

Universidad Nacional de Córdoba
Facultad de Ciencias Exactas, Físicas y Naturales



PROYECTO INTEGRADOR

*Sistema Embebido para Control de Acceso
Inteligente*

-2015-

Autores:
Dominguez, Mariano Agustín
Juan, Leandro Gastón

Director:
Dr. Ing. Orlando Micolini

Co-Director:
Marcelo Cebollada

Sistema Embebido para Control de Acceso Inteligente

Dominguez, Mariano Agustín

Matrícula: 34908760

Cel: 0351-157157848

E-Mail: mardom4164@gmail.com

Juan, Leandro Gastón

Matrícula: 34087384

Cel: 0297-154356037

E-mail: leandrojuan87@gmail.com

Proyecto integrador como requisito para obtener el título de:
Ingeniero en Computación

Director:

Dr. Ing Orlando Micolini

Co-director:

Marcelo Cebollada

Índice General

Contenido

Índice de ilustraciones	8
Índice de tablas	10
Capítulo 1 - Introducción	11
1.1 Objetivo Principal	11
1.2 Objetivo Secundario	11
1.3 Motivación.....	11
1.4 Resumen.....	12
1.5 Ante Proyecto.....	12
1.5.1 Introducción.....	12
1.5.2 Requerimientos Funcionales	12
1.5.3 Requerimientos No Funcionales.....	13
1.5.4 Trazabilidad de requerimientos funcionales	14
1.5.5 Trazabilidad de requerimientos no funcionales.....	14
1.5.6 Análisis de riesgos de requerimientos.....	15
Capítulo 2 - Marco Teórico	17
2.1 Introducción a los controles de acceso	17
2.1.1 Forma de identificación de una persona.....	17
2.1.2 Nivel de Autonomía	19
2.1.3 Por Alimentación	19
2.1.4 Tipo de Comunicación	20
2.2 Sistemas Embebidos.....	20
2.2.1 Raspberry Pi.....	20
2.2.2 Cubieboard	25
2.2.3 Arduino	25
2.3 Sistemas operativos en base a Linux para Sistemas Embebidos	26
2.3.1 Raspbian	26
2.3.2 Pidora.....	27
2.3.3 Arch Linux ARM	27

2.3.4 OpenELEC.....	28
2.3.5 Raspbmc	28
2.4 Sistemas de Base de datos	29
2.4.1 SQLite.....	29
2.4.2 MySql	30
2.4.3 PostgreSQL.....	31
2.5 Lenguajes de Programación	32
2.5.1 Introducción.....	32
2.5.2 Lenguaje C.....	32
2.5.3 Python.....	32
2.5.4 Java	33
2.6 Servidores Web	33
2.6.1 Introducción.....	33
2.6.2 Servidor Web Apache	34
2.6.3 Servidor Web NGINX	34
2.6.4 Servidor Web Django.....	34
2.7 Servidores de Comunicación.....	37
2.7.1 Introducción.....	37
2.7.2 Servidor Asterisk.....	37
2.7.3 Servidor 3CX	40
2.7.4 Servidor Elastix	40
2.8 Sistema de Vigilancia.....	41
2.8.1 Sistema de vigilancia – MOTION	41
2.8.2 Sistemas de Vigilancia-Mpeg-Streamer.....	42
2.9 Softphone	42
2.9.1 Twinkle.....	43
2.9.2 Linephone	45
2.10 Comunicación Web en tiempo real – JANUS	46
2.10.1 Plugins.....	47
2.11 Servidor para compartir archivos – SAMBA.....	49
2.11.1 Arquitectura Samba.....	49
Capítulo 3 - Estudio del problema.....	51

3.1 Introducción	51
3.2 Arquitectura del sistema	51
3.2.1 Control de acceso	51
3.2.2 Sistemas embebidos.....	53
3.2.3 Sistemas operativos.....	54
3.2.4 Sistemas de bases de datos	54
3.2.5 Lenguajes de programación	55
3.2.6 Servidores web	56
3.2.7 Servidor de comunicaciones.....	57
3.2.8 Sistema de vigilancia.....	58
3.2.9 Softphone	59
3.2.10 Comunicación web en tiempo real.....	59
3.3 Metodología de trabajo	60
3.4 Iteraciones del proyecto	61
Capítulo 4 –Diseño, implementación y resultados	62
4.1 Iteración 1	62
4.1.1 Objetivos.....	62
4.1.2 Diseño	62
4.1.3 Implementación.....	64
4.1.4 Testing	81
4.1.5 Conclusión	84
4.2 Iteración 2	85
4.2.1 Objetivos.....	85
4.2.2 Diseño	85
4.2.3 Implementación.....	86
4.2.4 Testing	88
4.2.5 Conclusión	92
4.3 Iteración 3	93
4.3.1 Objetivos.....	93
4.3.2 Diseño	93
4.3.3 Implementación.....	94
4.3.4 Testing	161

4.3.5 Conclusión	164
4.4 Iteración 4	164
4.4.1 Objetivos.....	164
4.4.2 Diseño	164
4.4.3 Implementación.....	165
4.4.4 Testing	179
4.4.5 Conclusión	182
4.5 Iteración 5	183
4.5.1 Objetivos.....	183
4.5.2 Diseño	183
4.5.3 Implementación.....	184
4.5.4 Testing	189
4.5.5 Conclusión	192
4.6 Hardware del sistema	192
4.6.1 Objetivos.....	192
4.6.1 Circuito reductor de 12v a 5v	192
4.6.2 MAX 232	193
4.6.3 Alimentación electrocerradura	194
4.6.4 Esquemático final	195
Capítulo 5 – Testing Integral del sistema	197
5.1 Cuadro de análisis	197
5.2 Evidencia	198
5.2.1 Intento de ingreso de usuario externo.....	198
5.2.2 Visualizacion de cámara web en tiempo real.....	200
5.2.3 Comunicación de voz via VoIP	200
5.2.4Apertura remota de puerta	202
Capítulo 6 - Conclusiones y trabajos futuros.....	203
6.1 Conclusión	203
6.2 Trabajos futuros	204
Bibliografía	205
ANEXOS.....	208
A - Configuración Motion	208

Índice de ilustraciones

Ilustración 1: Placa Raspberry Pi	21
Ilustración 2: Pines GPIO de Raspberry Pi.....	24
Ilustración 3: Diagrama de secuencia del sistema de acceso	64
Ilustración 4: Win32DiskManager.....	65
Ilustración 5: Interfaz gráfica Raspbian	67
Ilustración 6: SQLite Manager.....	73
Ilustración 7: Evidencia lectura de tarjeta RFID	82
Ilustración 8: Evidencia acceso a usuarios permitido/denegado	84
Ilustración 9: Diagrama de secuencia del sistema de acceso con vigilancia	86
Ilustración 10: Evidencia visualización de cámara web en tiempo real	90
Ilustración 11: Evidencia captura y guardado de imágenes	92
Ilustración 12: Diagrama de secuencia del sistema de registro y captura	94
Ilustración 13: Inicio del servidor Django	96
Ilustración 14: Interfaz web Django I.....	97
Ilustración 15: Interfaz web Django II	105
Ilustración 16: Login de la interfaz administrativa de Django	105
Ilustración 17: Interfaz administrativa de Django	106
Ilustración 18: Sincronización con la base de datos	107
Ilustración 19: Sección login de página web.....	120
Ilustración 20: Sección home de página web	124
Ilustración 21: Sección editar usuarios de página web	128
Ilustración 22: Sección ver usuarios de página web	131
Ilustración 23: Sección agregar usuarios de página web.....	136
Ilustración 24: Sección ver eventos de página web	140
Ilustración 25: Sección eventos de usuarios de página web	141
Ilustración 26: Sección eventos de accesos no permitidos de página web.....	143
Ilustración 27: Sección eventos web de página web	146
Ilustración 28: Sección eventos de usuarios web de página web	146
Ilustración 29: Sección capturas de imagen de página web.....	150
Ilustración 30: Sección franjas horarias de página web	153
Ilustración 31: Sección agregar franjas horarias de página web	156
Ilustración 32: Sección editar franjas horarias de página web	156
Ilustración 33: Sección agregar categorías de página web.....	159
Ilustración 34: Sección ayuda de página web	160
Ilustración 35: Evidencia registro de usuario y captura de clave mediante página web	163
Ilustración 36: Diagrama de secuencia del sistema de comunicación y acceso por web	165
Ilustración 37: Configuración Asterisk terminada	166
Ilustración 38: Compilación Asterisk terminada I.....	167
Ilustración 39: Compilación Asterisk terminada II	167

Ilustración 40: Asterisk Checkconfig	169
Ilustración 41: Registro de usuario I	170
Ilustración 42: Registro de usuario II	171
Ilustración 43: Registro de usuario III	171
Ilustración 44: Registro de usuario IV	172
Ilustración 45: Diagrama de secuencia del sistema de seguridad con aviso de fallos	184
Ilustración 46: Evidencia de levantar servicios caídos y aviso de fallas vía email	191
Ilustración 47: Esquemático del circuito reductor de 12V a 5V	193
Ilustración 48: Esquemático del circuito del MAX232	194
Ilustración 49: Esquemático del circuito de la electrocerradura.....	195
Ilustración 50: Esquemático del circuito general.....	196
Ilustración 51: Evidencia intento de ingreso de usuario externo	198
Ilustración 52: Evidencia visualización de cámara web en tiempo real	200
Ilustración 53: Evidencia apertura remota de puerta	202

Índice de tablas

Tabla 1: Requerimientos funcionales del sistema	13
Tabla 2: Requerimientos no funcionales del sistema	14
Tabla 3: Matriz de trazabilidad de requerimientos funcionales.....	14
Tabla 4: Matriz de trazabilidad de requerimientos no funcionales.....	15
Tabla 5: Referencias para análisis de riesgos.....	15
Tabla 6: Análisis de riesgos de requerimientos	16
Tabla 7: Especificaciones técnicas de la placa Raspberry Pi.....	23
Tabla 8: Medidas de valor.....	51
Tabla 9: Comparación control de acceso.....	52
Tabla 10: Comparación sistemas embebidos	53
Tabla 11: Comparación sistemas operativos	54
Tabla 12: Comparación sistemas de bases de datos	55
Tabla 13: Comparación lenguajes de programación	56
Tabla 14: Comparación servidores web.....	56
Tabla 15: Comparación servidor de comunicaciones	57
Tabla 16: Comparación sistemas de vigilancia	58
Tabla 17: Comparación softphones	59
Tabla 18: Iteraciones del proyecto	62
Tabla 19: Requerimientos involucrados en el sistema de acceso	62
Tabla 20: Cuadro de análisis de lectura de tarjeta RFID	81
Tabla 21: Cuadro de análisis de acceso a usuarios permitido/denegado	83
Tabla 22: Requerimientos involucrados en el sistema de acceso con vigilancia.....	85
Tabla 23: Cuadro de análisis de visualización de cámara web en tiempo real	89
Tabla 24: Cuadro de análisis de captura y almacenamiento de imágenes.....	91
Tabla 25: Requerimientos involucrados en el sistema de registro y captura de claves.....	93
Tabla 26: Cuadro de análisis del registro de usuarios y captura de clave mediante página web	161
Tabla 27: Requerimientos involucrados en el sistema de comunicación y acceso por web.....	164
Tabla 28: Cuadro de análisis de visualización en tiempo real y comunicación de voz vía VoIP.....	180
Tabla 29: Requerimientos involucrados en el sistema de seguridad con aviso de fallos.....	183
Tabla 30: Cuadro de análisis de levantar servicios caídos y aviso de falla vía email	189
Tabla 31: Cuadro de análisis de testing integral del sistema.....	198

Capítulo 1 - Introducción

1.1 Objetivo Principal

El objetivo principal del proyecto integrador es la creación e instalación de un sistema embebido de control de acceso inteligente para el Laboratorio de Arquitectura de Computadoras de la Facultad de ciencias exactas, físicas y naturales (FCEFYN) de la Universidad Nacional de Córdoba.

1.2 Objetivo Secundario

El sistema debe tener las funcionalidades de: controlar el acceso a un usuario, vigilancia mediante una cámara web, un servicio para poder realizar una comunicación de voz entre un usuario interno y otro externo, una opción para poder dar acceso al establecimiento remotamente, cargar usuarios y administrar el sistema vía web.

1.3 Motivación

Para este proyecto integrador decidimos desarrollar un sistema integral que abarque control de acceso, vigilancia y otras funcionalidades fundamentales, de bajo costo, tamaño pequeño y fácil de ocultar.

En la actualidad, existen muchos dispositivos de hardware capaces de realizar diversas tareas. Dichos dispositivos, conocidos como Sistemas Embebidos, son pequeñas computadoras potentes capaces de integrar distintos tipos de funcionalidades al mismo tiempo. Es por eso que en nuestro sistema, el control de acceso y vigilancia están fuertemente integrados en un único hardware.

En el mercado se encuentran presentes diversos sistemas para controlar acceso y muchos otros dispositivos para realizar tareas de vigilancia. Estas características son difíciles de integrar. La mayoría de los sistemas de control de acceso tienen solamente la funcionalidad de permitir o prohibir ingreso a los usuarios. De ser necesario contar con un sistema con muchas funcionalidades, éstos pueden ser muy costosos y por lo general suelen ser sistemas difíciles de ocultar debido a su gran tamaño.

Los fundamentos mencionados anteriormente son la principal motivación de nuestro Proyecto Integrador.

1.4 Resumen

En el capítulo 1 analizamos el problema en cuestión, partiendo de los requerimientos para el desarrollo del mismo y finalizando con un diagrama de análisis de riesgos de los requerimientos.

El capítulo 2 es el marco teórico de nuestro proyecto. Mostramos información sobre las herramientas existentes y necesarias para lograr nuestro objetivo.

El capítulo 3 contiene el estudio del problema, las comparaciones y elecciones sobre las herramientas mencionadas, la metodología de trabajo que empleamos y la división del proyecto en iteraciones.

El capítulo 4 contiene el diseño, implementación y testing del sistema, es decir, se realizan los pertinentes diagramas para comprender como son abordados los requerimientos, se muestra paso a paso como se realiza el proyecto, desde la instalación y configuración del software y hardware necesario, hasta la creación de scripts y software propio, focalizándose en la descripción de funcionalidades y la integración de las mismas. Finalizando, se analizan los resultados obtenidos de diferentes casos de prueba realizados.

En el capítulo 5 se encuentra el testing integral del sistema y en el capítulo 6 la conclusión sobre el desarrollo del proyecto.

Por ultimo está la bibliografía que fue consultada durante la creación del sistema y luego se da lugar a los anexos pertinentes al informe.

1.5 Ante Proyecto

1.5.1 Introducción

Para poder definir correctamente los requerimientos funcionales del proyecto se hicieron reiterativas consultas y reuniones con nuestros clientes, Dr. Ing. Orlando Micolini y Marcelo Cebollada. Además se consultó bibliografía especializada en seguridad como las revistas “SoftGuard” y “Negocio de Seguridad”. De esta forma se logró obtener un conjunto de ideas de lo que el sistema tiene que ser capaz de realizar.

Es por eso que en esta sección se realiza una descripción general del sistema. Se especifican los requerimientos funcionales y no funcionales, la relación existente entre requerimientos a través de matrices de trazabilidad y un análisis de riesgos para visualizar, de una manera más simple, los requerimientos más críticos a la hora de abordarlos.

1.5.2 Requerimientos Funcionales

Número	Prioridad	Requerimiento	Referencia
1	Alta	Se deberá poder visualizar el exterior del laboratorio mediante cámara web en tiempo real	

2	Alta	Deberá existir un sistema de control de ingreso con tarjetas por proximidad	
3	Alta	Al registrar evento de ingreso de usuario se deberá realizar una captura de imagen	
4	Alta	Las imágenes deben ser guardadas	
5	Alta	Software administrador deberá ser realizado en un entorno web	
6	Alta	Se deberán poder visualizar eventos de ingreso permitido y prohibido	
7	Alta	El Software administrador deberá ser capaz de abrir la puerta	
8	Alta	Deberá existir una comunicación de voz entre usuario externo e interno del laboratorio	
9	Alta	El software deberá poder administrar usuarios y claves	Softguard
10	Alta	Deberá existir un sistema de captura de clave de usuario	
11	Media	Ante fallos deberán existir avisos vía email	
12	Media	El sistema de ingreso deberá accionar un buzzer como aviso de ingreso habilitado	
13	Media	Ante cortes de energía el sistema deberá continuar funcionando	DVR 3104 H
14	Media	Se deberá poder programar franjas horarias en el sistema para generar eventos	Nzlinux RT
15	Media	Se deberán brindar reportes ante fallos	Flexnet
16	Media	El software administrador deberá ser accesible mediante dispositivos de telefonía celular	
17	Media	La visualización por parte de la cámara deberá ser posible también en ausencia de luz	
18	Baja	Se deberá realizar un sistema de borrado de imágenes para no sobrecargar la memoria	
19	Baja	Las tablas de eventos y usuarios deberán poderse exportar a archivos Microsoft Excel	
20	Baja	Deberán existir leds de control para visualizar el estado del sistema	
21	Media	Deberá existir un sistema que levante los servicios caídos automáticamente	

Tabla 1: Requerimientos funcionales del sistema

1.5.3 Requerimientos No Funcionales

Numero	Prioridad	Requerimiento	Referencia
1	Alta	Cumplir requerimientos de seguridad estándar ya establecidos	Softguard
2	Alta	Las bases de datos de ingreso de usuarios debe estar protegida	
3	Alta	Se debe guardar un registro de todas las claves de usuario otorgadas	
4	Alta	El tiempo de respuesta del sistema ante la proximidad de la tarjeta debe ser menor a 2 segundos	
5	Media	Comprimir lo máximo posible la imagen	

6	Media	Resolución de imagen suficiente para correcta visualización	
7	Media	El sistema deberá ser capaz de reconocer servicios caídos	NVR 6000
8	Media	La comunicación de audio con el exterior debe establecerse rápidamente	
9	Baja	Deberá existir documentación clara para solucionar fallas que puedan ocurrir	

Tabla 2: Requerimientos no funcionales del sistema

1.5.4 Trazabilidad de requerimientos funcionales

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	x		x	x			x			x				x	x	x					
2		x	x		x	x		x	x	x	x	x	x	x						x	
3		x	x			x													x		
4			x	x		x										x		x			
5				x	x	x	x	x	x				x		x		x	x			
6				x	x		x					x		x		x			x		
7					x	x				x				x			x				
8						x								x			x				
9						x	x					x		x			x				
10							x						x			x					
11								x		x										x	
12									x												
13										x		x									
14											x		x								
15												x						x	x		
16												x				x				x	
17													x								
18														x				x			
19															x						
20																x					
21																			x		

Tabla 3: Matriz de trazabilidad de requerimientos funcionales

1.5.5 Trazabilidad de requerimientos no funcionales

Matriz de trazabilidad de Requerimientos No Funcionales									
	1	2	3	4	5	6	7	8	9
1	x								
2		x	x						
3			x						

4				x					
5					x	x			
6						x			
7							x		
8								x	
9									x

Tabla 4: Matriz de trazabilidad de requerimientos no funcionales

1.5.6 Análisis de riesgos de requerimientos

Para realizar el análisis de riesgos se clasificaron los requerimientos de acuerdo a las siguientes referencias:

Impacto	Prioridad
1 Crítico	1 Muy Alta
2 Normal	2 Alta
3 Marginal	3 Media
4 Despreciable	4 Baja

Tabla 5: Referencias para análisis de riesgos

Luego los requerimientos quedaron clasificados de la siguiente manera:

Iteraciones	Requerimientos	Prioridad	Impacto
Iteración 1	Deberá existir un sistema de control de ingreso con tarjetas por proximidad	1	1
	Deberá existir un sistema de captura de clave de usuario	1	1
	Se deberán poder visualizar eventos de ingreso permitido y prohibido	1	3
	Se deberá poder administrar usuarios y claves	1	1
	El sistema de ingreso deberá accionar un buzzer como aviso de ingreso habilitado	3	4
Iteración 2	Se deberá poder visualizar el exterior del laboratorio mediante cámara web en tiempo real	1	2
	Al registrar evento de ingreso de usuario se deberá realizar una captura de imagen	1	2
	Las imágenes deben ser guardadas	2	2
Iteración 3	Software administrador deberá ser realizado en un entorno web	3	3
	El Software administrador deberá ser capaz de abrir la puerta	3	4
	Se deberá poder programar franjas horarias en el sistema para	3	4

	generar eventos		
	El software administrador deberá ser accesible mediante dispositivos de telefonía celular	4	4
	Las tablas de eventos y usuarios deberán poderse exportar a archivos Microsoft Excel	4	4
Iteración 4	Deberá existir una comunicación de voz entre usuario externo e interno del laboratorio	3	4
Iteración 5	Ante fallos deberán existir avisos vía email	4	4
	Ante cortes de energía el sistema deberá continuar funcionando	2	3
	Se deberán brindar reportes ante fallos	4	4
	La visualización por parte de la cámara deberá ser posible también en ausencia de luz	3	3
	Se deberá realizar un sistema de borrado de imágenes para no sobrecargar la memoria	3	4
	Deberán existir leds de control para visualizar el estado del sistema	4	4
	Deberá existir un sistema que levante los servicios caídos automáticamente	2	3

Tabla 6: Análisis de riesgos de requerimientos

Capítulo 2 - Marco Teórico

2.1 Introducción a los controles de acceso

Una definición bien amplia de un control de acceso es el uso de un mecanismo en función de la identificación y autorización de una persona para poder acceder a datos o recursos. Básicamente encontramos un sistema de control de acceso en múltiples formas y para diversas aplicaciones. Por ejemplo, podemos encontrar un control de acceso en una notebook en donde se debe colocar una huella en un lector para poder hacer uso de la misma, o bien, se debe introducir una contraseña para poder acceder al correo electrónico. Estos controles de acceso permiten o deniegan el uso de datos, pero también podemos hacer uso de estos sistemas para controlar puertas, cerraduras o un torniquete.

