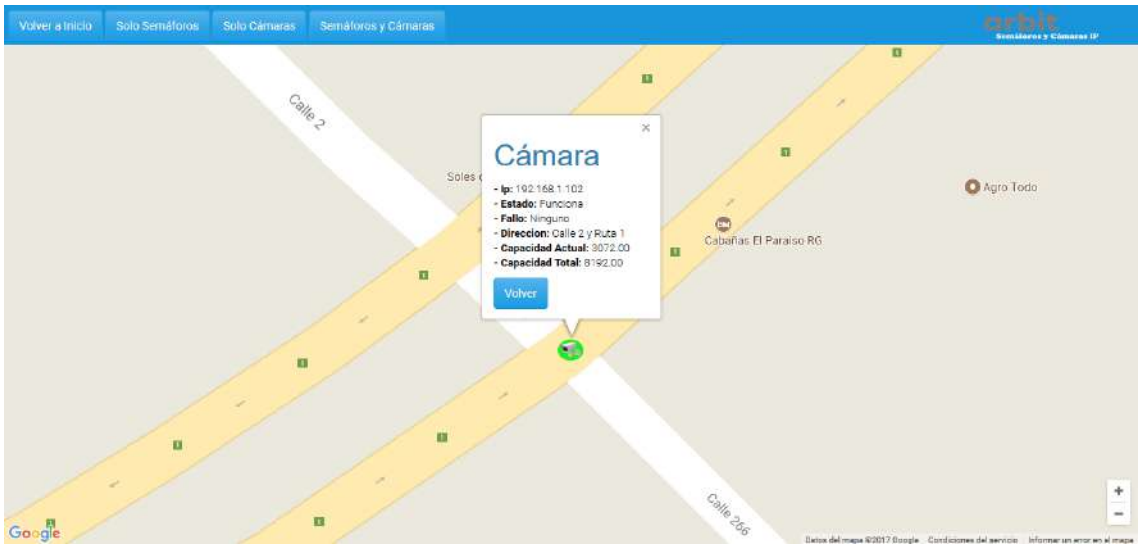


Figura 12.2 – Ventana Informativa de Cámara IP

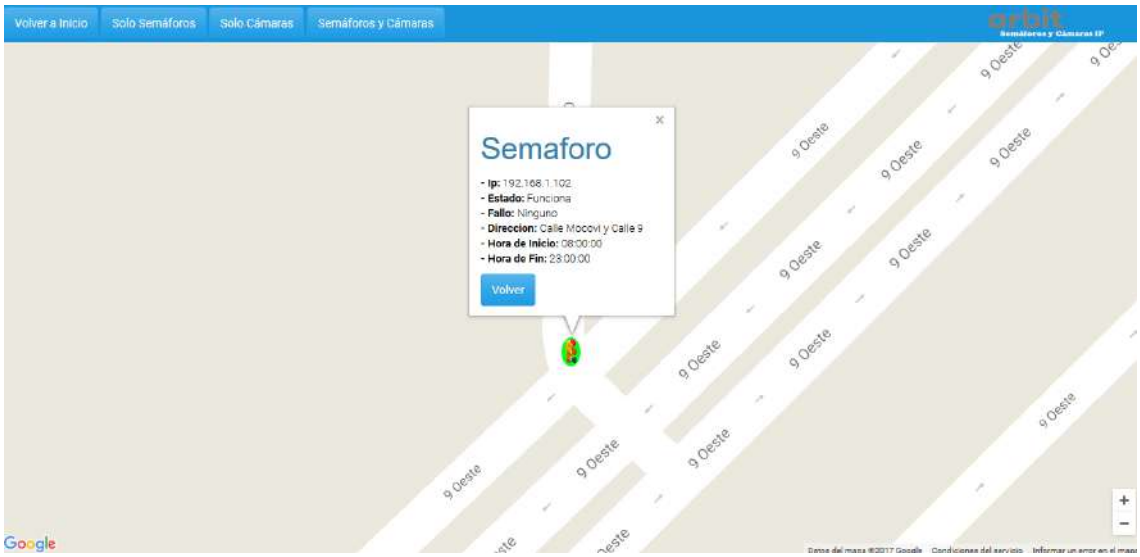


Figura 12.3 – Ventana Informativa de Semáforo

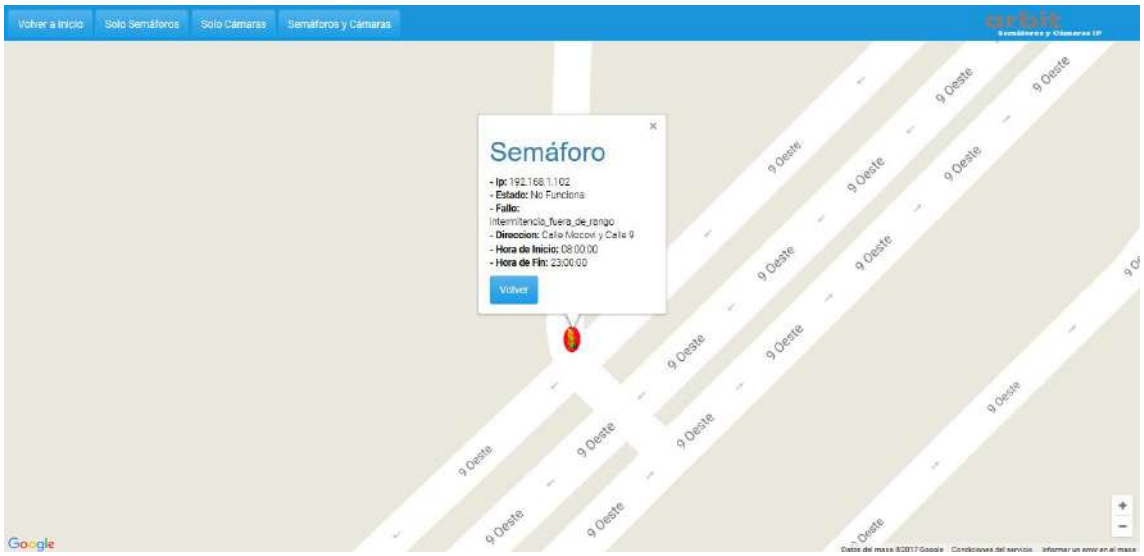


Figura 12.4 - Semáforo con Notificación de Fallo: Intermitencia Fuera de Rango

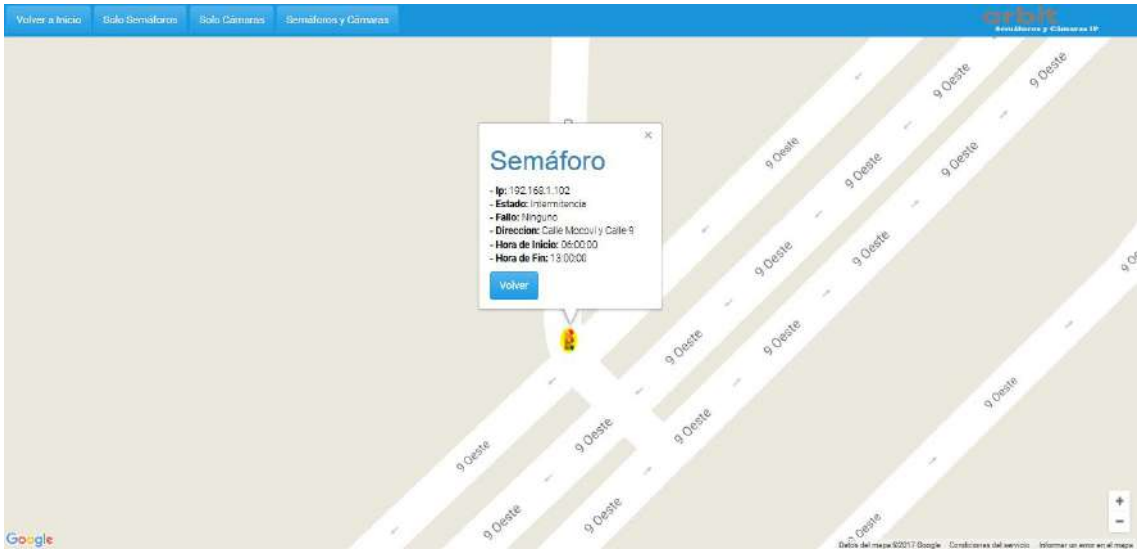


Figura 12.5 – Semáforo con Notificación de Estado: Intermitente

Características del Sitio

Estilos

Al momento de pensar en darle un estilo a las diferentes plantillas del sitio que se adecúe lo mejor posible a los requerimientos solicitados y que a su vez no conlleve muchas horas de programación y diseño, se evaluó la posibilidad de utilizar los estilos predefinidos que provee el *framework Bootstrap*.

Bootstrap no solo provee hojas de estilos –*CSS*- sino también, define las fuentes utilizadas y el código *javascript* –*JS*- necesario para el diseño particular que sea escogido. Gracias a tales características, el *framework* facilita la organización de contenido, permite la aplicación de estilos y provee una forma de desarrollar páginas web adaptables a diferentes tecnologías como son los dispositivos móviles.