Los sistemas de control de acceso se pueden clasificar por:

- Forma de identificación de una persona.
 - Acceso por huella dactilar
 - Acceso por clave en teclado
 - Acceso por proximidad de tarjetas de seguridad
- Nivel de Autonomía
 - Totalmente Autónomo
 - De Red
- Tipo de alimentación
 - Alimentación principal
 - Alimentación principal con batería integrada
- Tipo de Comunicación
 - Remota
 - Local

En nuestro caso, queremos emplear un control de acceso para permitir o denegar el acceso al Laboratorio de Arquitectura de Computadoras.

2.1.1 Forma de identificación de una persona

2.1.1.1 Acceso por huella dactilar

El sistema biométrico de huella dactilar nos permite realizar el control de acceso a un recurso, ejemplo una oficina, mediante el registro del patrón de la huella de una persona. Esto permite llevar un control de los empleados que entran y salen de la oficina.

Estos sistemas tienen la ventaja de que no se necesita ningún elemento extra para poder acceder al establecimiento, como por ejemplo una llave o una tarjeta. La desventaja que tiene es que son comúnmente deteriorados con intención por algunas personas.

2.1.1.2 Acceso por clave en teclado

Los teclados son una gran solución para incrementar la seguridad del sistema de control de accesos. El teclado permite la inserción del PIN que sólo conoce su propietario.

Habitualmente el uso del teclado se da para la identificación mediante PIN. Un **PIN** (**Personal Identification Number**) o Número de Identificación Personal, es un valor numérico usado para identificarse y poder tener acceso a ciertos sistemas o artefactos, como un teléfono móvil o un cajero automático. La mayoría de los **PIN** son 4 dígitos, dando lugar a 10.000 combinaciones. De esta manera un atacante tendría que realizar una media de 5.000 intentos para acertar con el PIN correcto.

Ideal para control de presencia, de asistencia y de horario en oficinas o empresas, posibilitando el control el horario de llegada o salida de los empleados, la gestión y contabilización de horas extras. Con el control de acceso a edificios de viviendas, en escuelas, institutos o universidades se consigue un acceso más cómodo y más seguro, sin necesidad de sobrecargarse de llaves.

2.1.1.3 Acceso por proximidad de tarjetas RFID

Un dispositivo de control de asistencia basado en tarjetas de proximidad, de ahí sus siglas **RFID** (**Radio Frequency Identification, Identificación por Radiofrecuencia**) permite el registro de las entradas/salidas del personal mediante la “aproximación” de tarjetas.

Una tarjeta de proximidad es una tarjeta plástica que lleva incrustada en su núcleo un circuito integrado y una antena de comunicación. Contiene un número de serie que es leído por frecuencia de radio, típicamente de 125kHz.

Las tarjetas de proximidad no necesitan ser leídas (como las magnéticas), solo es necesario acercar la tarjeta al radio de recepción de la antena. La distancia de lectura cambia dependiendo de la tecnología empleada, siendo desde unos centímetros para las tarjetas pasivas a unos metros para las activas.

Su principal ventaja es que el desgaste es mucho menor ya que no requiere contacto directo con el dispositivo de lectura.

Como se mencionó anteriormente existen dos tipos de tarjetas de proximidad:

- tarjetas activas, que incorporan una batería que permite incrementar el radio de lectura
- tarjetas pasivas que dependen del radio de recepción de la antena y por lo tanto no tienen límites de duración en el tiempo.

Otra típica clasificación de las tarjetas de proximidad se refiera a la frecuencia de funcionamiento. Las frecuencias empleadas habitualmente son:

- 125 KHz
- 13.56 Mhz
- 900 Mhz
- 2.4 Ghz

A diferencia de los códigos de barras, las tarjetas de proximidad no pueden ser clonadas ni imitadas. Asimismo, en vista que la lectura se realiza a nivel de frecuencias de radio, las tarjetas de proximidad no son afectadas por suciedad en el lector o en la tarjeta.

2.1.2 Nivel de Autonomía

2.1.2.1 Totalmente Autónomo

Los Controles de Acceso Autónomos son sistemas que permiten controlar una o más puertas, sin estar conectados a un PC o un sistema central, por lo tanto, no guardan registro de eventos. Aunque esta es la principal limitante, algunos controles de acceso autónomos tampoco pueden limitar el acceso por horarios o por grupos de puertas, esto depende de la robustez de la marca. Es decir, los más sencillos solo usan el método de identificación (ya sea clave, proximidad o biometría) como una "llave" electrónica.

2.1.2.2 De Red

Los Controles de Acceso en Red son sistemas que se integran a través de un PC local o remoto, donde se hace uso de un software de control que permite llevar un registro de todas las operaciones realizadas sobre el sistema con fecha, horario, autorización, etc. Van desde aplicaciones sencillas hasta sistemas muy complejos y sofisticados según se requiera.

2.1.3 Por Alimentación

2.1.3.1 Alimentación principal

El sistema se alimenta de una fuente de 12v conectada a la red principal de energía (220v). Ante un corte de energía, el sistema dejará de funcionar automáticamente.

2.1.3.2 Alimentación principal con batería integrada

El sistema se alimenta de una fuente de 12v conectada a la red principal de energía (220v). Además contiene una batería de 12v de respaldo para mantener funcionando el sistema ante cortes de energía.

2.1.4 Tipo de Comunicación

2.1.4.1 Remota

Es posible permitir o denegar acceso a un usuario externo remotamente mediante un navegador web. Dicha acción puede ser realizada desde una computadora, Tablet o dispositivo de telefonía móvil. A su vez, desde dichos dispositivos, es posible lograr una comunicación de voz vía Ip con entre los usuarios.

2.1.4.2 Local

El sistema permite o deniega el acceso independientemente de la intervención de otro usuario. Es decir, para lograr el acceso es necesario de contar, únicamente, con algún mecanismo de identificación personal.

2.2 Sistemas Embebidos

Un Sistema Embebido es un sistema de computación diseñado para realizar pocas funciones dedicadas. Se caracterizan por tener un pequeño tamaño y por consumir poca potencia.

A gran diferencia de los sistemas de propósito general (como ser una computadora personal o PC), los sistemas embebidos están diseñados para cubrir una necesidad específica. Algunos ejemplos de sistemas embebidos podrían ser dispositivos como un taxímetro, un sistema de control de acceso, la electrónica que controla una máquina expendedora o el sistema de control de una fotocopiadora entre otras múltiples aplicaciones.

Actualmente, gracias al avance tecnológico, los sistemas embebidos son más potentes y son capaces de realizar más tareas complejas.

2.2.1 Raspberry Pi

Raspberry Pi es la placa elegida para utilizar en el sistema, por lo cual decidimos colocar más información al respecto. La comparativa entre las diferentes placas y la decisión tomada se encuentra presente en el estudio del problema (Capítulo 3) de este informe.

El presente marco teórico está fundado en la bibliografía de referencia (1), (2) y (3).

2.2.1.1 Descripción general

Raspberry Pi es un ordenador de placa reducida (SBC) de bajo coste desarrollado en Reino Unido por la Fundación Raspberry Pi, con el objetivo de estimular la enseñanza de ciencias de la computación en las escuelas.

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), un procesador gráfico (GPU) VideoCore IV, y 512 MB de. El diseño no incluye un disco duro o una unidad de estado sólido, ya que usa una tarjeta SD para el almacenamiento permanente; tampoco incluye fuente de alimentación o carcasa.

La fundación da soporte para las descargas de las distribuciones para arquitectura ARM, Raspbian (derivada de Debian), RISC OS 5, Arch Linux ARM (derivado de Arch Linux) y Pidora (derivado de Fedora); y promueve principalmente el aprendizaje del lenguaje de programación Python, y otros lenguajes como Tiny BASIC, C y Perl.

Raspberry Pi usa mayoritariamente sistemas operativos basados en el núcleo Linux. Raspbian, una distribución derivada de Debian que está optimizada para el hardware de Raspberry Pi, se lanzó durante julio de 2012 y es la distribución elegida para el proyecto.

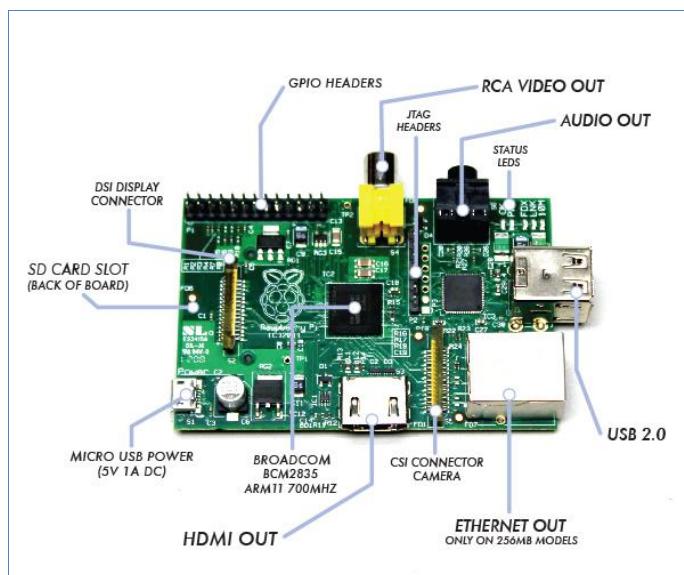


Ilustración 1: Placa Raspberry Pi

2.2.1.2 Especificaciones Técnicas

	Modelo A	Modelo B
SoC:	Broadcom BCM2835 (CPU + GPU + DSP + SDRAM + puerto USB)	
CPU:	ARM1176JZF-S a 700 MHz (familia ARM11)	
GPU:	Broadcom VideoCore IV, , OpenGL ES 2.0, MPEG-2 y VC-1 (con licencia),1080p30 H.264/MPEG-4 AVC	
Memoria (SDRAM):	256 MB (compartidos con la GPU)	512 MB (compartidos con la GPU) ⁴ desde el 15 de octubre de 2012
Puertos USB 2.0:	1	2 (vía hub USB integrado)
Entradas de vídeo:	Conector MIPI CSI que permite instalar un módulo de cámara desarrollado por la RPF	
Salidas de vídeo:	Conector RCA (PAL y NTSC), HDMI (rev1.3 y 1.4),Interfaz DSI para panel LCD	
Salidas de audio:	Conector de 3.5 mm, HDMI	
Almacenamiento integrado:	SD / MMC / ranura para SDIO	
Conectividad de red:	Ninguna	10/100 Ethernet (RJ-45) via hub USB ⁵

Periféricos de bajo nivel:	8 x GPIO, SPI, I ² C, UART	
Reloj en tiempo real:	Ninguno	
Consumo energético:	500 mA, (2.5 W)	700 mA, (3.5 W)
Fuente de alimentación:	5 V vía Micro USB o GPIO header	
Dimensiones:	85.60mm × 53.98mm (3.370 × 2.125 inch)	
Sistemas operativos soportados:	GNU/Linux: Debian (Raspbian), Fedora (Pidora), Arch Linux (Arch Linux ARM), Slackware Linux. RISC OS	

Tabla 7: Especificaciones técnicas de la placa Raspberry Pi

2.2.1.3 Pines GPIO

La tarjeta Raspberry Pi puede comunicarse con dispositivos externos mediante el conector GPIO incorporado. En dicho conector se integran pines de alimentación (+5 y +3.3 V), masa, y entradas/salidas capaces de implementar diferentes protocolos.

Sin entrar en detalles de protocolos, las dos posibles versiones de nuestra Raspberry hacen que nos encontremos a su vez con dos posibles escenarios hardware. Vemos aquí un esquema comparativo de la numeración de pines.

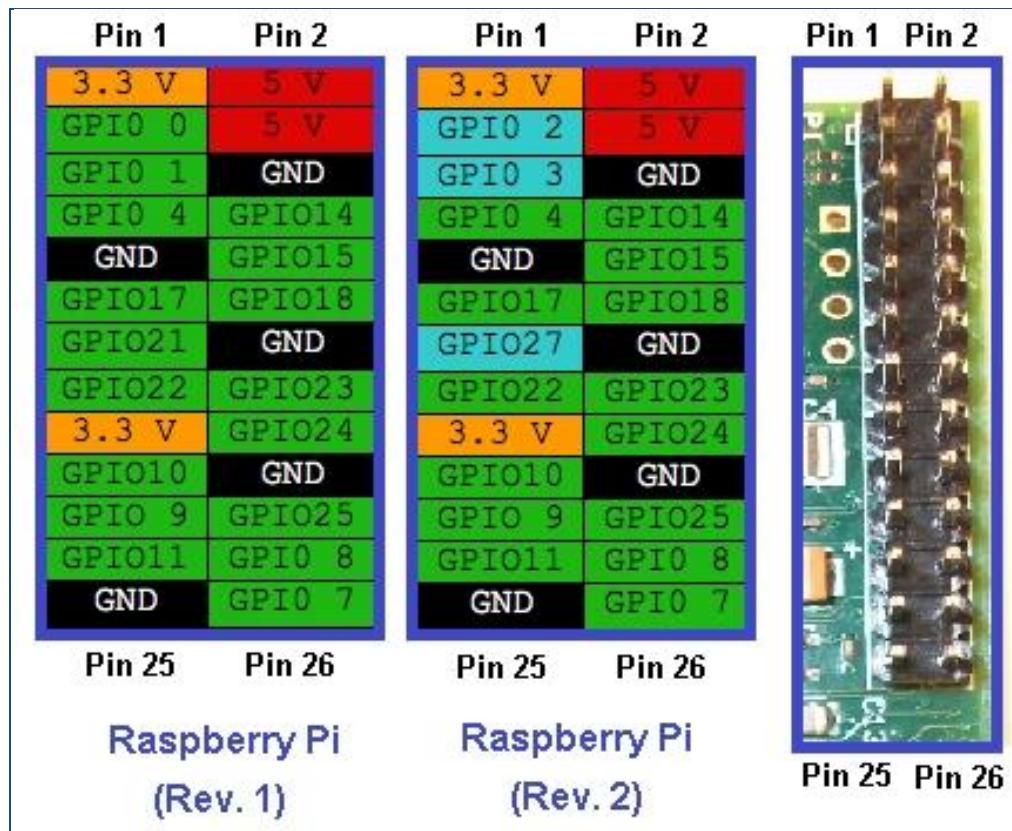


Ilustración 2: Pines GPIO de Raspberry Pi

Es importante tener en cuenta que, a nivel software, tenemos que saber con qué pin queremos comunicarnos. Además de los pines correspondientes a +5, +3.3 y masa, tenemos varios pines de uso genérico donde podemos conectar dispositivos hardware.

Es muy importante comentar que cualquier manipulación errónea, conexiónada equivocada o descarga estática sobre los pines GPIO puede dañarlos de forma permanente.

Nuestro Control de Acceso está implementado en un sistema embebido llamado Raspberry pi, en donde en el capítulo 3 justificamos su elección.

2.2.2 Cubieboard

El presente marco teórico está fundado en la bibliografía de referencia (4).

2.2.2.1 Descripción general

Cubieboard es una pequeña (10x6cm), extensible, de bajo costo y potente placa ARM con Allwinner A10 SoC. Cuenta con un 1 GHz ARM Cortex A8 CPU , con 1 GB de memoria RAM DDR3, un muy potente codificador/decodificador de video VPU, un GPU OpenGL ES Mali400 , y 96 cabeceras de expansión que pueden ser usadas como GPIO, I2C, UART, LVDS, PWM, SPI, CSI, y más.

2.2.2.2 Especificaciones de Hardware

1. AllWinnerTech SOC A10, ARM® Cortex™-A8 ARM® Mali400 MP1 cumple con OpenGL ES 2.0/1.1
2. 1GB DDR3 @480MHz
3. 4GB NAND flash interna, mas de 32GB en slot SD, mas de 2T en disco 2.5 SATA
4. Entrada 5VDC 2A o entrada USB otg
5. Ethernet 1x 10/100, soporte wifi usb
6. 2x USB 2.0 HOST, 1x mini USB 2.0 OTG, 1x micro sd
7. 1x HDMI 1080P salida de display
8. 1x IR, 1x en linea, 1x fuera de linea
9. Interfaz extendida de 96 pins, incluyendo I2C, SPI, RGB/LVDS, CSI/TS, FM-IN, ADC, CVBS, VGA, SPDIF-OUT, R-TP, y mas

2.2.3 Arduino

El presente marco teórico está fundado en la bibliografía de referencia (5).

Arduino es una plataforma de hardware libre, basada en una placa con un microcontrolador y un entorno de desarrollo, diseñada para facilitar el uso de la electrónica en proyectos multidisciplinares.

El hardware consiste en una placa con un microcontrolador Atmel AVR y puertos de entrada/salida.

Los microcontroladores más usados son el Atmega168, Atmega328, Atmega1280, y Atmega8 por su sencillez y bajo coste que permiten el desarrollo de múltiples diseños.

Por otro lado el software consiste en un entorno de desarrollo que implementa el lenguaje de programación Processing/Wiring y el cargador de arranque que es ejecutado en la placa.

Desde octubre de 2012, Arduino se usa también con microcontroladoras CortexM3 de ARM de 32 bits, 5 que coexistirán con las más limitadas, pero también económicas AVR de 8 bits. ARM y AVR no son plataformas compatibles a nivel binario, pero se pueden programar con el mismo IDE de

Arduino y hacerse programas que compilen sin cambios en las dos plataformas. Eso sí, las microcontroladoras CortexM3 usan 3,3V, a diferencia de la mayoría de las placas con AVR, que generalmente usan 5V. Sin embargo, ya anteriormente se lanzaron placas Arduino con Atmel AVR a 3,3V como la Arduino Fio y existen compatibles de Arduino Nano y Pro como Meduino en que se puede conmutar el voltaje.

Arduino se puede utilizar para desarrollar objetos interactivos autónomos o puede ser conectado a software tal como Adobe Flash, Processing, Max/MSP, Pure Data. Las placas se pueden montar a mano o adquirirse. El entorno de desarrollo integradolibre se puede descargar gratuitamente.

Arduino puede tomar información del entorno a través de sus entradas analógicas y digitales, puede controlar luces, motores y otros actuadores. El microcontrolador en la placa Arduino se programa mediante el lenguaje de programación Arduino (basado en Wiring) y el entorno de desarrollo Arduino (basado en Processing). Los proyectos hechos con Arduino pueden ejecutarse sin necesidad de conectar a un computador. También cuenta con su propio software que se puede descargar de su página oficial que ya incluye los drivers de todas las tarjetas disponibles lo que hace más fácil la carga de códigos desde el computador. El proyecto Arduino recibió una mención honorífica en la categoría de Comunidades Digital en el Prix Ars Electrónica de 2006.

2.3 Sistemas operativos en base a Linux para Sistemas Embebidos

En algunos Sistemas embebidos es posible introducirle un sistema operativo, comúnmente en base a Linux, para poder hacer uso eficiente del mismo. La elección del mismo se basa según a las características que se desee, por ejemplo si el sistema embebido será utilizado como servidor, como una computadora normal o como un media center entre otros.

A continuación mostraremos algunos de los más conocidos y utilizados actualmente.

2.3.1 Raspbian

El presente marco teórico está fundado en la bibliografía de referencia (6).

Raspbian es una distribución del sistema operativo GNU/Linux y por lo tanto libre basado en Debian Wheezy (Debian 7.0) para la placa computadora (SBC) Raspberry Pi, orientado a la enseñanza de informática.

Técnicamente el sistema operativo es un port no oficial de Debian Wheezy armhf para el procesador (CPU) de Raspberry Pi, con soporte optimizado para cálculos en coma flotante por hardware, lo que permite dar más rendimiento dependiendo del caso. El port fue necesario al no haber versión Debian Wheezy armhf para la CPU ARMv6 que contiene el Raspberry Pi.

La distribución usa LXDE como escritorio y Midori como navegador web. Además contiene herramientas de desarrollo como IDLE para el lenguaje de programación Python o Scratch, y diferentes ejemplos de juegos usando los módulos Pygame.

Destaca también el menú "raspi-config" que permite configurar el sistema operativo sin tener que modificar archivos de configuración manualmente. Entre sus funciones, permite expandir la partición root para que ocupe toda la tarjeta de memoria, configurar el teclado, aplicar overclock, etc.

2.3.2 Pidora

El presente marco teórico está fundado en la bibliografía de referencia (7).

Esta distribución, al igual que Raspbian, ha sido compilada y preparada especialmente para la Raspberry Pi, pero con el toque Fedora y el escritorio Xfce en lugar del LXDE.

Al igual que la distribución Raspbian de Debian, Pidora se ha compilado específicamente para aprovechar el hardware que ya está integrado en la Raspberry Pi.

Pidora ofrece un par de interesantes pequeños extras a su experiencia estándar de escritorio en Fedora. La reducción de empuje gráfico de la RPi provoca que el escritorio de GNOME se sustituya por el XFCE, más ligero. Pidora también ofrece un modo fácil de utilizar para los que no disponen de un monitor. Si se conectan los altavoces a la RPi, una voz dirá amablemente su dirección IP. Un truco inteligente para los que se conectan por ssh.

Pidora contiene una serie de módulos Raspberry Pi específicos de Python y bibliotecas nativas, como WiringPi, bcm2835 y python-rpi.gpio. El núcleo también está compilado para exponer las interfaces Raspberry Pi como I2C, SPI, serie y GPIO, y desde varias de ellas se puede acceder a las interfaces de archivo / sys (incluso de bash) sin el uso de las bibliotecas o módulos especiales. Además, Pidora contiene utilidades Raspberry y bibliotecas específicas para el acceso a la Broadcom VideoCore IV GPU Pi.

2.3.3 Arch Linux ARM

El presente marco teórico está fundado en la bibliografía de referencia (8).

Arch Linux ARM es una distribución de Linux para computadores ARM. Lleva adelante la filosofía de Arch Linux de mantener la simplicidad y el centrismo para el usuario, apuntando a usuarios de Linux competentes dándoles el completo control y la responsabilidad sobre el sistema.

Se proveen instrucciones para asistir a la navegación e instalación en las variadas plataformas ARM, sin embargo, el sistema en si ofrece muy poca asistencia para el usuario.

La distribución entera está en un ciclo de reléase constante que puede ser actualizado diariamente a través de pequeños paquetes en vez de grandes actualizaciones en una fecha determinada. La mayoría de los paquetes no son modificados de cuando el desarrollador los subió originalmente.

Arch Linux provee una estructura base liviana que permite al usuario dar forma al sistema de acuerdo a sus necesidades.

2.3.4 OpenELEC

El presente marco teórico está fundado en la bibliografía de referencia (9).

OpenELEC (abreviación de Open Embedded Linux Entertainment Center) es una distribución Linux diseñada para HTPCs y basada en el reproductor de medios XBMC.

OpenELEC proporciona un completo conjunto de aplicaciones de software para centro multimedia que incorpora una versión pre configurada de XBMC y add ons de terceros con emuladores de consolas de videojuegos de estilo retro y plug-ins para DVR. Es una distribución de Linux extremadamente ligera y de arranque muy rápido diseñada principalmente para arrancar desde tarjetas de memoria Flash, tales como CompactFlash o desde SSD; del mismo modo que la distribución XBMCbuntu(anteriormente conocida como XBMC Live) pero especialmente orientada a dispositivos set top box con especificaciones de hardware mínimas basadas en SoCs ARM o en gráficos y procesadores Intel x86.

La utilidad te permite reproducir fotografías, música, vídeos y cualquier contenido multimedia que tengas alojado en tu ordenador o en un dispositivo externo. Y todo lo puedes manejar desde su única interfaz.

Su instalación es sencilla, pero requiere de algunos pasos. El centro multimedia lo puedes instalar como un sistema operativo independiente o una configuración de inicio múltiple (soporta plug and play, discos duros externos, dispositivos USB, tarjetas de memoria, etc), o bien lo puedes almacenar en modo local.

Características principales:

- Es independiente y auto gestionable (trabajo fuera de tu sistema operativo original)
- Se actualiza a través de tu red inalámbrica o por OTA.
- Detecta dispositivos de audio o videos de forma automática (conectados al ordenador).
- Puedes incorporar plugins para ampliar los códec de audio y vídeo y sus prestaciones.
- Puedes personalizar su diseño gracias a sus Skin.
- Soporta subtítulos y multi idioma en las películas.
- Permite la administración remota a través de SSH.
- Lo puedes proteger mediante contraseña.

2.3.5 Raspbmc

El presente marco teórico está fundado en la bibliografía de referencia (10).

Raspbmc no es más que una distro de Linux muy ligera diseñada especialmente para funcionar en Raspberry Pi y que está basada en la distro XBMC en su versión 12, llamada "Frodo", por lo que estamos hablando de la posibilidad de convertir a este pequeñísimo computador de bajo costo en el centro multimedia perfecto.

Cuenta con una interfaz diseñada para el entretenimiento y streaming de contenido audiovisual directamente desde cualquier otro ordenador o dispositivo.

Además, gracias a estar basado en XBMC 12 cuenta con soporte de decodificación de audio DTS a través del software, u en el caso de hardware este ofrece una mayor variedad de soporte hasta algunos como MPGE-2 y VC1, por ejemplo.

Las posibilidades de Raspbmc no están limitadas a simplemente reproducir contenido vía streaming o tener una interfaz cómoda de manejar con un mando de control, sino que puede reproducir vídeo en alta definición (1080p), reproducir música desde la tarjeta SD y hasta ver televisión en vivo, instalando el programa necesario.

2.4 Sistemas de Base de datos

El presente marco teórico está fundado en la bibliografía de referencia (11) y (12).

Una base de datos es un conjunto de datos pertenecientes a un mismo contexto y almacenados sistemáticamente para su posterior uso. En este sentido; una biblioteca puede considerarse una base de datos compuesta en su mayoría por documentos y textos impresos en papel e indexados para su consulta. Actualmente la mayoría de las bases de datos están en formato digital, siendo este un componente electrónico, y por ende se ha desarrollado y se ofrece un amplio rango de soluciones al problema del almacenamiento de datos.

Existen programas denominados sistemas gestores de bases de datos, abreviado DBMS, que permiten almacenar y posteriormente acceder a los datos de forma rápida y estructurada. Las propiedades de estos DBMS, así como su utilización y administración, se estudian dentro del ámbito de la informática.

Nosotros necesitamos hacer uso de estos sistemas ya que necesitamos acceder y almacenar los datos relacionados a los usuarios, claves, registros, eventos, etc. Dichos datos es necesario y convenientes que estén organizados y ordenados, para luego ser accedidos de una forma más eficientes. A continuación describiremos algunos de los Motores de Base de datos que existen en la actualidad.

SQLite es el Sistema de Gestión de Base de Datos (SGBD) elegido para utilizar en el proyecto, por lo cual decidimos colocar más información al respecto. La comparativa entre los diferentes SGBD y la decisión tomada se encuentra presente en el estudio del problema (Capítulo 3) de este informe.

2.4.1 SQLite

SQLite es un sistema de gestión de bases de datos relacional compatible con ACID, contenida en una relativamente pequeña (~275 kB) biblioteca escrita en C.

A diferencia de los sistemas de gestión de bases de datos cliente-servidor, el motor de SQLite no es un proceso independiente con el que el programa principal se comunica. En lugar de eso, la biblioteca SQLite se enlaza con el programa pasando a ser parte integral del mismo. El programa utiliza la funcionalidad de SQLite a través de llamadas simples a subrutinas y funciones. Esto reduce la latencia en el acceso a la base de datos, debido a que las llamadas a funciones son más eficientes que la comunicación entre procesos. El conjunto de la base de datos (definiciones, tablas, índices, y los propios datos), son guardados como un sólo fichero estándar en la máquina host. Este diseño simple se logra bloqueando todo el fichero de base de datos al principio de cada transacción.

En su versión 3, SQLite permite bases de datos de hasta 2 Terabytes de tamaño, y también permite la inclusión de campos tipo BLOB

2.4.1.1 Características

La biblioteca implementa la mayor parte del estándar SQL-92, incluyendo transacciones de base de datos atómicas, consistencia de base de datos, aislamiento, y durabilidad (ACID), triggers y la mayor parte de las consultas complejas.

SQLite usa un sistema de tipos inusual. En lugar de asignar un tipo a una columna como en la mayor parte de los sistemas de bases de datos SQL, los tipos se asignan a los valores individuales. Por ejemplo, se puede insertar un string en una columna de tipo entero (a pesar de que SQLite tratará en primera instancia de convertir la cadena en un entero). Algunos usuarios consideran esto como una innovación que hace que la base de datos sea mucho más útil, sobre todo al ser utilizada desde un lenguaje de scripting de tipos dinámicos. Otros usuarios lo ven como un gran inconveniente, ya que la técnica no es portable a otras bases de datos SQL. SQLite no trataba de transformar los datos al tipo de la columna hasta la versión 3.

Varios procesos o hilos pueden acceder a la misma base de datos sin problemas. Varios accesos de lectura pueden ser servidos en paralelo. Un acceso de escritura sólo puede ser servido si no se está sirviendo ningún otro acceso concurrentemente. En caso contrario, el acceso de escritura falla devolviendo un código de error (o puede automáticamente reintentarse hasta que expira un tiempo de expiración configurable). Esta situación de acceso concurrente podría cambiar cuando se está trabajando con tablas temporales. Sin embargo, podría producirse un interbloqueo debido al multihilo.

2.4.2 MySql

El presente marco teórico está fundado en la bibliografía de referencia (13) y (14).

MySQL es un sistema de gestión de bases de datos relacional, multihilo y multiusuario. Está escrito en C y C++ y emplea el lenguaje SQL para consultas a la base de datos.

MySQL es muy utilizado en aplicaciones web, como Drupal o phpBB, en plataformas (Linux/Windows-Apache-MySQL-PHP/Perl/Python), y por herramientas de seguimiento de errores como Bugzilla. Su popularidad como aplicación web está muy ligada a PHP, que a menudo aparece en combinación con MySQL.

MySQL es una base de datos muy rápida en la lectura cuando utiliza el motor no transaccional MyISAM, pero puede provocar problemas de integridad en entornos de alta concurrencia en la modificación. En aplicaciones web hay baja concurrencia en la modificación de datos y en cambio el entorno es intensivo en lectura de datos, lo que hace a MySQL ideal para este tipo de aplicaciones. Sea cual sea el entorno en el que va a utilizar MySQL, es importante monitorizar de antemano el rendimiento para detectar y corregir errores tanto de SQL como de programación.

2.4.3 PostgreSQL

El presente marco teórico está fundado en la bibliografía de referencia (15).

PostgreSQL es un Sistema de gestión de bases de datos relacional orientado a objetos. Es libre y está publicado bajo la licencia BSD.

Como muchos otros proyectos de código abierto, el desarrollo de PostgreSQL no es manejado por una empresa y/o persona, sino que es dirigido por una comunidad de desarrolladores que trabajan de una forma desinteresada, altruista, libre y/o apoyados por organizaciones comerciales. Dicha comunidad es denominada el PGDG (PostgreSQL Global Development Group).

Mediante un sistema denominado MVCC (Acceso concurrente multiversión, por sus siglas en inglés) PostgreSQL permite que mientras un proceso escribe en una tabla, otros accedan a la misma tabla sin necesidad de bloqueos. Cada usuario obtiene una visión consistente de lo último a lo que se le hizo commit. Esta estrategia es superior al uso de bloqueos por tabla o por filas común en otras bases, eliminando la necesidad del uso de bloqueos explícitos.

2.5 Lenguajes de Programación

2.5.1 Introducción

Como mencionamos anteriormente. Estos sistemas embebidos tienen un sistema operativo en base a Linux. Por ende es compatible con muchos de los lenguajes de programación existentes. A continuación vamos a describir algunos de los lenguajes más importantes y conocidos.

2.5.2 Lenguaje C

El presente marco teórico está fundado en la bibliografía de referencia (16).

Se trata de un lenguaje de tipos de datos estáticos, débilmente tipificado, de medio nivel pero con muchas características de bajo nivel. Dispone de las estructuras típicas de los lenguajes de alto nivel pero, a su vez, dispone de construcciones del lenguaje que permiten un control a muy bajo nivel. Los compiladores suelen ofrecer extensiones al lenguaje que posibilitan mezclar código en ensamblador con código C o acceder directamente a memoria o dispositivos periféricos.

La primera estandarización del lenguaje C fue en ANSI, con el estándar X3.159-1989. El lenguaje que define este estándar fue conocido vulgarmente como ANSI C. Posteriormente, en 1990, fue ratificado como estándar ISO (ISO/IEC 9899:1990). La adopción de este estándar es muy amplia por lo que, si los programas creados lo siguen, el código es portable entre plataformas y/o arquitecturas.

2.5.3 Python

El presente marco teórico está fundado en la bibliografía de referencia (17).

Python es un lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis que favorezca un código legible.

Se trata de un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, usa tipado dinámico y es multiplataforma.

Una característica importante de Python es la resolución dinámica de nombres; es decir, lo que enlaza un método y un nombre de variable durante la ejecución del programa (también llamado enlace dinámico de métodos). Otro objetivo del diseño del lenguaje es la facilidad de extensión. Se pueden escribir nuevos módulos fácilmente en C o C++. Python puede incluirse en aplicaciones que necesitan una interfaz programable.

2.5.4 Java

El presente marco teórico está fundado en la bibliografía de referencia (18).

Java es un lenguaje de programación de propósito general, concurrente, orientado a objetos que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible. Su intención es permitir que los desarrolladores de aplicaciones escriban el programa una vez y lo ejecuten en cualquier dispositivo (conocido en inglés como WORA, o "write once, run anywhere"). Esto significa que el código que es ejecutado en una plataforma no tiene que ser recompilado para correr en otra. Java es, a partir de 2012, uno de los lenguajes de programación más populares en uso, particularmente para aplicaciones de cliente-servidor de web, con unos 10 millones de usuarios reportados.

2.6 Servidores Web

2.6.1 Introducción

Un servidor web o servidor HTTP es un programa informático que procesa una aplicación del lado del servidor, realizando conexiones bidireccionales y/o unidireccionales y síncronas o asíncronas con el cliente y generando o cediendo una respuesta en cualquier lenguaje o Aplicación del lado del cliente. El código recibido por el cliente suele ser compilado y ejecutado por un navegador web. Para la transmisión de todos estos datos suele utilizarse algún protocolo. Generalmente se usa el protocolo HTTP para estas comunicaciones, perteneciente a la capa de aplicación del modelo OSI. El término también se emplea para referirse al ordenador que ejecuta el programa.

Por otro lado, el entorno web hace referencia a un ambiente de desarrollo y/o ejecución programas o servicios en el marco de la web en general.

El entorno web es una forma de interfaz de usuario gráfico. Por ejemplo, para recibir email se puede utilizar una aplicación (como Outlook de Microsoft), pero también es muy usual emplear un “entorno web” para la recepción y envío de correos electrónicos, como el que ofrecen Gmail de Google o Hotmail de Microsoft.

Otro ejemplo, el aprendizaje de un curso puede hacerse de manera física entre los alumnos y un profesor, pero también puede realizarse en un “entorno web”, donde los alumnos puedan acceder a una serie de herramientas probablemente usando un navegador web en internet.

Existen herramientas, programas, lenguajes de programación y desarrollo que son específicos para el diseño de aplicaciones dentro de un entorno web. De hecho se cree que poco a poco las aplicaciones e incluso gran parte del sistema operativo irán migrando hacia un entorno web.

2.6.2 Servidor Web Apache

El presente marco teórico está fundado en la bibliografía de referencia (19).

El servidor HTTP Apache es un servidor web HTTP de código abierto, para plataformas Unix (BSD, GNU/Linux, etc.), Microsoft Windows, Macintosh y otras, que implementa el protocolo HTTP/1.1 y la noción de sitio virtual. Cuando comenzó su desarrollo en 1995 se basó inicialmente en código del popular NCSA HTTPd 1.3, pero más tarde fue reescrito por completo.

Apache es usado principalmente para enviar páginas web estáticas y dinámicas en la World Wide Web. Muchas aplicaciones web están diseñadas asumiendo como ambiente de implantación a Apache, o que utilizarán características propias de este servidor web.

Apache es el componente de servidor web en la popular plataforma de aplicaciones LAMP, junto a MySQL y los lenguajes de programación PHP/Perl/Python.

2.6.3 Servidor Web NGINX

El presente marco teórico está fundado en la bibliografía de referencia (20).

Nginx (pronunciado en inglés “engine X”) es un servidor web/proxy inverso ligero de alto rendimiento y un proxy para protocolos de correo electrónico (IMAP/POP3).

Es software libre y de código abierto, licenciado bajo la Licencia BSD simplificada. Es multiplataforma, por lo que corre en sistemas tipo Unix (GNU/Linux, BSD, Solaris, Mac OS X, etc.) y Windows.

Las principales características es que es un servidor de archivos estáticos, índices y autoindexado. Utiliza un proxy inverso con opciones de caché. Posee balanceo de carga y tiene tolerancia a fallos. Además tiene soporte de HTTP sobre SSL.

2.6.4 Servidor Web Django

Django es el Servidor web elegido para utilizar en el proyecto, por lo cual decidimos colocar más información al respecto. La comparativa entre los diferentes servidores web y la decisión tomada se encuentra presente en el estudio del problema (Capítulo 3) de este informe.

El presente marco teórico está fundado en la bibliografía de referencia (21) y (22).

Django es un framework de desarrollo web de código abierto, escrito en Python, que respeta el patrón de diseño conocido como Modelo–vista–controlador.

La meta fundamental de Django es facilitar la creación de sitios web complejos. Django pone énfasis en el re-uso, la conectividad y extensibilidad de componentes, el desarrollo rápido y el principio “No te repitas” (DRY, del inglés Don't Repeat Yourself). Python es usado en todas las partes del framework, incluso en configuraciones, archivos, y en los modelos de datos.

Este Framework de python es el que se utilizó para este proyecto. En la etapa de implementación aclararemos sus fundamentos.

2.6.4.1 Características

Los orígenes de Django en la administración de páginas de noticias son evidentes en su diseño, ya que proporciona una serie de características que facilitan el desarrollo rápido de páginas orientadas a contenidos.

En lugar de requerir que los desarrolladores escriban controladores y vistas para las áreas de administración de la página, Django proporciona una aplicación incorporada para administrar los contenidos, que puede incluirse como parte de cualquier página hecha con Django y que puede administrar varias páginas hechas con Django a partir de una misma instalación; la aplicación administrativa permite la creación, actualización y eliminación de objetos de contenido, llevando un registro de todas las acciones realizadas sobre cada uno, y proporciona una interfaz para administrar los usuarios y los grupos de usuarios (incluyendo una asignación detallada de permisos).

La distribución principal de Django también aglutina aplicaciones que proporcionan un sistema de comentarios, herramientas para sindicar contenido vía RSS y/o Atom, "páginas planas" que permiten gestionar páginas de contenido sin necesidad de escribir controladores o vistas para esas páginas, y un sistema de redirección de URLs.

Otras características de Django son:

- Un mapeador objeto-relacional.
- Aplicaciones "enchufables" que pueden instalarse en cualquier página gestionada con Django.
- Una API de base de datos robusta.
- Un sistema incorporado de "vistas genéricas" que ahorra tener que escribir la lógica de ciertas tareas comunes.
- Un sistema extensible de plantillas basado en etiquetas, con herencia de plantillas.
- Un despachador de URLs basado en expresiones regulares.
- Un sistema "middleware" para desarrollar características adicionales; por ejemplo, la distribución principal de Django incluye componentes middleware que proporcionan cacheo, compresión de la salida, normalización de URLs, protección CSRF y soporte de sesiones.
- Soporte de internacionalización, incluyendo traducciones incorporadas de la interfaz de administración.

- Documentación incorporada accesible a través de la aplicación administrativa (incluyendo documentación generada automáticamente de los modelos y las bibliotecas de plantillas añadidas por las aplicaciones).

3.7.1.1 Arquitectura

Aunque Django está fuertemente inspirado en la filosofía de desarrollo Modelo Vista Controlador, sus desarrolladores declaran públicamente que no se sienten especialmente atados a observar estrictamente ningún paradigma particular, y en cambio prefieren hacer "lo que les parece correcto". Como resultado, por ejemplo, lo que se llamaría "controlador" en un "verdadero" framework MVC se llama en Django "vista", y lo que se llamaría "vista" se llama "plantilla".

Gracias al poder de las capas mediator y foundation, Django permite que los desarrolladores se dediquen a construir los objetos Entity y la lógica de presentación y control para ellos.

- Presentación

Aquí se maneja la interacción entre el usuario y el computador. En Django, ésta tarea la realizan el template engine y el template loader que toman la información y la presentan al usuario (vía HTML, por ejemplo). El sistema de configuración de URLs es también parte de la capa de presentación...

- Control

En esta capa reside el programa o la lógica de aplicación en sí. En Django son representados por las views y manipulators. La capa de presentación depende de ésta y a su vez ésta lo hace de la capa de dominio.

- Mediator

Es el encargado de manejar la interacción entre el subsistema Entity y foundation. Aquí se realiza el mapeo objeto-relacional a cargo del motor de Django.

- Entity

El subsistema entity maneja los objetos de negocio. El mapeo objeto-relacional de Django permite escribir objetos de tipo entity de una forma fácil y estándar.

- Foundation

La principal tarea del subsistema foundation es la de manejar a bajo nivel el trabajo con la base de datos. Se provee soporte a nivel de foundation para varias bases de datos y otras están en etapa de prueba.

3.7.1.2 Soporte de Bases de Datos

Respecto a la base de datos, la recomendada es PostgreSQL, pero también son soportadas MySQL y SQLite 3. Se encuentra en desarrollo un adaptador para Microsoft SQL Server. Una vez creados los data models, Django proporciona una abstracción de la base de datos a través de su API que permite crear, recuperar, actualizar y borrar objetos. También es posible que el usuario ejecute sus propias consultas SQL directamente. En el modelo de datos de Django, una clase representa un registro de una tabla en la base de datos y las instancias de esta serán las filas en la tabla.