El diseño que ha sido elegido para el sitio proviene de los estilos que ofrece una rama del *Bootstrap* denominada *Cerulean*.

Para aplicar estos estilos automáticamente a las plantillas base de las que heredan el resto de las plantillas del administrador de Django, fue necesario instalar el paquete ***django-admin-bootstrap*** en su versión 2.5.7.

Por otro lado, fue necesario descargar de la página oficial los estilos y demás paquetes referidos al tipo *Bootstrap* elegido. Luego, el contenido fue volcado en el directorio /var/www/static/bootstrap del servidor para que las plantillas que no pertenecen al administrador de *Django*, como lo son la vista operador y la vista de video logren importar estos estilos. De igual manera, y en complemento con los estilos provistos por *Bootstrap*, en tales plantillas fue necesario generar estilos propios específicos a determinados requerimientos.

En la *Figura 13.1* puede observarse la vista representativa de elementos con el estilo *Bootstrap Cerulean* que fueron utilizados.

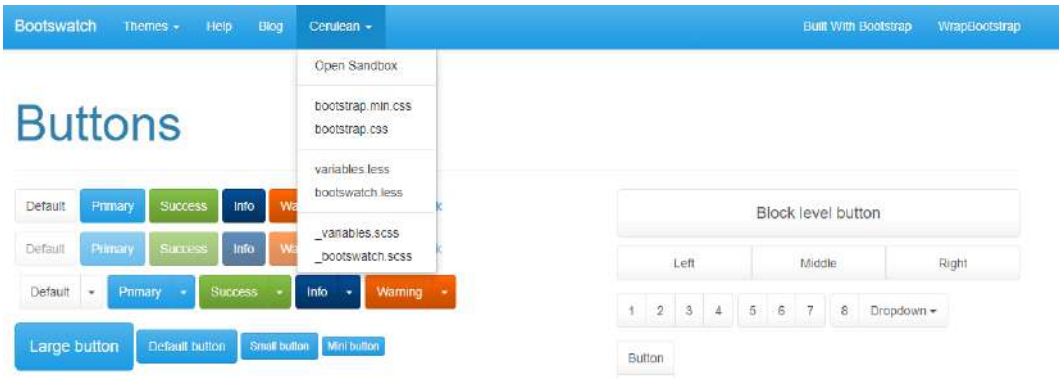


Figura 13.1 - Estilo Bootstrap Cerulean

Imágenes

Algunas imágenes incluidas en el proyecto, como lo son el logo de Arbit que se muestra en todas las páginas del sitio y los íconos de semáforos y cámaras IP que se muestran en el mapa como marcadores, fueron de creación propia utilizando la herramienta Adobe Photoshop CS5 bajo el sistema operativo Windows 10.
Otras Imágenes fueron extraídas gratuitamente de diferentes sitios:

- Imagen ícono del sitio web: www.favicon.cc (Ver *Figura 14.1*)
- Imagen de fondo en Login de usuario: www. pixabay.com

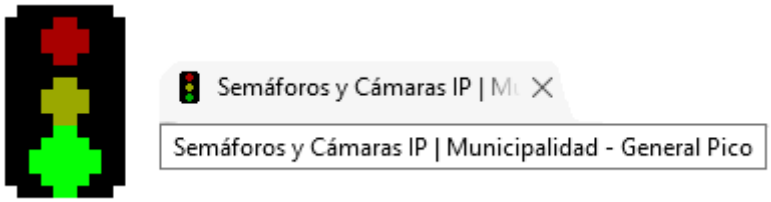


Figura 14.1 – Ícono del Sitio

Diseño *Responsive*

Aunque contar con un diseño *responsive* no fue un requerimiento funcional, fueron adaptadas ciertas funcionalidades a que si lo sea. Es decir, ciertas funcionalidades del sistema, como lo son el listado, la carga y edición de semáforos y cámaras IP, el login de usuarios y la vista operador, pueden observarse correctamente desde cualquier navegador y dispositivo, tales como celulares, netbooks, noteboks, entre otros.