2.7 Servidores de Comunicación

2.7.1 Introducción

Para poder establecer una comunicación de voz entre usuarios externos e internos de un establecimiento es necesario contar con una plataforma de comunicaciones que permita el traspaso de paquetes de voz. Actualmente la central de comunicaciones más completa que existe se denomina Asterisk y contiene infinidad de funcionalidades que nos van a permitir poder establecer la comunicación mediante VoIP. También existen otras que son pagas tal como 3CX.

2.7.2 Servidor Asterisk

Asterisk es el Servidor de comunicaciones elegido para utilizar en el proyecto, por lo cual decidimos colocar más información al respecto. La comparativa entre los diferentes servidores de comunicaciones y la decisión tomada se encuentra presente en el estudio del problema (Capítulo 3) de este informe.

El presente marco teórico está fundado en la bibliografía de referencia (23), (24) y (25).

Asterisk es el mayor proyecto de software libre diseñado para la integración y unificación de los sistemas de comunicaciones conocidos

Originalmente fue concebido como una plataforma para la generación de un sistema PBX, pero con el tiempo ha ido evolucionando a otro tipo de usos, como Pasarelas VoIP, sistemas integrales para call-centers, salas de conferencias, buzones de voz, y todo tipo de aplicaciones que tengan relación con las comunicaciones en tiempo real.

Comparativamente Asterisk es para el mundo de las comunicaciones lo mismo que sería Apache para el mundo de las aplicaciones web. Apache es un servidor web, y Asterisk es un servidor de comunicaciones.

Asterisk es una plataforma de comunicaciones basada en la filosofía Open Source. Es capaz de convertir un ordenador común en un completo servidor de comunicaciones.

Considerando Asterisk como una plataforma integral de comunicaciones, podría considerarse la más importante, y ha resultado como única por muchos años en un entorno, donde todos los sistemas de comunicación eran totalmente privativos. Aunque con el tiempo, fueron sacando interfaces comúnmente conocidos como CTI para la integración de sistemas de terceros para cumplir funciones muy específicas, la potencia de estas interfaces era bastante limitada dado que el núcleo de los sistemas privados, permanecía cerrado al público.

2.7.2.1 Funcionalidades

El sistema asterisk incorpora todas las funcionalidades que pueden esperarse en una central convencional y asimismo muchísimas funcionalidades avanzadas que tendrían un elevado coste en sistemas tradicionales propietarios. A continuación enumeramos sólo las más importantes:

Funciones básicas:

- **Transferencias (directa o consultiva):** Permite transferir una llamada en curso a otra extensión. Existen dos formas:
 - Transferencia atendida: consultando al nuevo destino si quiere que le pasen la llamada,
 - Transferencia directa: pasando la llamada sin consultar al destinatario.

En versiones anteriores a la 1.8, al transferir una llamada se perdía el CLID (el número del usuario llamante). Esto no ocurre con la versión 1.8, que mantiene por tanto el CLID tras una transferencia.

- **Desvíos:** Permiten la transferencia automática de una llamada entrante hacia un número determinado (interno o externo) cuando se cumplen determinadas condiciones: por ejemplo si el número está ocupado, si no contesta, etc.
- **Capturas (de grupo o de extensión):** La captura permite coger una llamada que se está recibiendo en una extensión desde otra distinta.
 - Captura de extensión: por defecto se hace con el código *8 + la extensión.
 - Captura de grupo: se predefinen unos determinados grupos de extensiones de modo que al marcar un código de asterisk - por defecto el *8 - se coge cualquier llamada que esté recibiendo el grupo en el que estamos.
- **Conferencia múltiple:** En función del modelo de terminal se podrá establecer una comunicación entre múltiples usuarios de la centralita.
- **Llamada directa a extensión:** Si además del número de cabecera disponemos de diferentes números públicos (DDIs), podremos enrutar directamente la llamada entrante a uno de estos DDIs, a una extensión de la centralita.
- **Ring groups: grupos de llamadas.** Una llamada entrante podrá ser dirigida directamente a un ring group, que es un grupo de extensiones que sonarán de acuerdo a una determinada estrategia previamente establecida. Si la llamada no se descuelga no podrá ser tratada posteriormente y se perderá.

- **DND (Do not disturb):** Opción de no molestar, que podrá ser configurado en Asterisk mediante un código o directamente en el terminal.

Funciones avanzadas:

- **Correo Vocal (*Voicemail*) integrado con correo electrónico.** En caso de que el usuario no pueda atender una llamada, se puede programar que se transfiera a un sistema de buzón de voz. En caso de que se deje un mensaje, se enviará un correo electrónico avisando del mismo al usuario destinatario.
- **Operadora Automática (IVR):** Una operadora automática o IVR es una aplicación de telefonía que permite interactuar con el usuario que realiza la llamada, de forma que éste pueda pulsar opciones previamente anunciadas y acceder de forma automática a los destinos programados.
- **Música en espera con archivos WAV:** Asterisk nos permite introducir categorías de música en espera basadas en archivos .wav y mp3. De este modo podremos poner diferentes melodías para ser reproducidas como música en espera.
- **Colas de Llamadas (ACD):** Un sistema de colas o ACD es una aplicación que distribuye las llamadas entrantes a un grupo específico de agentes de acuerdo a una determinada estrategia. Si la llamada no puede ser descolgada, no se pierde y puede ser transferida a otro destino.
- **Salas de Audio-Conferencias:** Una sala de audio conferencias es un sistema que permite conectar a múltiples usuarios en una misma conversación telefónica. Los usuarios pueden acceder a la sala desde una extensión interna, o bien desde el exterior (a través de un número directo o bien a través de un IVR). Es un sistema muy útil para hacer reuniones internas (por ejemplo seguimiento de ventas) o bien con clientes o proveedores. No hay una capacidad máxima de salas por lo que podremos definir tantas como nos hagan falta (la limitación principal serán los recursos del servidor). Las salas tampoco tienen una capacidad máxima de llamadas por lo que principalmente la limitación vendrá dada por el número de líneas que la empresa disponga para salir a la PSTN.
- **Gestión de llamadas entrantes según horario o fecha (Time Conditions).** Con la incorporación del gestor web FreePBX en su versión 2.5 o superior, es muy sencillo definir un horario y calendario laboral que nos permita hacer un tratamiento diferenciado de las llamadas entrantes según el mismo. Por ejemplo, si estamos en horario laboral y no es festivo, la llamada se envía a la extensión 100 (de operadora). En caso contrario, se puede enviar a una locución que advierta de que son horas no laborables, o bien es un día festivo.
- **Extensiones DISA:** Es posible configurar opciones de post marcación para determinadas llamadas entrantes, de forma que una vez hemos comunicado con la centralita, podemos llamar a un nuevo destino de forma sencilla y automatizada.
- **Callback:** Llamada automática de respuesta a una llamada perdida. Cuando redirigimos una llamada al módulo de Callback el sistema lo que hará será colgar y originar una llamada hacia el número que nos ha llamado, de este modo se pueden centralizar costes de llamada. La llamada saldrá siguiendo las normas de routing saliente de llamadas.
- **Retrollamada:** funcionalidad disponible en la versión 1.8. Si se hace una llamada a una extensión y esta no contesta (por estar ocupado o ausente), se puede activar la función Asterisk de retrollamada. En cuanto el usuario llamado cuelgue, se avisa al que activo la función de retrollamada para que este pueda llamar de nuevo a la extensión inicial.

- **Informes detallados de llamadas (CDR):** Detalle de llamadas realizadas/recibidas por extensión, para imputación de costes departamentales, por cliente o incluso para facturación.
- **Integración CTI:** Integración de la telefonía con sistemas informatizados de gestión comercial o de atención al cliente (CRM). Estos sistemas permiten por ejemplo ejecutar una llamada desde el PC o bien recibir información sobre una llamada entrante en la pantalla.

2.7.3 Servidor 3CX

El presente marco teórico está fundado en la bibliografía de referencia (26).

3CX IP PBX es un software basado en las bondades de una Central Telefónica tradicional, se instala bajo Windows de Microsoft y fue creado por 3CX. Al igual que una PABX basada en hardware, el sistema permite las llamadas entre los teléfonos del sistema y las líneas telefónicas del exterior de la red de la telefonía pública (PSTN) mediante la tecnología de voz sobre IP (VoIP). La central IP 3CX es comparable a la PBX IP de Asterisk ya que son software que emulan la función de un sistema telefónico o PABX.

Si bien hay una edición gratuita de 3CX no es de código abierto o software GNU. 3CX IP PBX no sólo esta comercialmente soportada, sino que también es apoyada por una comunidad voluntaria de los usuarios, expertos de redes y telefonía a través de un foro de Internet.

El 3CX IP PBX es un sistema que se creó en una plataforma basadas en Microsoft Windows y no hay versión para Linux. Se basa exclusivamente en el protocolo SIP y el estándar no es compatible con protocolos como IAX2 (Inter-Asterisk eXchange protocol).

Las características esenciales que se encuentran en la 3CX son comparables a las funciones de una Central telefónica tradicional basada en hardware: Llamando de extensión a extensión, grupos de búsqueda de teléfonos, buzón de voz, respuesta interactiva de voz con menús etc.

Las organizaciones pueden recurrir a sus líneas telefónicas PSTN a través de un PSTN-Gateway de puerta de enlace IP (FXO) o pueden tener un sistema híbrido que utiliza la PSTN y servicios de VoIP telefónicos para aprovechar el costo y la disponibilidad.

2.7.4 Servidor Elastix

El presente marco teórico está fundado en la bibliografía de referencia (27).

Elastix es una distribución libre de Servidor de Comunicaciones Unificadas que integra en un solo paquete:

- VoIP PBX
- Fax
- Mensajería Instantánea

- Correo electrónico
- Colaboración

Elastix implementa gran parte de su funcionalidad sobre cuatro programas de software muy importantes como son Asterisk, Hylafax, Openfire y Postfix. Estos brindan las funciones de PBX, Fax, Mensajería Instantánea y Correo electrónico respectivamente. Elastix corre sobre CentOS como sistema operativo y actualmente su versión más estable es Elastix 2.4.0

2.8 Sistema de Vigilancia

El presente marco teórico está fundado en la bibliografía de referencia (28) y (29).

Con la palabra sistemas de vigilancia nos referimos a todo tipo de aparatos para la detección inmediata y sistemática, la visualización o vigilancia de un proceso con ayuda técnica, sensores u otros sistemas de vigilancia, como por ejemplo una cámara.

Un sistema de vigilancia no necesita ser caro ni difícil de instalar. Con los productos adecuados, solo se necesitan unos pocos componentes y pocas o ninguna herramienta.

En este proyecto necesitamos utilizar una cámara web para poder capturar imágenes del exterior del laboratorio y poder ver en tiempo real desde un navegador web. Es decir, necesitamos de un software capaz de capturar imágenes instantáneas y poder realizar un streaming de video.

Existen varias alternativas para lograr esto. Una de ellas consiste en usar la herramienta web mpeg-streamer. La otra es usando la herramienta Motion.

2.8.1 Sistema de vigilancia – MOTION

Motion es un programa que monitoriza la señal de vídeo proveniente de una cámara. Es una herramienta que nos permite tener como salida archivos jpeg, ppm, e incluso secuencias de vídeo mpeg. Además, nos permite visualizar múltiples webcams a la vez así como live streaming.

2.8.1.1 Funcionalidades

- Tomar capturas de imagen a partir de movimiento.
- Mirar múltiples dispositivos de video al mismo tiempo.
- Mirar múltiples entradas en una tarjeta de captura al mismo tiempo.
- Streaming por webcam en tiempo real.
- Creación de películas mpeg en tiempo real usando libavcodec de ffmpeg.
- Tomar capturas de imagen automáticas en intervalos regulares.
- Tomar capturas de imagen automáticas en intervalos irregulares usando cron.
- Envío de emails cuando se detecta movimiento.

- Envió de mensajes SMS cuando se detecta movimiento.
- Ejecutar comandos externos cuando se detecta movimiento.
- Seguimiento de movimiento.
- Comunica eventos con bases de datos MySQL o PostgreSQL.
- Interfaz web usando proyectos relacionados a motion como motion.cgi, Kenneths Webcam Package, Kevins Webpage, X-Motion y muchos más.
- Configuración de usuario y definición de usuarios en una pantalla (display).
- Control automático de umbral y ruido.
- Texto en imágenes altamente configurable.
- Definición de path y nombres de archivos de las películas e imágenes almacenadas altamente configurables.

2.8.2 Sistemas de Vigilancia-Mpeg-Streamer

El presente marco teórico está fundado en la bibliografía de referencia (30).

MJPEG-Streamer es una aplicación que se ejecuta por línea de comandos. A grandes rasgos, se encarga de obtener frames JPG (imágenes) capturadas desde una cámara compatible y transmitirlas como M-JPEG (secuencia de vídeo) mediante el protocolo HTTP para poder visualizarlo en navegadores, VLC y otras herramientas.

¿Cómo funciona?

Su funcionamiento se basa en unos plugins de entrada y salida. Es decir, un plugin (de entrada) copia las imágenes JPEG a un directorio de acceso global, mientras que otro plugin (de salida) procesa las imágenes, sirviéndolas como un simple fichero de imagen, o bien, emite las mismas de acuerdo a los estándares MPG existentes.

2.9 Softphone

El presente marco teórico está fundado en la bibliografía de referencia (31) y (32).

Además de un servidor de comunicaciones también es necesario contar con teléfonos IP para realizar la comunicación entre usuarios externos e internos. Aquí es donde aparecen estos dispositivos virtuales denominados Softphone.

Un softphone (en inglés combinación de software y de telephone) es un software que es utilizado para realizar llamadas a otros softphones o a otros teléfonos convencionales usando un VoIP (Voz sobre IP) o ToIP (Telefonía sobre IP).

Normalmente, un Softphone es parte de un entorno Voz sobre IP y puede estar basado en el estándar SIP/H.323 o ser privativo. Hay muchas implementaciones disponibles, como la ampliamente disponible Skype, Windows Messenger o NetMeeting de Microsoft .

Los Softphone funcionan bien con la mayoría de los ITSP - Proveedores de Servicios de Telefonía por Internet. Se puede conectar usando un teléfono USB o un enlace USB a un SoftPhone y obtener un servicio gratuito VoIP de teléfono a teléfono.

Los SoftPhone son realmente parte de un grupo tecnológico mayor, el CTI (Integración Computadora Telefonía).

Algunos softphones están implementados completamente en software, que se comunica con las PABX a través de la (LAN) Red de Área Local - TCP/IP para controlar y marcar a través del teléfono físico. Generalmente se hace a través de un entorno de centro de llamadas, para comunicarse desde un directorio de clientes o para recibir llamadas. En estos casos la información del cliente aparece en la pantalla de la computadora cuando el teléfono suena, dando a los agentes del centro de llamadas determinada información sobre quién está llamando y cómo recibirla y dirigirse a esa persona.

Existen muchísimos Softphones en la web que pueden ser descargados. Entre los más conocidos tenemos: Twinkle, LinePhone, X-lite, 3CX, Zoiper, etc. Prácticamente todos realizan las mismas funciones por lo que solo explicaremos dos de ellos.

2.9.1 Twinkle

Twinkle es el Softphone elegido para utilizar en el proyecto, por lo cual decidimos colocar más información al respecto. La comparativa entre los diferentes Softphones y la decisión tomada se encuentra presente en el estudio del problema (Capítulo 3) de este informe.

Twinkle es un softphone para voz sobre IP (VoIP) y de comunicación instantánea de mensajes utilizando el protocolo SIP. Se puede utilizar para la comunicación directa entre un teléfono IP y otro teléfono IP o en una red utilizando un proxy SIP para rutear las llamadas y mensajes.

Twinkle está disponible únicamente para el sistema operativo Linux (licencia GPL).

2.9.1.1 Funcionalidades

- 2 líneas de llamada
- Múltiples identidades de llamadas activas
- Ring tones customizables
- Llamada en espera
- Retención de llamada
- Llamada de conferencia de 3 vías
- Silenciador
- Redirección de llamadas en demanda
- Redirección de llamadas incondicionales
- Redirección de llamadas cuando se está ocupado
- Redirección de llamadas cuando no se contesta
- Rechazo de solicitud de redirección de llamadas
- Transferencias de llamadas a ciegas
- Transferencia de llamadas con consulta (transferencia asistida)
- Rechazo de solicitud de transferencia de llamadas
- Rechazo de llamadas

- Repetir última llamada
- Opción de “no molestar”
- Auto contestación
- Indicador de mensaje en espera
- Marcado rápido de buzón de voz
- Scripts definidos por el usuario accionados en eventos de llamadas
- Eventos RFC 2833 DTMF
- DTMF en banda
- DTMF fuera de banda (SIP INFO)
- Soporte STUN para NAT transversal
- Envió de paquetes NAT “keep alive” cuando se usa STUN
- NAT transversal a través de aprovisionamiento estático
- Conexiones TCP persistentes para NAT transversal
- Indicador de llamada perdida
- Historial de detalles de llamadas para llamadas entrantes, salientes, exitosas y perdidas
- Soporte DNS SRV
- Failover automático hacia un servidor alternativo si el servidor no está disponible
- Otros programas pueden originar llamadas SIP vía Twinkle
- Icono de bandeja de sistema
- Menú de bandeja de sistema para originar responder llamadas rápidamente mientras Twinkle está oculto
- Número de reglas de conversión definibles por el usuario
- Libro de direcciones simple
- Soporte para transporte UDP y TCP para SIP
- Presencia
- Mensajería instantánea
- Transferencia de archivos simple con mensajería instantánea
- Indicador de composición de mensaje instantáneo
- Interfaz por línea de comandos (CLI)

2.9.1.2 Seguridad VoIP

- Comunicación de voz segura por ZRTP/SRTP
- Soporte de autenticación MD5 para todas las solicitudes SIP
- Soporte de autenticación AKAv1-MD5 para todas las solicitudes SIP
- Ocultamiento de identidad

2.9.1.3 Códigos y drivers de audio

Twinkle soporta los siguientes códigos de audio:

- G.711 A-law (64 kbps payload, 8 kHz sampling rate)
- G.711 μ-law (64 kbps payload, 8 kHz sampling rate)
- GSM (13 kbps payload, 8 kHz sampling rate)
- Speex narrow band (15.2 kbps payload, 8 kHz sampling rate)
- Speex wide band (28 kbps payload, 16 kHz sampling rate)
- Speex ultra wide band (36 kbps payload, 32 kHz sampling rate)
- iLBC (13.3 or 15.2 kbps payload, 8 kHz sampling rate)
- G.726 (16, 24, 32 or 40 kbps payload, 8 kHz sampling rate)

Para todos los códigos están disponibles las siguientes opciones de pre procesamiento para mejorar la calidad en la llamada de extremo a extremo:

- Control de ganancia automática
- Reducción de ruido
- Detección de actividad de voz
- Control acústico de eco

Twinkle soporta también los siguientes drivers de audio:

- Open Sound System (OSS)
- Advanced Linux Sound Architecture (ALSA)

2.9.2 Linephone

El presente marco teórico está fundado en la bibliografía de referencia (33).

Linphone es una aplicación VoIP disponible en ordenadores con Linux, Windows,1 o equipos de Apple con Mac OS X, Android, y teléfonos móviles iPhone. Se utiliza el Session Initiation Protocol (SIP) para la comunicación y está disponible bajo licencia GPL. Linphone utiliza GTK + GUI para Linux y se puede también funcionar como una aplicación en modo consola. es compatible con la telefonía mediante el uso de un proveedor de servicios de telefonía por Internet (ITSP).

Para el desarrollo del sistema de control utilizamos el Softphone Twinkle, debido a que es un software muy liviano, fácil de instalar, y que contiene las funcionalidades necesarias (Interfaz de línea de comandos CLI, auto contestación, y otras).

2.10 Comunicación Web en tiempo real – JANUS

El presente marco teórico está fundado en la bibliografía de referencia (34) y (35).

WebRTC – que significa Web Real Time Communication – es básicamente un estándar abierto para embeber capacidades de comunicación multimedia en tiempo real directamente en el navegador web. El entorno abierto y estándar elimina la necesidad de software cliente, plugins y descargas. El esfuerzo de WebRTC está siendo estandarizado en el nivel de la API en la W3C, y en el nivel del protocolo en la IETF.

Facilita las aplicaciones de llamadas de voz, chat de video y compartimiento de archivos entre navegadores. El códec soportado actualmente para WebRTC es VP8. Utiliza un servidor denominado Servidor de Conferencias Web que en conjunto con un Servidor STUN es requerido para proveer la página inicial y sincronizar las conexiones entre dos nodos WebRTC.

WebRTC es un trabajo en progreso con implementaciones avanzadas en los navegadores Chrome y Firefox. Está soportado por Google Chrome, Mozilla Firefox y Opera. La razón por la que se creó WebRTC es atacar problemas de privacidad que aparecen al exponer capacidades y flujos locales.

Disruptive Analysis sugiere que para finales de 2016 los usuarios individuales de WebRTC alcanzarán los 1.000 millones y las PCs, teléfonos inteligentes y tabletas que soporten WebRTC llegarán a 4.000 millones.