Organización de la gestión de usuarios

En el sitio no es posible la registración de usuarios externos, solo se permite la configuración de usuarios desde el acceso al administrador. Esto se debe a que el sistema solo será utilizado por personal interno a la organización municipal. Por lo tanto, fueron precargados usuarios y grupos usuarios en torno a los requerimientos funcionales descriptos anteriormente. Los grupos de usuarios solo podrán ser cargados o modificados por la empresa Arbit ya que los accesos a las diferentes funcionalidades del sitio son regulados según el grupo de usuario al que pertenezcan los usuarios registrados. Los grupos de usuarios que fueron precargados según los requerimientos fueron:

- ✓ Personal Administrador: Acceso y edición de todo el contenido del sitio, con restricción de edición a grupos de usuarios.
- ✓ Personal Cámaras: Acceso sólo a vista de videos y vista operador con filtro de cámaras activado. Edición del registro de cámaras.
- ✓ Personal Semáforos: Acceso sólo a vista operador con filtro de semáforos activado. Edición del registro de semáforos.

Además, fueron precargados usuarios que se vinculan a tales grupos para la prueba de acceso y contenido.

HARDWARE DE IMPLANTACIÓN

Infraestructura

Infraestructura de Intersección

En cada intersección de la ciudad donde están colocados semáforos y cámaras IP para la regulación del tránsito, la empresa Arbit incorporará dos nuevos dispositivos con el fin de proporcionar las herramientas adecuadas para ejecutar correctamente las funciones que provee el sitio web. Los dos dispositivos que serán incorporados en cada intersección son:

- Router o Enrutador: dispositivo que proporciona conectividad a nivel de red cuya función principal consiste en enviar o encaminar paquetes de datos de una red a otra. En este caso, servirá para poder acceder de manera remota a los dispositivos vinculados al mismo en la intersección, es decir, tanto al dispositivo creado por la empresa que informará sobre el estado de cada semáforo como a las diferentes cámaras IP.
- Controlador de Semáforo: Este dispositivo, creado por la empresa Arbit, tendrá varias funcionalidades. Por un lado, se encargará de tomar como entrada la señal proveniente de las lámparas de cada semáforo y acondicionar la misma, es decir, reducir los 220 Volts de tensión de las lámparas hasta los niveles adecuados de tensión que consumirá el microprocesador encargado de procesar la señal. Por otro lado, para lograr medir la corriente que pasa por cada lámpara del semáforo incluirá un sensor de corriente. El prototipo realizado originalmente para simulación, será incorporado con las mismas características -exceptuando el reemplazo de pulsadores por sensores de corriente- dentro del dispositivo para detectar el estado de los semáforos vinculados en cada intersección.

En la *Figura 15.1* pueden observarse cada uno de los elementos que estarán vinculados en una intersección. El ejemplo describe una intersección de cuatro vías que cuenta con un semáforo y una cámara IP por vía. La conexión establecida entre los diferentes dispositivos de la intersección será de tipo Ethernet.

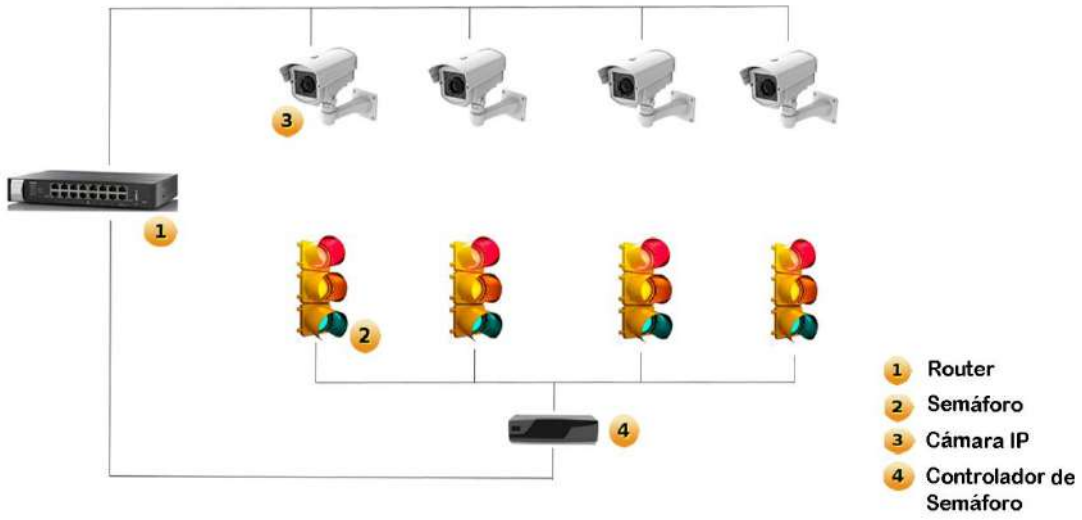


Figura 15.1 - Ejemplo Infraestructura de Intersección

Arquitecturas

Arquitectura Cliente-Servidor

El servidor generado en la máquina virtual será trasladado, también de forma virtual, a un servidor de propiedad municipal. Este último será particionado en varios servidores virtuales privados (VPS), donde uno de ellos es el que contendrá el sitio web.