Los principales componentes de WebRTC incluyen:

- getUserMedia, que permite a un navegador web acceder a la cámara y el micrófono
- PeerConnection, que establece las llamadas de audio / vídeo
- DataChannels, que permiten a los navegadores a compartir datos a través de peer-to-peer

Para realizar la comunicación existe una plataforma denominada Janus que sirve como pasarela WebRTC. Janus es un software open source yestá concebido para ser una pasarela de propósito general. No provee ninguna funcionalidad extra más que implementar medios para implementar una comunicación WebRTC con un navegador, intercambiando mensajes JSON en el, y retransmitiendo RTP/RTCP y mensajes entre navegadores y la lógica de aplicación del servidor a la que están unidos.

Cualquier funcionalidad/aplicación específica es provista por los plugins del lado del servidor, a los cuales el browser puede conectarse luego mediante un Gateway para sacar ventajas de la funcionalidad que proveen. Ejemplos de esos plugins pueden ser implementaciones o aplicaciones como Test de eco, puentes de conferencias, grabadores multimedia, o pasarelas SIP entre otros.

Para el sistema de control utilizamos el plugin “SIP Gateway”.

2.10.1 Plugins

- Echo Test

Este plugin sirve simplemente para devolver ciegamente cualquier envío que se realiza al mismo. Básicamente uno está unido a sí mismo, así como el audio y video que se envía al Gateway que vuelve a sí mismo como si se tratase de un “eco”.

También se proveen algunos controles para manipular los medios de comunicación antes de enviarlos. Se puede silenciar audio y video, por ejemplo, que será como decirle al Gateway que elimine las tramas y no devuelva eco.

A su vez se puede tratar de tapar el bitrate: este control le dirá al Gateway que manipule los paquetes RTCP REMB que pasan por allí, y así poder simular la limitación de ancho de banda.

- Streaming

Particularmente este plugin provee 3 aproximaciones diferentes de streaming:

1. Un stream “on-demand” originado por un archivo: diferentes usuarios accediendo a este stream recibirán una vista personal del mismo stream
2. Un stream “pseudo-live”, igualmente originado por un archivo: Todos los diferentes usuarios que acceden a este stream recibirán la misma vista compartida del mismo stream.
3. Un stream en vivo, originado por un script gstreamer: como en el stream “pseudo-live”, diferentes usuarios obtendrán el mismo stream.

Se pueden probar todos en la misma sesión.

- Video call

Es un simple plugin de video llamada para Janus, permitiendo dos peers WebRTC para llamarse mutuamente a través de la puerta de entrada (Gateway). La idea es proveer un servicio similar al ya conocido AppRTC demo, pero con la media fluyendo a través del Gateway en vez de ser una comunicación peer-to-peer

El plugin provee un mecanismo falso para registrarse. Un peer unido al plugin necesita especificar el nombre de usuario, que actúa como un “número de teléfono”: si el nombre de usuario es gratis, está asociado al peer, lo que permite que él o ella pueda ser llamado usando el nombre de usuario de otro peer. Los peers pueden llamar a otro peer especificando sus nombres de usuario, o esperando una llamada. La aproximación usada por este plugin es similar a una empleada por el test de eco: todas las tramas (RTP/RTCP) que vienen de un peer se retransmiten con el otro.

Como con el plugin de test de eco, existen knobs para controlar cuando el audio y/o video debe ser silenciado o no.

- SIP Gateway

Es un simple plugin SIP para Janus, permitiendo a los peers WebRTC que se registren en un servidor SIP (en este caso Asterisk) y llamar a usuarios SIP a través del gateway. Específicamente, cuando se une al plugin, se les solicita a los peers que provean sus credenciales de servidor SIP (la dirección

del servidor SIP y sus nombres de usuario). Esto logra que el plugin se registre en el servidor SIP y actúa como un cliente SIP en beneficio del peer web. La mayoría de los estados SIP están enmascarados por el plugin, y solo los eventos relevantes y funcionalidades están disponibles para el peer web: los peers pueden llamar a extensiones del servidor SIP o esperar por invitaciones entrantes, y durante una llamada pueden enviar tonos DTMF.

El concepto detrás de este plugin es permitir a diferentes páginas web asociadas al mismo peer unirse al plugin al mismo tiempo y sin embargo tener que registrarse solo una vez. Lo mismo se aplica a las llamadas: mientras una llamada debe ser notificada a todas las páginas web asociadas al peer, solo una será capaz de atenderla.

Este plugin fue el utilizado por el sistema de acceso para realizar la comunicación entre el usuario interno y el externo, utilizando la placa Raspberry Pi como servidor asterisk.

- Video MCU (videoconferencing)

Este es un plugin que implementa una video conferencia MCU para Janus. Esto significa que el plugin implementa una sala de conferencias virtual en donde los peers pueden unirse e irse en cualquier momento. Esta sala está basada en un patrón “Publish/Subscribe”. Cada peer puede publicar su propia conferencia de audio/video: esto genera que el stream esté disponible en la sala para que los otros participantes puedan unirse. Esto significa que el plugin permite la realización de diversos escenarios: desde una simple “webinar” (un speaker, muchos listeners), a una conferencia de video en donde todos los peers envían y reciben de y hacia todos los demás participantes).

- Audio conference

Este es un plugin que implementa un puente de audio conferencia para Janus, específicamente mezclando streams Opus. Esto significa que provee soporte en el SDP solo para Opus, y deshabilita el video. La codificación y decodificación Opus es implementada utilizando libopus. El plugin provee una API para permitir a los peers que se unan y dejen las salas de conferencias. Los peers pueden silenciar/activar sonido ellos mismos mandando mensajes específicos al plugin: Cuando un peer silencia/activa sonido, un evento es disparado a los otros participantes.

- Voice Mail

Este es un plugin que implementa un simple servicio de VoiceMail para janus, específicamente grabando streams Opus. Esto significa que provee soporte en el SDP solo para Opus, y deshabilita el video. Cuando un peer se contacta con el plugin, el plugin empieza a grabar tramas de audio que recibe y, luego de 10 segundos, cierra la conexión (peerConnection) y retorna a la URL del archivo a grabar.

Una vez que se retorna a una URL, el plugin permite que se configure donde se deben guardar las grabaciones (por ejemplo una carpeta en el servidor web) y el path base para usar cuando se retorna a URLs.

- Recorder/Playout

Esta es una aplicación simple que implementa dos diferentes funcionalidades: permite grabar un mensaje que es enviado con WebRTC en el formato definido en el archivo recorded.c y subsecuentemente repetir (replay) esa grabación también a través del WebRTC.

Esta aplicación apunta a mostrar cuan fácil es grabar las tramas enviadas por un peer, y como esta grabación puede ser re-utilizada directamente, sin necesariamente utilizar procesos de post-procesamiento.

El proceso de configuración es bastante simple: solo elegir donde deben ser guardadas las grabaciones. La misma carpeta será usada también para listar las grabaciones disponibles que pueden ser repetidas (replayed).

2.11 Servidor para compartir archivos – SAMBA

El presente marco teórico está fundado en la bibliografía de referencia (36) y (37).

Samba es un servidor SMB(Server Message Block) libre, desarrollado por Andrew Tridgell y que en la actualidad es mantenido por un grupo de personas de todo el mundo, como casi todos los proyectos distribuidos bajo la Licencia Pública General de GNU. Samba es capaz de ejecutarse en una gran cantidad de variantes Unix, como Linux, Solaris, SunOS, HP-UX, ULTRIX, Unix de Digital, SCO Open Server y AIX por nombrar tan sólo algunas. Con Samba podremos hacer que nuestro sistema Linux actúe como servidor SMB dentro de la red.

¿Cómo funciona este servicio?

Samba es en sí un paquete muy complejo, que brinda a los usuarios Linux de un sin fin de posibilidades a la hora de interactuar con equipos Windows y Linux que estén coexistiendo en redes heterogéneas.

¿Cuáles son los beneficios al instalar un servidor Samba en Linux?

- Compartir uno o más sistemas de archivos.
- Compartir impresoras, instaladas tanto en el servidor como en los clientes.
- Samba permite compartir entre máquinas Windows y Linux recursos.
- Siendo un recurso una carpeta o la impresora.

2.11.1 Arquitectura Samba

Samba está compuesta de un servidor y un cliente, así como de algunas herramientas que permiten realizar servicios prácticos o hacer un test de la configuración.

El servidor está compuesto de dos aplicaciones (llamadas demonios): smbd, núcleo del servidor, provee los servicios de autenticación y acceso a los recursos. nmbd, permite mostrar los servicios disponibles en Samba (visualización de los servidores Samba en la red,...).

El cliente: smbclient es un cliente para Linux que provee una interfaz que permite la transferencia de archivos, el acceso a impresoras, etc.

smbtar: permite transferir de o hacia un archivo TAR bajo Linux

testparm: comprueba la sintaxis del archivo smb.conf, el archivo de configuración de Samba.

El protocolo de comunicación que permite esta comunicación entre Windows y Linux se llama SMB (Server Message Block). Puesto a punto por Microsoft en 1987, retomando un concepto desarrollado por IBM en 1985 (NetBIOS), este protocolo se apoya sobre NetBEUI (así como sobre TCP/IP). El interés de TCP/IP proviene del hecho que es ampliamente adoptado. Por ello TCP/IP ya ha sido implementado en la mayoría de sistemas operativos (Unix, Linux, AmigaOS, MacOS, OS/2,...) según el esquema siguiente:

- Aplicación
- SMB
- NetBios
- TCP/IP
- NetBeui
- IPX/SPX
- Controladores de red

Capítulo 3 - Estudio del problema

3.1 Introducción

En esta sección se realizan las comparativas entre los diferentes software y hardware que son opción para implementar en el sistema y la decisión que es tomada para seleccionar los más óptimos para nuestro proyecto.

Además explicamos cual es la metodología de trabajo y dividimos el proyecto en iteraciones.

3.2 Arquitectura del sistema

Para clasificar el software o hardware en las tablas de comparación se utilizan las siguientes medidas:

1	Malo
2	Regular
3	Bueno
4	Muy Bueno

Tabla 8: Medidas de valor

3.2.1 Control de acceso

3.2.1.1 Comparación

En el siguiente cuadro se comparan los diferentes dispositivos para controlar acceso que se tuvieron en cuenta para desarrollar el sistema de acuerdo a las características fundamentales que creemos que debe tener el mismo.

	Proximidad RFID	Teclado numérico	Huella dactilar
Velocidad de marcación	4	1	4
Acceso a distancia	2	-	-
Seguridad	3	2	4
Costo	3	4	1

Vida útil	4	2	2
-----------	---	---	---

Tabla 9: Comparación control de acceso

3.2.1.2 Decisión

El sistema elegido para implementarse es el de acceso por proximidad de tarjetas de seguridad.

La tecnología de proximidad es bastante adecuada en vista que:

- la lectura de la tarjeta se realiza a distancia (típicamente hasta a 15 centímetros)
- la marcación es rápida
- no existen problemas de suciedad
- no existen problemas de desgaste (fricción)
- Bajo costo

Existen en el mercado diversos lectores RFID. La mayoría cumple con las exigencias que presenta el sistema, como es el caso de los lectores: **Wiegand Intemperie, Klaxia RD1, PCT-100**. Dichos lectores se encuentran disponibles en el país y tienen un costo similar.

El lector utilizado es el **KLAXIA RD1** por recomendación del Dr.Ing.Orlando Micolini el cual posee las siguientes características:

Tamaño	55x75 mm, profundidad 30 mm.
Alimentación	9 a 12 VCA o VCC mínimo 50 mA, máximo 1 A.
Temperatura de operación	-40 a +85 °C.
Rango de lectura	Max. 10 cm.
Frecuencia de operación	125 KHz.
Codificación	Manchester, 64 bits.
Modelo RD1	Relay electrónico integrado, salida VCA o VCC max. 1A, apto cerradura eléctrica, cerrojo o pestillo.
Modelo RD1-A	Relay mecánico integrado, apto cerradura eléctrica, cerrojo, pestillo o cerradura electromagnética.
Salida RS-232	Comanda 250 VCA máx. 7 Amperes o 24 VDC máx. 10 Amperes.
Alcance RS-232	1200 baudios, 8 bits, Paridad N, 1 stop bit. 1500 metros, cable apantallado.

3.2.2 Sistemas embebidos

3.2.2.1 Comparación

En el siguiente cuadro se comparan los diferentes ordenadores de placa reducida que se tuvieron en cuenta para desarrollar el sistema de acuerdo a las características fundamentales que creemos que debe tener el mismo.

	Raspberry Pi	Cubieboard	Arduino
Costo	3	1	3
Disponibilidad	3	1	4
Documentación disponible	4	2	4
Pines digitales	4	4	4
Puerto Ethernet	4	4	2*
Puertos USB	4	4	2*
I/O Audio	4	4	2*
Procesamiento de video	3	4	-

Tabla 10: Comparación sistemas embebidos

(*) Es necesario adquirir shields extra lo cual incrementa su costo y tamaño considerablemente.

3.2.2.2 Decisión

Para poder tomar una decisión sobre cuál de todos los sistemas embebidos elegir, tuvimos en cuenta una serie de requisitos para llegar a su elección. Si bien el dispositivo elegido debe cumplir con las principales funcionalidades, el mismo tiene que ser lo más económico posible, tener que poder conseguirse fácilmente en el país y poder tener soporte tanto oficial o no oficial de su hardware y de su software. Por eso decidimos utilizar el ordenador Raspberry Pi que no sólo es capaz de resolver todos nuestros requerimientos sino que también es uno de los más económicos. Posee gran información en la web tanto de su hardware como de sus sistemas operativos, existen foros de ayuda y una página oficial.

3.2.3 Sistemas operativos

3.2.3.1 Comparación

En el siguiente cuadro se comparan los diferentes sistemas operativos que se tuvieron en cuenta para desarrollar el sistema de acuerdo a las características fundamentales que creemos que debe tener el mismo.

	Raspbian	Pidora	Arch Linux ARM	OpenELEC*	Raspbmc*
Costo	4	4	4	4	4
Documentación disponible	4	3	3	2	2
Facilidad de instalación	4	4	4	4	4
Usabilidad	4	2	2	3	2

Tabla 11: Comparación sistemas operativos

(*) Orientados a centros multimedia.

3.2.3.2 Decisión

Para el desarrollo del Sistema de control y seguridad optamos por el sistema operativo Raspbian, el cual es el SO por defecto de la placa Raspberry Pi. Existe mucha documentación, es de fácil instalación y además posee una interfaz amigable que permite el rápido aprendizaje y utilización del mismo.

3.2.4 Sistemas de bases de datos

3.2.4.1 Comparación

En el siguiente cuadro se comparan los diferentes sistemas de gestión de bases de datos que se tuvieron en cuenta para desarrollar el sistema de acuerdo a las características fundamentales que creemos que debe tener el mismo.

	SQLite	MySQL	PostgreSQL
Tamaño (Kb)	4	2	2
Usabilidad	4	4	3
Integración con lenguajes de programación	4	4	4
Seguridad	4	3	3

Tabla 12: Comparación sistemas de bases de datos

3.2.4.2 Decisión

Entre todos los motores de bases de datos hemos elegido el sistema SQLite ya que es uno de los más livianos y fáciles de manejar. Ocupa muy poco espacio en disco lo cual es esencial para este proyecto. Tiene una fuerte integración con el lenguaje Python lo cual hace mucho más fácil su programación y manipulación de los datos.

SQLite al funcionar como un simple archivo con tablas y filas, hace que sea más difícil poder acceder a sus datos, a diferencia de Mysql que ingresando a través del ip con puerto, usuario y contraseña. Esta ventaja nos hace que los datos estén más seguros, dado que solo el sistema operativo de las Raspberry Pi podrá acceder a esos datos.

3.2.5 Lenguajes de programación

3.2.5.1 Comparación

En el siguiente cuadro se comparan los diferentes lenguajes de programación que se tuvieron en cuenta para desarrollar el sistema de acuerdo a las características fundamentales que creemos que debe tener el mismo.

	C	Python	Java
Usabilidad	2	4	3
Integración con base de datos SQLite	4	4	4
Portabilidad	4	4	2
Documentación	4	4	4
Compatibilidad con Sistema Operativo Raspbian	3	4	3

Tabla 13: Comparación lenguajes de programación

3.2.5.2 Decisión

Como mencionamos anteriormente, en el sistema operativo Raspbian para Raspberry Pi es posible crear scripts en distintos lenguajes de programación. Para este sistema necesitamos uno que sea capaz de poder manejar una base de datos, manipular los pines GPIO del ordenador y que sea capaz de interactuar con la cámara web para poder capturar las imágenes. Los dos lenguajes más aptos para poder realizar nuestros requerimientos son Python y C. Ambos son capaces de realizar las funciones que necesitamos, ambos tienen gran soporte e información en internet.

Cuando se instala Raspbian, éste sistema operativo ya viene instalado y compilado casi todos los módulos necesarios en Python. Es por eso que elegimos ese lenguaje de programación. Python es simple, intuitivo y ya viene con módulos compilados que nos permite facilitar la programación. Como por ejemplo, existe un módulo “serial” que nos facilita poder utilizar la UART de las Raspberry pi. Otro de los módulos más utilizados es el de “SQLite” en donde nos permite de una manera rápida y sencilla poder ejecutar comandos SQL hacia nuestra base de datos.

3.2.6 Servidores web

3.2.6.1 Comparación

En el siguiente cuadro se comparan los diferentes servidores web que se tuvieron en cuenta para desarrollar el sistema de acuerdo a las características fundamentales que creemos que debe tener el mismo.

	Apache	NGINX	Django
Usabilidad	3	3	3
Integración con Python	2	2	4
Documentación	4	3	4
Compatibilidad con Sistema Operativo Raspbian	4	4	4

Tabla 14: Comparación servidores web

3.2.6.2 Decisión

Ya que nuestro lenguaje de programación elegido es python, decidimos hacer uso del Framework Django, ya que éste nos permite reutilizar código o scripts de python en nuestro servidor web.

3.2.7 Servidor de comunicaciones

3.2.7.1 Comparación

En el siguiente cuadro se comparan los diferentes servidores de comunicaciones que se tuvieron en cuenta para desarrollar el sistema de acuerdo a las características fundamentales que creemos que debe tener el mismo.

	Asterisk	3CX	Elastix
Costo	4	1	4
Transporte de voz vía IP (VoIP)	4	4	4
Documentación	4	3	3
Compatibilidad con Sistema Operativo Raspbian	4	1	2

Tabla 15: Comparación servidor de comunicaciones

3.2.7.2 Decisión

Para poder implementar la comunicación por voz entre un usuario interno y otro externo necesitamos poder transportar la voz a través de IP. Por eso necesitamos de un servidor de VoIP. Entre todos los que existen nosotros hicimos uso del servidor Asterisk.

La elección se llevó a cabo dado que Asterisk es un servidor gratuito. Existe mucha información sobre su instalación y configuración como así existen foros de ayuda. También lo elegimos porque se puede embeber en la Raspberry (El servidor 3CX corre sólo en plataforma Windows).

3.2.8 Sistema de vigilancia

3.2.8.1 Comparación

En el siguiente cuadro se comparan los diferentes software de vigilancia que se tuvieron en cuenta para desarrollar el sistema de acuerdo a las características fundamentales que creemos que debe tener el mismo.

	Motion	Mpeg Streamer
Costo	4	4
Integración con múltiples cámaras web	4	1
Documentación	3	3
Compatibilidad con Sistema Operativo Raspbian	4	4
Captura de imágenes	4	1
Visualización en tiempo real	4	4
Usabilidad	4	3
Facilidad de instalación	4	2

Tabla 16: Comparación sistemas de vigilancia

3.2.8.2 Decisión

Para lograr el Streaming de video y la captura de imagen decidimos utilizar el software Motion. Si bien tanto el Motion como el software Mpge-streamer cumplen con nuestro requerimientos, Motion tiene la posibilidad de interconectar hasta 3 cámaras USB y 1 cámara Ip extra. Por lo cual si a trabajos futuros deseamos agregar más cámaras, ya tenemos el mismo software capaz de resolver el problema.

3.2.9 Softphone

3.2.9.1 Comparación

En el siguiente cuadro se comparan los diferentes servidores de comunicaciones que se tuvieron en cuenta para desarrollar el sistema de acuerdo a las características fundamentales que creemos que debe tener el mismo.

	Twinkle	Linephone
Costo	4	4
Tamaño (Kb)	4	3
Documentación	2	2
Compatibilidad con Sistema Operativo Raspbian	4	4
Auto-contestación de llamada	4	4
Interfaz de comandos por consola	4	4

Tabla 17: Comparación softphones

3.2.9.2 Decisión

Para poder lograr la comunicación es necesario la instalación de un softphone dentro de la Raspberry pi. El que elegimos fue Twinkle ya que es gratuito, liviano, corre bajo Linux. Otra de las características que cumple es que se puede ejecutar bajo un comando desde la consola, y lo más importante es que tiene un “Auto Answer” que nos permite auto contestar la llamada. Cualquiera de las dos eran opciones viables y similares, pero nos decidimos por twinkle ya que es de menor tamaño lo cual no es un dato menor en un sistema embebido.

3.2.10 Comunicación web en tiempo real

Para realizar exitosamente la comunicación entre usuarios externos e internos hay que implementar una pasarela WebRTC (Web Real Time Communication). Es decir, una comunicación en tiempo real basada en un navegador, ya que será por este medio que se realice el llamado para establecer la conversación.

La única herramienta que conseguimos de código abierto y que cumpla con nuestros requerimientos se denomina JANUS y fue la utilizada para desarrollar nuestro sistema.

Al ser el único software que encontramos en la web no existen competidores para el mismo y por lo tanto no se realizó ningún tipo de comparación ni toma de decisiones.

3.3 Metodología de trabajo

El modelo de desarrollo de software utilizado es el proceso iterativo con algunas características de scrum. En cada ciclo de desarrollo se parte de ciertos requerimientos a ser desarrollados en el prototipo, obteniéndose un sistema funcional al final de cada iteración.

Las características de scrum están dadas debido a que el equipo de desarrollo no contaba con la experiencia de haber realizado desarrollo de sistemas embebidos para control de acceso como así tampoco la tenía en el manejo de ordenadores de placa reducida del estilo de la Raspberry Pi. A su vez la prioridad que se presentaba al momento de comenzar con el desarrollo del sistema de seguridad fue la satisfacción de las necesidades del cliente, por lo tanto el sistema debía adaptarse a las características que este pretendía.

Otra variable que se presentaba es que el sistema debía ir modificándose y mejorando de acuerdo a los cambios y nuevas características y funcionalidades que el cliente requería en tiempo de desarrollo, por lo cual las características de una metodología ágil como Scrum eran las que mejor se adaptaban para desarrollar el proyecto.

3.4 Iteraciones del proyecto

Iteración1: Control de acceso

Elección de sistema de acceso a implementar
Lectura y escritura de datos entre la placa y el sistema de acceso
Instalación y creación de la base de datos
Script para controlar el acceso del personal
Programar franjas horarias
Brindar reportes de eventos

Iteración 2: Vigilancia

Elección de software de streaming a utilizar
Instalación y configuración de cámara web en la placa
Captura de imágenes y guardado de las mismas
Visión nocturna con leds infrarrojos

Iteración3: Software de control y manejo de sistema

Elección de servidor web
Instalación y configuración del servidor web
Creación de página principal con login
Visualización de cámara
Visualización de usuarios registrados
Aregar, editar y eliminar usuarios
Visualización de eventos
Visualización de imágenes
Abrir puerta desde página web
Exportar tablas a Microsoft Excel

Iteración 4: Comunicación vía VoIP

Elección de softphone a utilizar
Instalación y configuración de softphone
Auto contestación de llamada por parte de la placa
Comunicación bidireccional entre el administrador y el usuario externo
Integración con página web

Iteración5: Seguridad del Hardware

Colocación de LEDs de control
Aviso de falla del sistema víaSMS o email
Aregar batería para asegurar funcionamiento ante cortes de energía
Script de control para levantar servicios caídos

Tabla 18: Iteraciones del proyecto

Capítulo 4 –Diseño, implementación y resultados

El diseño y los casos de prueba elegidos en este capítulo son aquellos que consideramos más críticos e importantes en el sistema de seguridad. Se verifica el cumplimiento de determinados requerimientos previamente planteados al comienzo del desarrollo del proyecto. La implementación esta dada de forma completa, abarcando todas las funcionalidades del sistema y no solo las más críticas.

4.1 Iteración 1

4.1.1 Objetivos

En esta iteración se desarrolla el módulo de control de acceso, que permite el ingreso a toda persona que esté registrada como usuario activo en el sistema mediante el uso de una tarjeta de proximidad RFID.

4.1.2 Diseño

En este apartado se describe cómo es el diseño del sistema de acceso de los usuarios, se visualizan los requerimientos involucrados referidos al ingreso de los usuarios externos, así como también se muestra la secuencia involucrada para que pueda realizarse.

4.1.2.1 Requerimientos

REQUERIMIENTOS INVOLUCRADOS		
Número	Prioridad	Requerimiento
2	Alta	Deberá existir un sistema de control de ingreso con tarjetas por proximidad
6	Alta	Se deberán poder visualizar eventos de ingreso permitido y prohibido
12	Media	El sistema de ingreso deberá accionar un buzzer como aviso de ingreso habilitado

Tabla 19: Requerimientos involucrados en el sistema de acceso

4.1.2.2 Diagrama de secuencia

En la siguiente ilustración podemos observar el diagrama de secuencia del sistema de acceso:

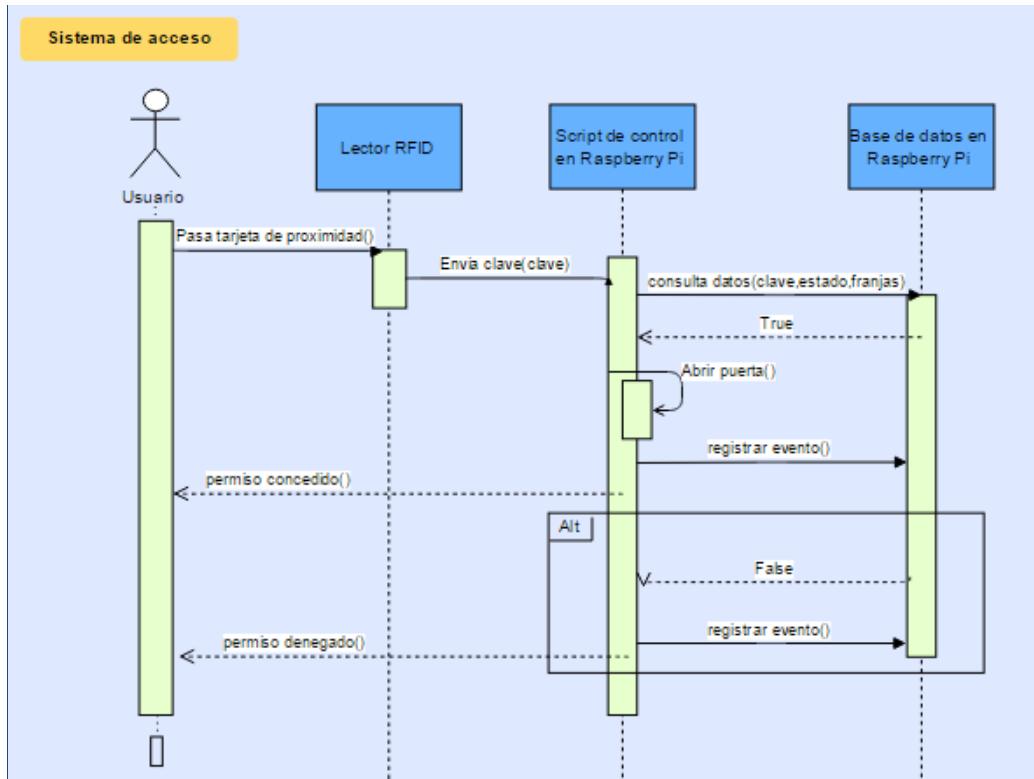


Ilustración 3: Diagrama de secuencia del sistema de acceso

4.1.3 Implementación

4.1.3.1 Instalación, configuración y actualización del sistema operativo en Raspberry Pi

4.1.3.1.1 Instalación de RASPBIAN

Para instalar Raspbian en la Raspberry Pi es necesario crear una memoria SD de arranque con dicho sistema operativo. Esto se puede lograr de la siguiente manera:

- 1) Descargar la imagen de Raspbian de la siguiente URL: <http://www.raspberrypi.org/downloads>
- 2) La imagen está comprimida en un archivo ZIP. Descomprimir la imagen. Obtendremos un archivo con extensión img.
- 3) Introducir la tarjeta SD y crear la SD arrancable a partir del archivo de imagen:
 - 3.1) Si usas un SO basado en UNIX, utiliza la herramienta dd: [http://en.wikipedia.org/wiki/Dd_\(Unix\)](http://en.wikipedia.org/wiki/Dd_(Unix))

3.2) Si usas Windows, utiliza la herramienta Win32DiskImager: <http://sourceforge.net/projects/win32diskimager>

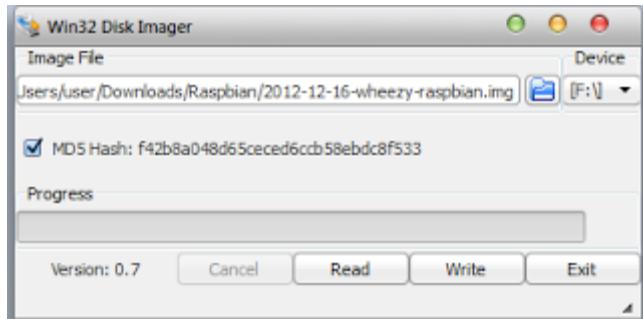


Ilustración 4: Win32DiskManager

En el caso de utilizar Win32DiskImager, el procedimiento es muy sencillo:

- 1) Hacer clic en el ícono de la carpeta para seleccionar el archivo de imagen de Raspbian.
- 2) Seleccionar la unidad (Device) donde se encuentra la SD.
- 3) Hacer clic en el botón "Write" para generar la SD arrancable.

4.1.3.1.2 Configuración de RASPBIAN

Luego se debe configurar, por lo que se deben seguir los siguientes pasos:

1) **Opción "expand_rootfs - Expand root partition to fill SD card".** La SD tendrá dos particiones: una de swap (intercambio) y otra principal, donde se aloja Raspbian. Esta partición ocupa unos pocos MB, exactamente lo que ocupa Raspbian. El resto de la SD está sin definir. Mediante esta opción se extiende la partición de Raspbian hasta ocupar todo el espacio restante.

2) **Opción "configure_keyboard - Set keyboard layout".** Permite configurar el teclado.

2.1) Primero pedirá el tipo de teclado (los teclados antiguos suelen ser de 102 teclas. Los nuevos son de 105)

2.2) Despues pedirá el idioma del teclado. Por defecto está configurado en inglés británico. Seleccionar "Other" (otro), y buscar el idioma correspondiente. En nuestro caso, "Spanish".

2.3) A continuación solicitará qué tipo de idioma, en concreto, es el teclado. Un mismo idioma puede tener diversos teclados distintos, dependiendo del país o de la región. Hay teclados distintos para cada país latinoamericano.

2.4) Despues solicitará el tipo de distribución de la tecla AltGr.

2.5) Luego pedirá la tecla para componer AltGr (compose key). En algunos teclados, para componer la combinación de AltGr se requiere pulsar dos o tres teclas. Por defecto, seleccionar "No compose key".

2.6) Por último, solicitará si deseamos asignar la combinación de teclas Ctrl +`Alt + Borrar para salir del modo de interfaz gráfica. Por defecto, la opción es NO.

3) **Opción "change_locale - Set locale"**. Permite configurar el idioma para el sistema operativo. Esto influye, principalmente en el modo de representar los números, la moneda y las fechas. Aparecerá un menú con muchas opciones. Por defecto, está seleccionada en_GB, es decir, el inglés de Gran Bretaña. Se desmarcará con la tecla espacio. A continuación localizamos nuestro idioma, teniendo en cuenta que las dos primeras letras es el idioma, y las dos siguientes el país o región. Una vez localizado, seleccionar, concretamente el mapa de códigos. Se recomienda utilizar UTF-8. Marcar la opción con espacio. Después, pulsar Enter para confirmar.

4) **Opción "change_timezone - Set timezone"**. Permite seleccionar la zona horaria. Primero se elige la región continental y a continuación la ciudad más cercana que tenga la zona horaria.

5) **Opción "change_pass - Change password for 'pi' users"**. Permite configurar la contraseña (por defecto es, usuario 'pi' y contraseña 'raspberry'). Pedirá dos veces la contraseña.

6) **Opción "boot_behaviour - Start desktop on boot?"**. Si elegimos "Yes", permite comenzar con el entorno gráfico al arrancar Raspberry.

7) **Opción "Finish"**. Termina la configuración de Raspbian, guardando los cambios realizados.

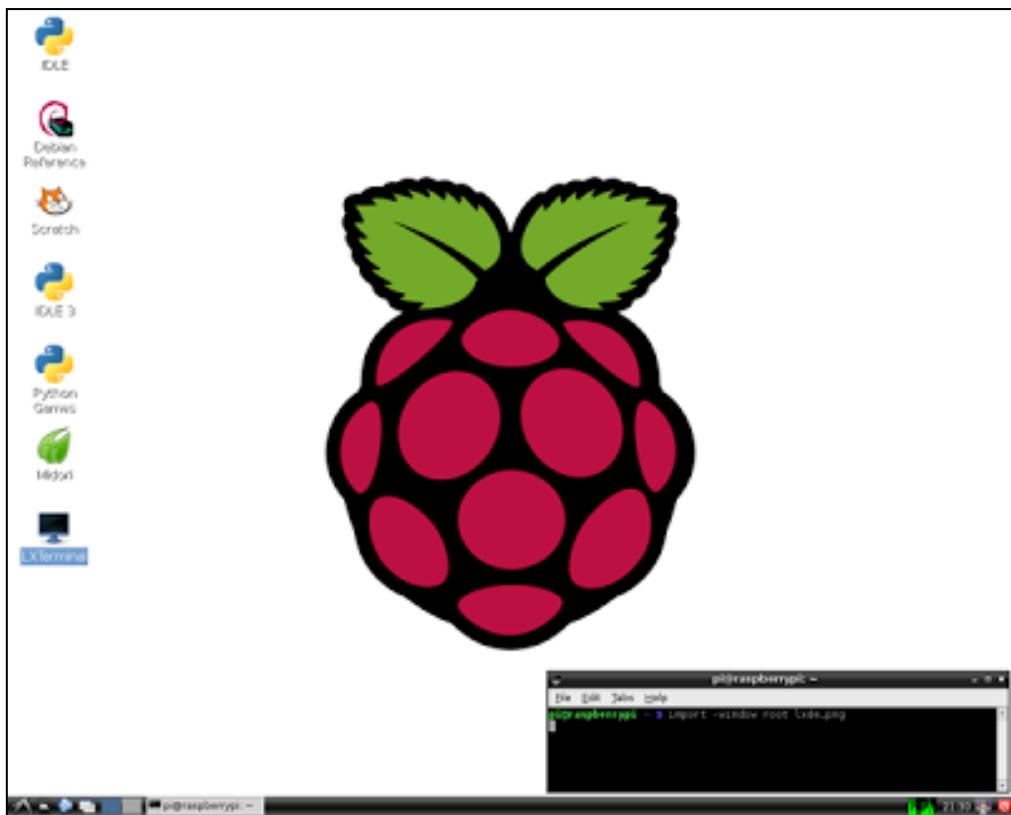


Ilustración 5: Interfaz gráfica Raspbian

4.1.3.1.3 Actualización del Sistema Operativo y kernel

Cuando instalamos un sistema operativo Linux en la Raspberry Pi, normalmente ya existe en circulación una actualización o una revisión de mejoras que nuestro sistema no contiene. Para actualizar el sistema, debemos de realizar una serie de acciones para poder tener el último kernel compilado en nuestra Raspberry Pi.

Lo primero que haremos será actualizar los repositorios:

sudo apt-get update

A continuación actualizaremos todos los programas:

sudo apt-get upgrade

Así como es posible actualizar el sistema operativo (Raspbian) de la tarjeta Raspberry Pi también es posible actualizar el firmware de la misma y su kernel. Esto puede traer mejoras como gestión de energía y soporte actualizado para los puertos GPIO entre otras cosas

Esto normalmente sucede mediante el paquete raspberrypi-bootloader el cual se encuentra contenido en los repositorios estándar de Raspbian. Esta versión se actualiza con regularidad.

frecuencia y a pesar de no acostumbra a ser la última disponible, acostumbra a ser una bastante probada y estable.

Estos archivos -kernel, firmware y bootloader- residen en una partición independiente del directorio raíz en la tarjeta SD. Por este motivo no es la Raspberry Pi que se actualiza sino la tarjeta SD con el sistema operativo.

Si se desea actualizar el firmware y el kernel de la tarjeta a las últimas versiones disponibles se debe utilizar el siguiente comando:

sudo rpi-update

Veremos cómo se descarga el kernel de los repositorios github y se instala. Una vez termine la actualización, nos pedirá que reiniciemos para que los cambios surjan efecto. Una vez hayamos reiniciado, comprobaremos la versión del kernel para confirmar que se ha actualizado de la siguiente manera:

uname -r

Debe tenerse en cuenta que esta versión posiblemente sea menos estable que la incluida en los repositorios.

4.1.3.2 Instalación y configuración de SAMBA

Para que cada desarrollador pueda trabajar desde su propia PC fue necesario compartir los archivos desde la Raspberry Pi para que puedan ser accedidos y modificados. Aquí podremos observar los pasos para instalar el servidor Samba y su respectiva configuración

4.1.3.2.1 Instalación

En primer lugar debemos realizar la instalación. Para ello debemos utilizar los siguientes comandos en la terminal de la placa Raspberry Pi:

```
sudo apt-get install samba samba-common-bin
```

El sistema puede realizar la siguiente pregunta. En ese caso se deberá responder con “y” + [enter]:

```
Do you want to continue? [Y/n]
```

A continuación es conveniente realizar una copia del fichero de configuración de Samba que se genera durante la instalación:

```
sudo cp /etc/samba/smb.conf /etc/samba/smb.conf.old
```

4.1.3.2.2 Activar la seguridad

Esto es opcional pero muy recomendable, obliga al servidor Samba a preguntar por usuario y contraseña antes de permitir la conexión desde otro ordenador.

Editar el fichero de configuración de Samba:

```
sudo nano /etc/samba/smb.conf
```

Buscar en el fichero la sección que contiene el siguiente texto:

Authentication

Cambiar el texto:

security = user

Por el texto:

security = user

Grabar los cambios en el fichero y salir:

Pulsar Control-X

Teclear y Pulsar [enter]

Reiniciar el servidor Samba para que cargue la nueva configuración

```
sudo /etc/init.d/samba restart
```

El sistema debe responder con lo siguiente:

```
Stopping Samba daemons: nmdb smdb
```

```
Starting Samba daemons: nmdb smdb
```

4.1.3.2.3 Configurar áreas de almacenamiento privadas y usuarios

Esta configuración permite que Samba asigne un directorio “home” a cada usuario. Editar el fichero de configuración de Samba (smb.conf):

```
$ sudo nano /etc/samba/smb.conf
```

Buscar en el fichero la sección que contiene el siguiente texto:

[homes]

Cambiar el texto:

```
read only = yes  
Por el texto:  
read only = no
```

Grabar los cambios en el fichero y salir:

cargue la nueva configuración

```
sudo /etc/init.d/samba restart
```

El sistema debe responder con lo siguiente:

```
Stopping Samba daemons: nmdb smdb  
Starting Samba daemons: nmdb smdb
```

Por defecto el usuario pi está ya definido. Para permitir a pi ser usuario de Samba, hacer lo siguiente:

```
sudo smbpasswd -a pi
```

Será solicitada una contraseña.

Para añadir nuevos usuarios a Samba se debe hacer lo siguiente.

Lo primero añadir el usuario al Sistema:

```
sudo useradd -m -G users
```

```
sudo passwd
```

Será solicitada una contraseña.

y a continuación permitir el acceso de ese usuario a Samba:

```
sudo smbpasswd -a
```

Será solicitada una contraseña.

4.1.3.2.4 Configurar un área de almacenamiento pública

En este caso se muestra como compartir una carpeta de la Raspberry, que es donde estarán presentes todos los archivos que se querrán modificar (templates, scripts de python, etc) Asignar los permisos de acceso a la carpeta:

```
sudo chown -R root:users /usr/src
```

```
sudo chmod -R ug=rwx,o=rx /usr/src
```

Editar el fichero de configuración de Samba:

```
sudo nano /etc/samba/smb.conf
```

Situarse en el final del fichero y añadir el texto siguiente:

```
[public]
comment = Public Storage
path = /usr/src
valid users = @users
force group = users
create mask = 0660
directory mask = 0771
read only = no
```

Grabar los cambios en el fichero y salir:

Pulsar Control-X

Teclear y Pulsar [enter] Reiniciar el servidor Samba para que cargue la nueva configuración

```
sudo /etc/init.d/samba restart
```

El sistema debe responder con lo siguiente:

```
Stopping Samba daemons: nmdb smdb
```

```
Starting Samba daemons: nmdb smdb
```

4.1.3.3 Instalación y creación de la base de datos

En este apartado veremos los pasos a seguir para instalar el motor de base de datos SQLite3 y además crearemos las tablas que vamos a necesitar para el correcto funcionamiento del sistema de control de acceso.

4.1.3.3.1 Instalación de SQLite3

Lo primero es instalar el “motor” de la base de datos. La siguiente orden instala todo lo necesario

```
$ sudo apt-get install sqlite3
```

Para manejar SQLite de manera gráfica, existen herramientas tales como Sqliteman. Pero en nuestro caso no vamos a necesitar la interfaz gráfica, ya que buscamos optimizar recursos y ocupar el menor espacio posible en la memoria de la placa Raspberry Pi.

4.1.3.3.2 Creación de la base de datos

En primer lugar debemos crear la base de datos. Para ello utilizamos un complemento de Mozilla Firefox denominado “SQLite manager” que facilita mucho este proceso. Con este complemento nos evitamos de tener que instalar software extra en la Raspberry Pi y además nos provee una interfaz mediante el navegador que nos permite ganar tiempo en la creación de la base de datos.