Como ya ha sido especificado, el servidor es el encargado de procesar toda petición realizada por un cliente. Todo procesamiento se realiza desde la máquina servidora, por lo que podría decirse que se trata de una arquitectura cliente-servidor de tipo ‘servidor pesado, cliente liviano’ donde el cliente se emplea sólo para el nivel de presentación (utilizando un navegador web). De esta manera, puede lograrse una mayor seguridad frente a intentos de accesos indebidos.

Arquitectura de Red

Una empresa, encargada de proveer servicios de internet en la ciudad, será designada para otorgar un rango de IP que pueda ser utilizado con el fin de identificar tanto a las cámaras como a los controladores de semáforos y al servidor que analizará y gestionará los mismos.

Cada cámara IP y Controlador de semáforo será identificado por una IP, que será definida como privada, ya que se designará una red privada para los dispositivos a los cuales solo el servidor tendrá acceso. Por lo tanto, el servidor tendrá asignada, por un lado, una IP privada para la comunicación con los dispositivos y por otro, una IP pública para posibilitar a los usuarios registrados el acceso a su servicio – sitio web- de manera remota desde cualquier parte del mundo.

La arquitectura fue definida de esa manera con el fin de evitar conocimiento público de la red a la cual pertenecen los equipos. Creando así, un nexo entre los dispositivos y el sistema web al fin de resguardar la seguridad de los mismos.

Las IP designadas a cada dispositivo deben ser IP fijas ya que el sistema funciona correctamente con configuración de IP estática. Este requerimiento tiene un coste más elevado que la asignación dinámica de IP. Por lo tanto, si perjudica abruptamente el presupuesto establecido, las IP deberán ser designadas a los equipos de manera dinámica. Esto significará utilizar un servidor DHCP para la asignación dinámica de IP. Cada equipo deberá reportar al sistema el cambio de dirección para que logre ser identificado. De esta manera, deben realizarse pequeños cambios o adaptaciones en el sistema a fin de que cada dirección IP sea actualizada automáticamente.

En la *Figura 16.1* se puede identificar, a grandes rasgos, cómo es la comunicación entre los equipos, el servidor y los clientes.

Figura 16.1 – Arquitectura de Red

Conclusión:

Los requerimientos solicitados por el municipio han sido satisfechos en su completitud con la elaboración de este proyecto. Aunque hasta el momento el sistema no haya sido implantado, la simulación presentada dejó en evidencia la conformidad de los administrativos municipales.

Las horas previstas estimadas para cada actividad fueron acertadas, pudiendo culminar las mismas en el tiempo designado para su realización.

Ha sido posible obtener un producto escalable tanto en desarrollo como en ejecución. En la ejecución del sistema no sólo pueden controlarse equipos a nivel local o municipal, como es este caso, sino que también es posible hacerlo a nivel provincial, nacional e internacional. En el desarrollo pueden incorporarse fácilmente mejoras en cuanto a especificación de fallos y seguridad ya que son dos ítems relevantes en el proyecto y su escalabilidad fue diseñada y prevista desde un comienzo.

Como trabajo futuro se pretende, además de mejorar lo realizado, incorporar nuevas funcionalidades tales como informes de estados de los equipos en rangos definidos de tiempos, gráficos estadísticos, envíos de email cuya función sea la de informar cambios en el sistema como por ejemplo informar sobre el fallo de un equipo o la puesta en marcha nuevamente del mismo, entre otras propuestas.

Por último, gracias a la realización del proyecto fue posible conocer herramientas y lenguajes de programación que no habían sido tratados hasta el momento en el período universitario, pudiendo así contar con mayor conocimiento y experiencia para encarar proyectos futuros en el ámbito laboral.

Bibliografía:

- Date C.: Introducción a los Sistemas de Base de Datos, 7ta Edición, 2001
- Pressman R.: Ingeniería de Software: Un Enfoque Práctico, 6ta Edición, 2006
- Holovaty, Adrián y Kaplan-Moss, Jacob: El libro de Django, 2008.
- Van Rossum, Guido: El tutorial de Python, Argentina, 2009.
- Django Software Foundation: Django Documentation: Publicación 1.8.x, 2015.
- Mohammed J. Kabir.: La biblia del Servidor Apache 2, 2012.
- Víctor Gabriel Valencia Alaix: Principios sobre semáforos, Medellín, 2000.