Una vez instalado el complemento accedemos a él mediante el navegador y nos dirigimos a la pestaña “base de datos” y luego clicamos en “Nueva Base de datos”. En nuestro caso la creamos con el nombre “database”:

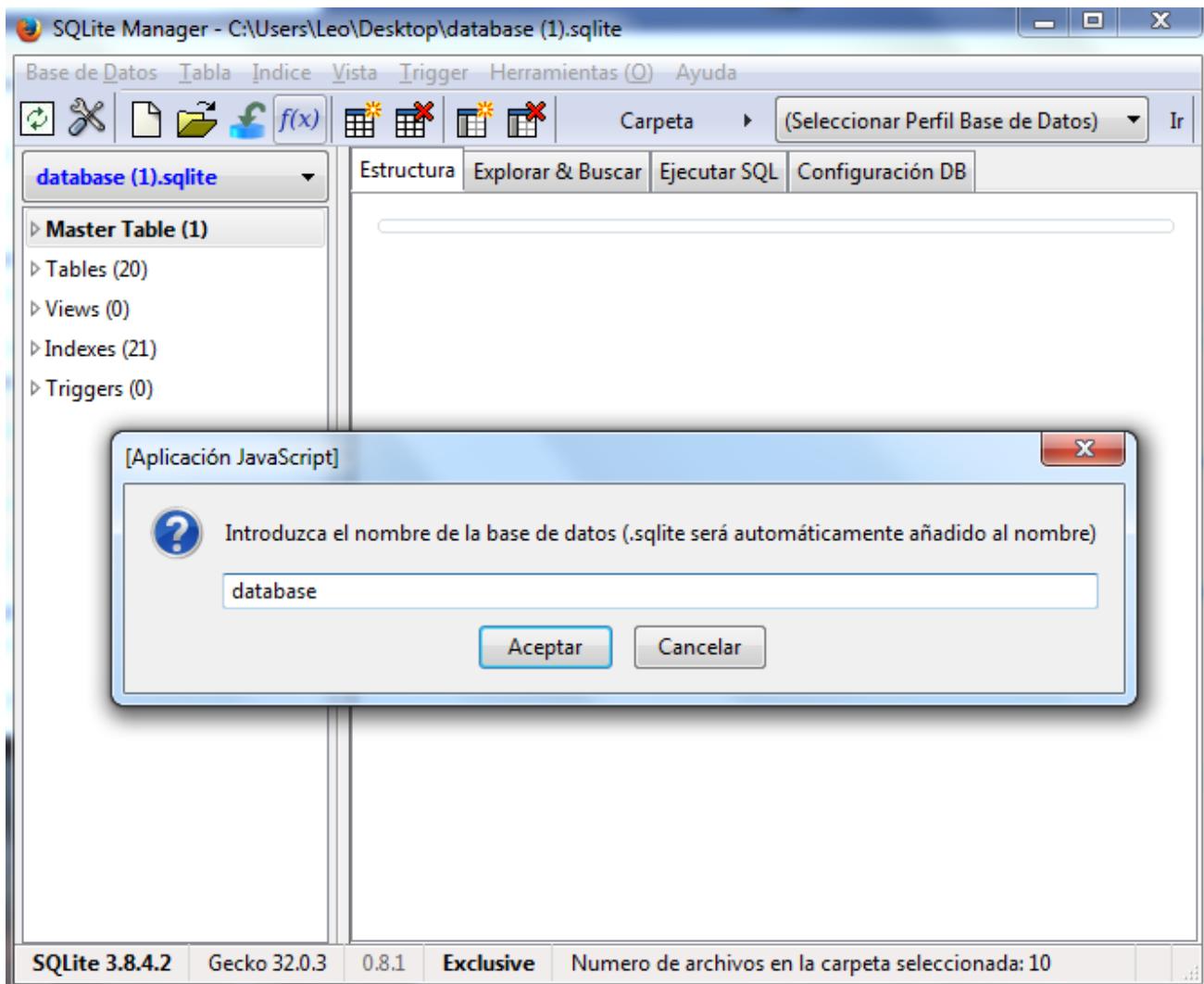


Ilustración 6: SQLite Manager

La ventaja de tener este complemento es que luego podremos manipular la base de datos y realizar modificaciones más fácilmente.

4.1.3.4 Script para controlar acceso

Una vez realizada la captura del código de cada tarjeta/llavero, mediante el lector de radiofrecuencia, es necesario realizar acciones que permitan o denieguen el acceso al laboratorio, así como también capturar imágenes, abrir la puerta y realizar consultas a la base de datos del sistema. Para ello realizamos un script en lenguaje Python denominado "**control.py**", del cual dividiremos su código a continuación en varias partes para analizar las distintas funcionalidades que posee.

4.1.3.4.1 Main y llamado a funciones

En esta sección del script veremos la parte principal del mismo, el cual realiza el llamado a las funciones principales de la siguiente manera:

```
if __name__ == '__main__':
    serialPort = connect()
    while1:
        rfid = readRfid(serialPort)
        cursordb = conectBD()
        db = cursordb[0]
        cursor = cursordb[1]
        searchUser(rfid, db, cursor)
        t = threading.Thread(target=captureCode, args=(rfid,))
        threads.append(t)
        t.start()
```

Como podemos observar en primer lugar realizamos la conexión serial para habilitar la comunicación entre el lector de tarjetas y la placa Raspberry Pi. Este main será un bucle infinito que permitirá poder leer constantemente el lector de radiofrecuencia a la espera de algún evento de captura de clave. Las funciones específicas que son llamadas las veremos a continuación individualmente.

Antes de entrar en el bucle infinito se conecta el puerto serie a partir de la siguiente función:

```
defconnect():
    serialPort = serial.Serial("/dev/ttyAMA0", 1200, timeout=0.5, rtscts=False, dsrdtr=False,
xonxoff=False, bytesize=serial.EIGHTBITS, parity=serial.PARITY_NONE)
    return serialPort
```

La variable “serialPort” es la que se utiliza dentro del main para realizar la conexión serial a través de la función “connect()”. Los datos completados en la función “Serial()” son los necesarios para comunicarse con el lector específico que fue utilizado para el sistema de control de acceso de este proyecto integrador.

4.1.3.4.2 Lectura de clave RFID

Esta función va a permitir la captura de la clave que es aproximada al lector. Es la primer función que llama el Main.

```
defreadRfid(serialPort):
    while1:
        userCode = serialPort.readline()
        if (len(userCode) >6):
            vectorRfid = []
            vectorRfid = userCode.split(chr(13).encode('ascii'))
            rfid = vectorRfid[0]
    return rfid
```

Como podemos observar lo primero que se realiza es una lectura de línea en el puerto serie asignado, la cual se guarda en la variable “*userCode*”. Una vez capturado el código vemos si es mayor a 6 caracteres, lo cual nos permite identificar si lo que fue leído se aproxima a ser una clave de tarjeta rfid. Luego dividimos la línea en 13 caracteres (que son los que posee una clave rfid, incluyendo espacios) y los guardamos en un vector denominado “*rfid*”. Este vector será el utilizado para manejar la clave durante todo el script.

4.1.3.4.3 Conectar a la Base de Datos

Para realizar las consultas necesarias para permitir o denegar el acceso (entre otras cosas) es fundamental primero realizar la conexión a la base de datos, para lo cual contamos con la siguiente función:

```
defconectBD():
    db = sqlite3.connect('/usr/src/web/database.sqlite')
    cursor = db.cursor()
    DataBase = []
    DataBase.append(db)
    DataBase.append(cursor)
    return DataBase
```

Lo primero que realizamos es la conexión a la base de datos sqlite3 que ya fue generada anteriormente, para lo cual debemos colocar el path al archivo específico de nuestra base de datos. Luego generamos el cursor, que nos va a permitir recorrer y procesar los registros de resultado de las consultas que se realicen.

Por ultimo creamos un vector denominado “*dataBase*” al cual le asignaremos las variables generadas recientemente (*db* y *cursor*) y dicho vector será el utilizado luego en todo nuestro script para trabajar con la base de datos.

4.1.3.4.4 Abrir puerta

Esta es una función muy simple que va a permitir abrir la puerta del laboratorio y será utilizada siempre que el usuario este habilitado para ingresar. El código de la función es el siguiente:

```
defopenDoor():
    GPIO.output(3,True)
    time.sleep(2)
    GPIO.output(3,False)
```

Para poder abrir la puerta se utiliza un pulso que es enviado al pin 3 de la Raspberry Pi (el cual se comunicara con la cerradura eléctrica). Una vez enviado el pulso la función se duerme mediante un “*sleep()*” por dos segundos, permitiendo así dar tiempo al usuario para que pueda ingresar al laboratorio. Una vez pasados dichos segundos se apaga el pulso (mandando un *False* a el pin 3 de la Raspberry Pi).

Más adelante veremos que quien llama a “*openDoor*” es la función que controla las franjas horarias.

4.1.3.4.5 Consulta de usuario en base de datos

Esta función será la encargada de preguntar el estado del usuario que intento ingresar al laboratorio mediante la aproximación de su tarjeta personal de seguridad.

```
defsearchUser.rfid, db, cursor):
    mi_query = "SELECT*FROM control_datos_usuarios_dj WHERE clave_usuario ='%s'"%(rfid)
    cursor.execute(mi_query)
```

```

user = cursor.fetchone()

if user <>None:

    estado = user[9]

    idUser = user[0]

    if estado =='ACTIVO':

        checkTime(db,cursor,idUser)

        registerEvent(db,cursor,user)

    else:

        registerFailedEvent(db,cursor,rfid)

```

En primer lugar esta función recibe como parámetros las variables necesarias para la consulta a la base de datos y también la clave que fue aproximada al lector de radiofrecuencia. Luego se realiza la consulta respectiva a la tabla “control_datos_usuarios_dj” que devuelve los datos de la persona a la cual está asignada la clave que fue aproximada al lector.

Si la consulta no devuelve resultados, se llamará a la función “registerFailedEvent()” que es la encargada de registrar el evento de intento de acceso fallido al laboratorio. Por el contrario si la consulta devolvió resultados es porque existe un usuario registrado que tiene asignada dicha clave, por lo cual se capturan los datos de ESTADO y el ID de ese usuario en dos variables distintas. Si el usuario presenta un estado ACTIVO quiere decir que estaría habilitado para ingresar al laboratorio, solo resta saber las franjas horarias que tiene asignadas el mismo para ver si en ese horario puede acceder. Para eso se llama a la función “checkTime()”, y luego se registra el evento de ingreso al laboratorio mediante la función “registerEvent()”. Ambas funciones serán analizadas luego con mayor profundidad.

4.1.3.4.5 Control de franjas horarias

Esta función es la que va a permitir realizar el último chequeo de datos para permitir o denegar el acceso al laboratorio, ya que en ella se realiza una consulta a la base de datos del sistema para ver los horarios en que un usuario específico tiene habilitado el ingreso. El código de la función es la siguiente:

```

defcheckTime(db,cursor,idUser):
    today = time.strftime("%w")
    mi_query ="SELECT desde_franjas,hasta_franjas FROM control_datos_usuarios_dj usr join
control_franjas_horarias_dj franjas

```

```

onusr.categoría_usuario_id=franjas.id_personal_franjas_idwhereusr.id='%i'andfranjas.dia_franjas_id
='%'%(idUser,today)
    cursor.execute(mi_query)
    users = cursor.fetchall()
    now = time.strftime("%X")
    for user in users:
        desde = user[0].replace(":", "")
        hasta = user[1].replace(":", "")
        ahora = now.replace(":", "")
        if int(ahora) >= int(desde) and int(ahora) <= int(hasta):
            openDoor()
        else:
            pass

```

Lo primero que realizamos es guardar el día (fecha) actual en una variable denominada “today”. Luego realizamos una consulta a la base de datos que devuelve los horarios (desde_franjas y hasta_franjas) que tiene un usuario específico (idUser) para entrar al laboratorio en el día actual (today). Por consiguiente guardamos en otra variable “now” la hora actual y luego sepáramos y cambiamos el formato de las variables para que puedan ser comparadas fácilmente (se eliminan los “:”).

Por último realizamos la comparación necesaria para ver si la hora actual está incluida dentro del intervalo [desde_franjas, hasta_franjas] que tiene el usuario involucrado. De ser exitosa esta comparación se llama a la función “openDoor()” para abrir la puerta del laboratorio y permitir el ingreso.

4.1.3.4.6 Registro de eventos

Esta función permite registrar en la base de datos un evento de acceso al laboratorio exitoso. Cada vez que un usuario que esté presente en la base de datos y su estado sea ACTIVO e intente ingresar al laboratorio, quedara registrado en una tabla específica. La cual luego puede ser consultada mediante el software de control web. El código de esta función es el siguiente:

```

defregisterEvent(db,cursor,user):
    idUser = user[0]
    firstName = user[1]
    lastName = user[2]
    mi_query ="INSERT INTO control_eventos_dj(id_usuario_eventos, nombres_eventos,
apellidos_eventos, fechayhora_eventos, lugar_eventos) VALUES
('%s','%s','%s',DATETIME('now','localtime'),'LAC')%(idUser,firstName,lastName)
    cursor.execute(mi_query)

```

```
db.commit()
```

Como vemos es una función muy simple que captura en distintas variables el id, nombre y apellido del usuario que accedió al laboratorio y luego guarda en la tabla “control_eventos_dj” esos datos y suma el horario actual y el lugar (que es un char constante). De esta manera queda registrado el evento de ingreso de un usuario específico al laboratorio.

4.1.3.4.7 Registro de intentos de acceso fallidos

Esta es otra función simple que, al igual que la vista anteriormente, se encarga de registrar en una tabla específica un evento en particular. En este caso ese evento se dará cuando se intenta acceder al laboratorio sin tener los permisos necesarios.

```
defregisterFailedEvent(db,cursor,rfid):
    if rfid[5:]<>"00000":
        mi_query ="INSERT INTO control_eventos_no_permitidos_dj(clave_eventos_no_permitido,
fechayhora_eventos_no_permitido, lugar_eventos_no_permitido) VALUES
('%s',DATETIME('now','localtime'),'LAC')%(rfid)
        cursor.execute(mi_query)
    db.commit()
```

Para poder capturar estos eventos vimos que existía un patrón que sucedía frecuentemente en el que el lector de radiofrecuencia capturaba códigos que no pertenecían al acercamiento de ningún tipo de tarjeta o llavero RFID. Estos códigos finalizaban siempre con 5 ceros consecutivos, por lo tanto se decidió ignorar esos códigos para solo registrar intentos de acceso mediante tarjetas o llaveros RFID que no estén en la base de datos.

Lo único que realiza la función es insertar en la tabla “control_eventos_no_permitidos_dj” la clave de la tarjeta que no está habilitada, la fecha y hora actual y el lugar al que se intentó acceder. Esta tabla podrá ser consultada también mediante el software de control web.

4.1.3.4.8 Captura de clave para agregar usuarios

Para poder agregar un nuevo usuario a la base de datos del sistema de acceso es necesario asignarle una clave que será la que pertenece a la tarjeta de seguridad que se le otorgara. Para poder capturar esa clave es necesaria esta función:

```
defcaptureCode(rfid):
    try:
        db = sqlite3.connect('/usr/src/web/database.sqlite')
        cursor = db.cursor()

        mi_query = "INSERT INTO control_captura_clave (id,clave_captura,lugar_captura) VALUES ('1','%s','LAC')"%rfid
        cursor.execute(mi_query)
        db.commit()
        time.sleep(5)

        mi_query="DELETEFROM control_captura_clave WHERE id='1'"
        cursor.execute(mi_query)
        db.commit()

    except IOError as e:
        pass
    except ValueError:
        pass
```

En primer lugar nos conectamos a la base de datos y luego creamos el cursor que nos va a permitir recorrer los registros que se obtienen de las consultas. Luego mediante una query se inserta la clave leída en la tabla “control_captura_clave”, la cual es una tabla que posee una sola fila creada específicamente para capturar la clave. Luego dormimos la función por 5 segundos para luego eliminar dicho registro. En esos 5 segundos se va a copiar el valor de la clave en un casillero específico dentro del software de control web para que se pueda continuar con el registro del nuevo usuario.

4.1.4 Testing

4.1.4.1 Lectura de tarjeta RFID

En este testing pondremos a prueba el script “control.py”, el cual se encarga de establecer la comunicación serial con el lector RFID, leer los datos y procesarlos. Además testearemos el conexiónado del lector con la placa Raspberry Pi mediante el circuito estabilizador de tensión.

4.1.4.1.1 Cuadro de análisis

Lectura de tarjeta RFID	
Requerimientos	R2
Propósito	Comprobar el funcionamiento correcto de la lectura del código de tarjeta de seguridad al aproximarla al lector RFID.
Entorno de Ejecución	El script realizado en Python será ejecutado en la placa Raspberry Pi.
Inicialización	Ejecutar el script “ control.py ”. Establecer la comunicación entre la placa Raspberry Pi y el lector RFID mediante el circuito estabilizador de tensión.
Finalización	Código de tarjeta de proximidad capturado.
Acciones	Ejecutar el script “ control.py ” que queda a la espera de un dato. Dicho script levanta la librería “ serial ” que es la que establece la comunicación con el lector.
Resultados	
Resultados Esperados	Visualización correcta de los 10 bits hexadecimales que codifican cada tarjeta de proximidad RFID.
Resultados Obtenidos	Al ejecutar el script y aproximar la tarjeta de seguridad se puede observar que se capturan correctamente los códigos de cada una de ellas.

Tabla 20: Cuadro de análisis de lectura de tarjeta RFID

4.1.4.1.2 Evidencia

Se utilizaron las siguientes tarjetas de proximidad RFID:

- Tarjeta 1: 28004332FA
- Tarjeta 2: 310094FC3D
- Tarjeta 3: 5000AA5DEC

Las tarjetas fueron aproximadas al lector RFID alternadamente en un total de 3 intentos por cada una de ellas.

```
root@raspberrypi:/usr/src# python control.py
Conexion serial exitosa
Clave capturada: 28004332FA
Clave capturada: 310094FC3D
Clave capturada: 5000AA5DEC
Clave capturada: 28004332FA
Clave capturada: 310094FC3D
Clave capturada: 5000AA5DEC
Clave capturada: 28004332FA
Clave capturada: 310094FC3D
Clave capturada: 5000AA5DEC
```

Ilustración 7: Evidencia lectura de tarjeta RFID

4.1.4.2 Acceso a usuarios permitido/denegado

En este testing pondremos a prueba el sistema de acceso de usuarios, el cual deberá permitir el ingreso a aquellos usuarios que estén registrados y activos en una base de datos dentro del sistema, y denegar el acceso a aquellos que no se encuentren en dichas condiciones.

4.1.4.2.1 Cuadro de análisis

Acceso a usuarios permitido/no permitido	
Requerimientos	R2 – R9
Propósito	Comprobar que se concede el acceso si la tarjeta de seguridad aproximada está registrada y habilitada para el ingreso. Caso contrario denegar dicho acceso.
Entorno de Ejecución	El script realizado en Python será ejecutado en la placa Raspberry Pi, al igual que la base de datos.
Inicialización	Ejecutar el script “ control.py ”. Establecer la comunicación entre la placa Raspberry Pi y el lector RFID mediante el circuito estabilizador de tensión. Ejecutar el servicio de base de datos “ SQLite3 ”
Finalización	Acceso permitido o denegado
Acciones	Ejecutar el script “ control.py ” que queda a la espera de un dato. Dicho script levanta la librería “ serial ” que es la que establece la comunicación con el lector y también permite conectarse a la base de datos “ database.sqlite ”.
Resultados	
Resultados Esperados	En el caso de un usuario registrado y activo deberá permitir el ingreso. En el caso de un usuario que no esté registrado o este en estado inactivo no deberá permitir el ingreso.
Resultados Obtenidos	Al ejecutar el software controlador y el script y luego aproximar la tarjeta de seguridad se puede observar que permite el ingreso a aquellos usuarios que se encuentra en condiciones para ello. También vemos que no se habilita el ingreso a aquellos que no presentan las condiciones necesarias.

Tabla 21: Cuadro de análisis de acceso a usuarios permitido/denegado

4.1.4.2.2 Evidencia

Se utilizaron las siguientes tarjetas de proximidad RFID:

- Tarjeta 1: 28004332FA -----> REGISTRADA Y ACTIVA
- Tarjeta 2: 310094FC3D -----> REGISTRADA E INACTIVA
- Tarjeta 2: 5000AA5DEC -----> NO REGISTRADA

Las tarjetas fueron aproximadas al lector RFID alternadamente.

```
root@raspberrypi:/usr/src# python control.py
Conexion serial exitosa
Clave capturada: 28004332FA
Conexion con Base de Datos database.sqlite exitosa
Usuario Registrado y Activo
#####
Clave capturada: 310094FC3D
Conexion con Base de Datos database.sqlite exitosa
Usuario Registrado y No Activo
#####
Clave capturada: 5000AA5DEC
Conexion con Base de Datos database.sqlite exitosa
Usuario No Registrado
#####
Clave capturada: 28004332FA
Conexion con Base de Datos database.sqlite exitosa
Usuario Registrado y Activo
#####
Clave capturada: 310094FC3D
Conexion con Base de Datos database.sqlite exitosa
Usuario Registrado y No Activo
#####
Clave capturada: 5000AA5DEC
Conexion con Base de Datos database.sqlite exitosa
```

Ilustración 8: Evidencia acceso a usuarios permitido/denegado

4.1.5 Conclusión

Desarrollamos un primer script en python cuya función era leer los datos proporcionados por la aproximación de la tarjeta al lector RFID, permitir o negar el ingreso y capturar la imagen. En un principio este script tenía un tiempo de respuesta elevado (aproximadamente 3 segundos), por lo cual decidimos mejorarlo. Por esta razón investigamos sobre el uso de hilos en python y lo aplicamos a nuestro script. De esta manera logramos reducir el tiempo de respuesta de manera considerable (menor a 1 segundo).

4.2 Iteración 2

4.2.1 Objetivos

En esta iteración se desarrolla el sistema de vigilancia para el laboratorio, el cual permite la visualización del exterior del mismo mediante una cámara web y capturar imágenes cuando se aproxima la tarjeta al lector RFID.

4.2.2 Diseño

En este apartado se describe cómo es el diseño del sistema de acceso de los usuarios en conjunto con el sistema de vigilancia. Se observa cómo se capturan las imágenes al registrarse un intento de acceso al laboratorio mediante la proximidad de una tarjeta de seguridad.

4.2.2.1 Requerimientos

REQUERIMIENTOS INVOLUCRADOS		
Número	Prioridad	Requerimiento
1	Alta	Se deberá poder visualizar el exterior del laboratorio mediante cámara web en tiempo real
3	Alta	Al registrar evento de ingreso de usuario se deberá realizar una captura de imagen
4	Alta	Las imágenes deben ser guardadas

Tabla 22: Requerimientos involucrados en el sistema de acceso con vigilancia

4.2.2.2 Diagrama de secuencia

En la siguiente ilustración podemos observar el diagrama de secuencia del sistema de acceso con vigilancia:

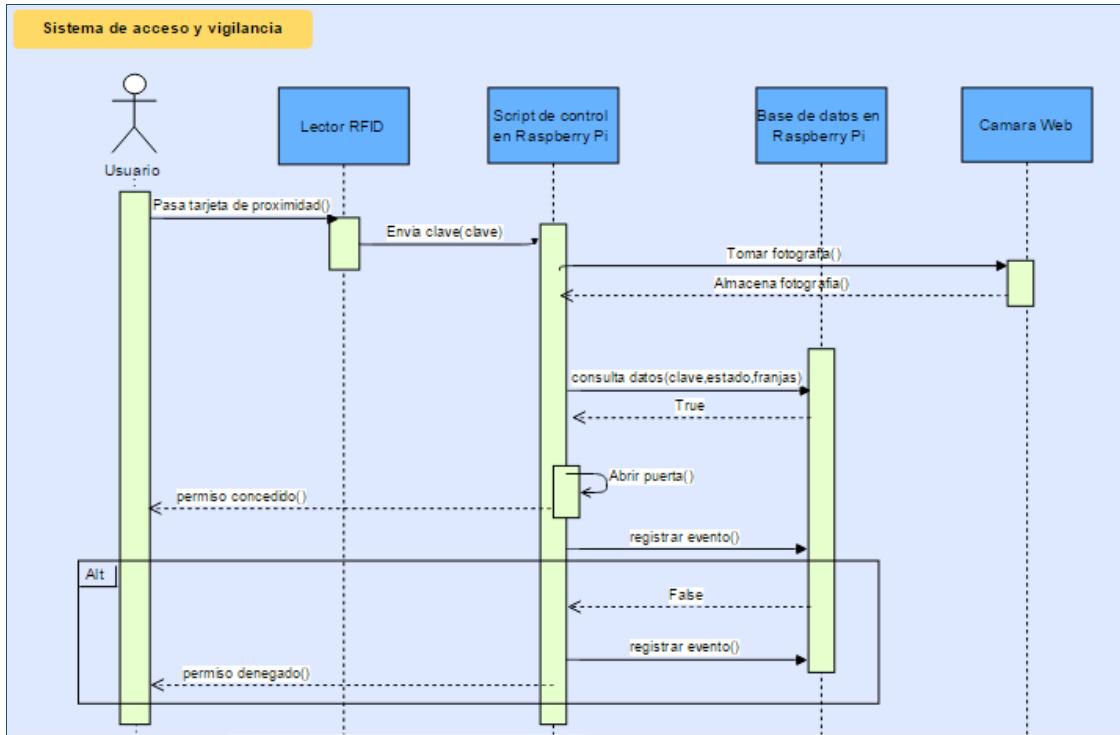


Ilustración 9: Diagrama de secuencia del sistema de acceso con vigilancia

4.2.3 Implementación

4.2.3.1 Instalación y configuración del software de vigilancia

Para realizar la vigilancia es necesario, además de una cámara web, un software que permita la visualización de la misma y que permita configurar ciertos aspectos importantes a tener en cuenta. Para ello utilizamos el software Motion.

El primer paso será conectar nuestra webcam a un puerto USB de la Raspberry, comprobando que la reconoce con el siguiente comando:

```
lsusb
```

4.2.3.1.1 Instalación Motion

Para instalar el software usaremos la línea de comandos, luego de logearse en la Raspberry:

```
sudo apt-get install motion
```

A continuación se instalaran varios paquetes en el proceso de instalación; Solo hay que tipar “y” para proceder con la instalación, la cual nos lleva un par de minutos.

El archivo de configuración de Motion se encuentra presente en el ANEXO A de este informe.

4.2.3.1.2 Ejecución Motion

Por ultimo ejecutamos el comando que nos permite inicializar el servicio de la siguiente manera:

```
sudo motion -n
```

Luego, si colocamos la IP de nuestra placa Raspberry Pi seguido del puerto de visualización de la cámara (en este caso el 8081) en un navegador podremos observar las imágenes capturadas por la cámara web:

```
http://[IP_de_la_Raspberry]:8081
```

Para acceder a la página de configuración del software debemos ingresar la siguiente dirección en el navegador:

```
http://[IP_de_la_Raspberry]:8080
```

4.2.3.2 Captura de imágenes

Para poder realizar capturas de imagen es necesario, en primer lugar, acceder a la función “*readRfid()*” dentro del archivo “**control.py**” y agregar las líneas que se encuentran en color rojo como vemos a continuación:

```
def readRfid(serialPort):  
    while 1:  
        userCode = serialPort.readline()  
        if (len(userCode) >6):  
            vectorRfid = []  
            vectorRfid = userCode.split(chr(13).encode('ascii'))
```

```

rfid = vectorRfid[0]

thpicture = threading.Thread(target=takePicture)

threads.append(thpicture)

thpicture.start()

return rfid

```

De esta manera creamos un hilo que nos permite llamar a la función “*takePicture()*” y realizar la captura de imagen en paralelo a la lectura de tarjeta de proximidad por parte del lector RFID.

A continuación vemos como se construye la función “*takePicture()*”:

```

def takePicture():

    foto = os.system("curl http://localhost:8080/0/action/snapshot")

```

Esta función lo único que realiza es una llamada a la dirección web que se encarga de manejar las acciones que puede realizar la cámara web a través del software **Motion**. Es decir, ingresa a la configuración de **Motion** para realizar la acción de tomar una captura de imagen instantáneamente.

4.2.4 Testing

4.2.4.1 Visualización de la cámara web en tiempo real

En este testing pondremos a prueba la visualización del exterior del laboratorio por medio de la cámara web. Para realizar esto se utiliza el software “*motion*” que es el encargado de controlar la cámara web. Este software abre un puerto específico (8081) que, integrado a la IP de la Raspberry Pi, permite visualizar imágenes en tiempo real.

4.2.4.1.1 Cuadro de análisis

Visualización de la cámara web en tiempo real	
Requerimientos	R1
Propósito	Comprobar el funcionamiento correcto de la visualización de la cámara web.
Entorno de Ejecución	Sistema operativo de la Raspberry Pi.
Inicialización	Conectar la cámara web al puerto USB de la placa Raspberry Pi. Ejecutar el software “ motion ”.
Finalización	Visualización en tiempo real del exterior del laboratorio.

Acciones	En un navegador web colocar la siguiente dirección: [http://IP_RASPBERRY:8081]
Resultados	
Resultados Esperados	Visualización correcta del exterior del laboratorio en tiempo real.
Resultados Obtenidos	En un comienzo, al ejecutar el software, se obtuvo una correcta visualización del exterior del laboratorio. Luego de ciertos intervalos de tiempo la cámara dejaba de funcionar por problemas de drivers.

Tabla 23: Cuadro de análisis de visualización de cámara web en tiempo real

4.2.4.1.2 Evidencia

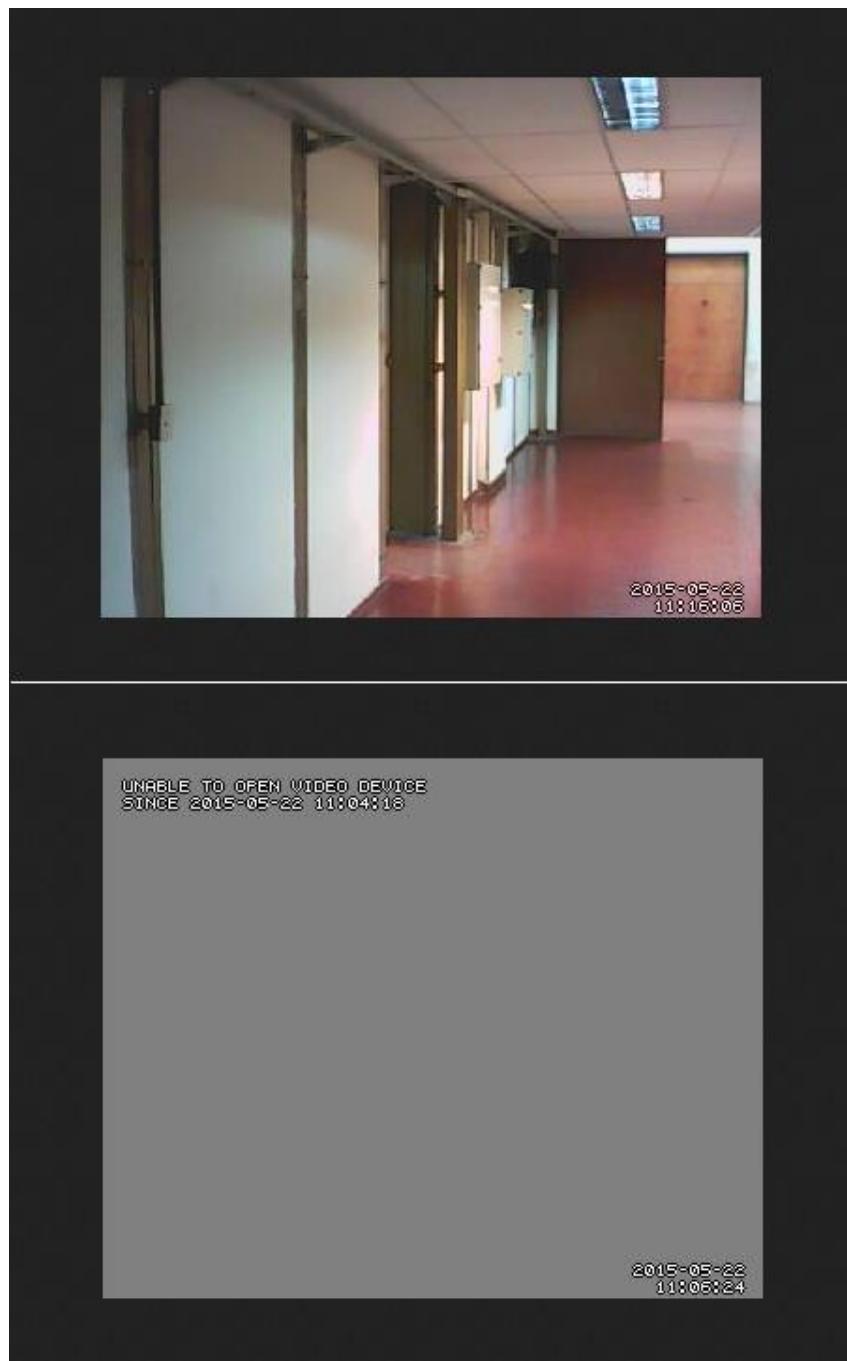


Ilustración 10: Evidencia visualización de cámara web en tiempo real

Debido a los problemas de drivers que tenía la cámara web se tuvo que reemplazar por una nueva que tenga mayor compatibilidad con la placa Raspberry Pi.

4.2.4.2 Captura y almacenamiento de imágenes

En este testing pondremos a prueba la captura de imágenes por parte de la cámara web en el momento en que el lector RFID lee el código de una tarjeta de proximidad. También se verá cómo se realiza el almacenamiento de imágenes en el directorio correspondiente. Para realizar esto se

utiliza el script “**control.py**” que se encarga de generar la orden para capturar la imagen y el software “**motion**” que es el encargado de controlar la cámara web.

4.2.4.2.1 Cuadro de análisis

Captura y almacenamiento de imágenes	
Requerimientos	R3 – R4
Propósito	Comprobar el funcionamiento correcto de la captura y almacenamiento de imágenes por medio de la cámara web al aproximar una tarjeta de seguridad al lector RFID.
Entorno de Ejecución	El script realizado en Python será ejecutado en la placa Raspberry Pi, al igual que el software controlador de la cámara web.
Inicialización	Ejecutar el script “ control.py ”. Ejecutar el software “ motion ”. Conectar la cámara web al puerto USB de la placa Raspberry Pi. Establecer la comunicación entre la placa Raspberry Pi y el lector RFID mediante el circuito estabilizador de tensión.
Finalización	Imagen almacenada en un directorio.
Acciones	Ejecutar el script “ control.py ” que queda a la espera de un dato. Dicho script levanta la librería “ serial ” que es la que establece la comunicación con el lector. Ejecutar el software “ motion ” que inicia el servidor para la cámara web.
Resultados	
Resultados Esperados	Imagen capturada en el instante en que se aproxima una tarjeta al lector RFID y posterior guardado de la misma a un directorio especificado con anterioridad.
Resultados Obtenidos	Al ejecutar el software controlador y el script y luego aproximar la tarjeta de seguridad se puede observar que se realizan las capturas de imagen de manera correcta. Luego se guardan exitosamente en el directorio correspondiente.

Tabla 24: Cuadro de análisis de captura y almacenamiento de imágenes

4.2.4.2.2 Evidencia

Se utilizaron las siguientes tarjetas de proximidad RFID:

- Tarjeta 1: 28004332FA
- Tarjeta 2: 310094FC3D

Las tarjetas fueron aproximadas al lector RFID alternadamente en un total de 3 intentos por cada una de ellas.

```
root@raspberrypi:/usr/src# python control.py
Conexion serial exitosa
Clave capturada: 310094FC3D
snapshot for thread 0
Done
Fotografia capturada
Fotografia guardada en: /usr/src/web/login/static/photo
Clave capturada: 28004332FA
snapshot for thread 0
Done
Fotografia capturada
Fotografia guardada en: /usr/src/web/login/static/photo
Clave capturada: 310094FC3D
snapshot for thread 0
Done
Fotografia capturada
Fotografia guardada en: /usr/src/web/login/static/photo
Clave capturada: 28004332FA
snapshot for thread 0
Done
Fotografia capturada
Fotografia guardada en: /usr/src/web/login/static/photo
Clave capturada: 310094FC3D
snapshot for thread 0
Done
Fotografia capturada
Fotografia guardada en: /usr/src/web/login/static/photo
Clave capturada: 28004332FA
snapshot for thread 0
Done
Fotografia capturada
Fotografia guardada en: /usr/src/web/login/static/photo
```

Ilustración 11: Evidencia captura y guardado de imágenes

4.2.5 Conclusión

En un comienzo se logró capturar imágenes en el momento deseado y visualizar el exterior del laboratorio en tiempo real. Luego de un tiempo observamos que el driver controlador de la cámara web dejaba de funcionar ocasionándonos un problema para la captura de imagen y la visualización del exterior. Además ésta cámara no detectaba la luz infrarroja, lo cual era una gran problema cuando había ausencia de luz.

Luego de una exhaustiva investigación, llegamos a la conclusión de que el sistema operativo Raspbian no era compatible con el driver de la cámara web FaceCam 1020, por lo tanto buscamos información sobre la compatibilidad de diferentes cámaras web en la placa Raspberry Pi. Esto nos llevó a adquirir una nueva cámara web (Logitech C170) con la cual se solucionaron todos los problemas.

4.3 Iteración 3

4.3.1 Objetivos

En esta iteración se desarrolla el software administrador y de manejo del sistema, el cual está alojado en un servidor web. Dicho software permite acceder a todas las funcionalidades del sistema tales como: ver usuarios registrados, ver eventos de ingreso, visualizar cámara web, abrir la puerta de forma remota, agregar usuarios al sistema, programar franjas horarias, ver las capturas de imagen realizadas por la cámara web, capturar clave de tarjetas de proximidad, entre otras.

4.3.2 Diseño

En este apartado describimos como es el proceso para agregar un nuevo usuario al sistema de control de acceso y además como se realiza la captura de la clave de la tarjeta de proximidad, que será única para cada usuario.

4.3.2.1 Requerimientos

REQUERIMIENTOS INVOLUCRADOS		
Numero	Prioridad	Requerimiento
9	Alta	El software deberá poder administrar usuarios y claves
10	Alta	Deberá existir un sistema de captura de clave de usuario

Tabla 25: Requerimientos involucrados en el sistema de registro y captura de claves

4.3.2.2 Diagrama de secuencia

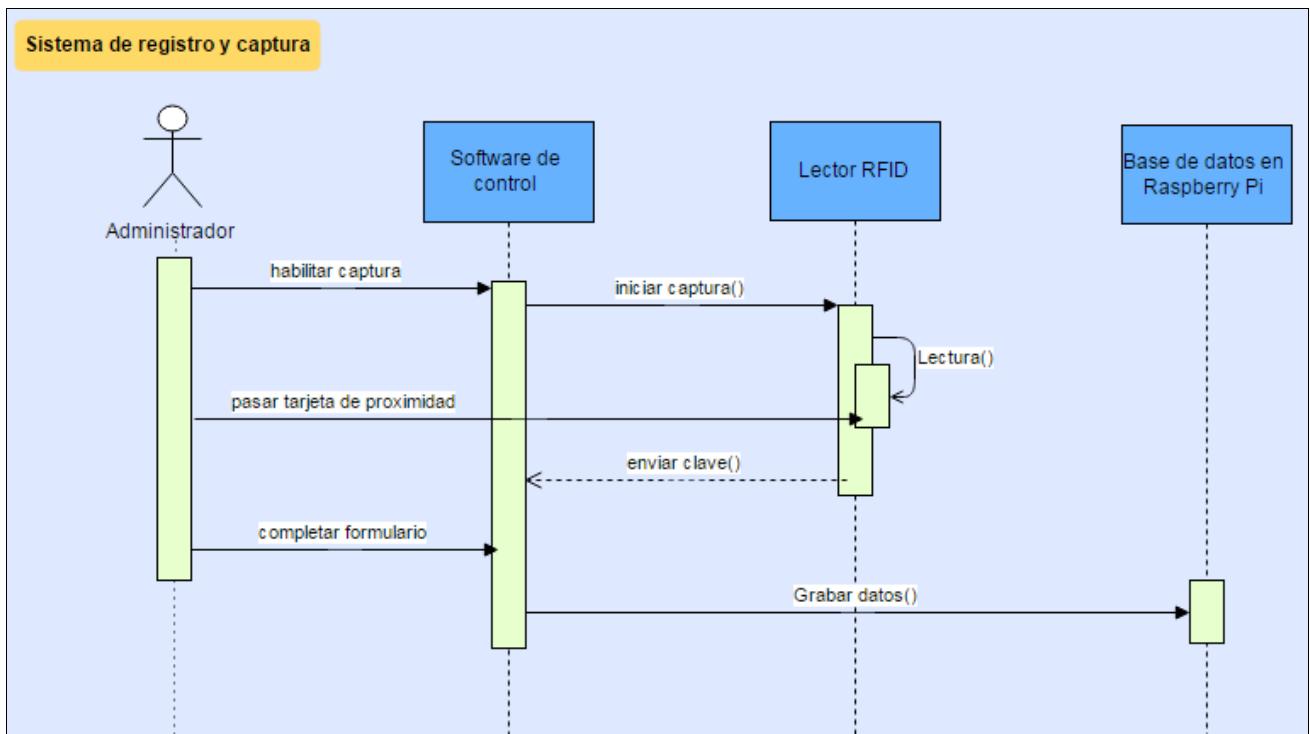


Ilustración 12: Diagrama de secuencia del sistema de registro y captura

4.3.3 Implementación

4.3.3.1 Desarrollo del entorno Web

El entorno web será el software de control del sistema de acceso y seguridad. A partir de él se podrá tanto administrar el sistema como también observar los eventos y reportes que se generan debido al uso del mismo. Existe una serie de funcionalidades que serán descriptas en este apartado, detallando como fueron generadas y como utilizarlas de manera correcta.

Para realizar el entorno web utilizamos un framework denominado DJANGO. Este provee una estructura MVT (Model Template View), por lo cual esa será la manera de describir cada funcionalidad en esta sección.

4.3.3.1.1 Instalación de Django

En primer lugar debemos instalar PIP, el cual es un sistema de gestión de paquetes usado para instalar y gestionar paquetes de software escritos en Python. Para ello colocamos el siguiente comando en la consola:

```
apt-get install python-pip
```

Ahora si estamos en condiciones de comenzar con la instalación de Django. Django es un framework hecho en Python, por lo tanto se necesita que hayas previamente instalado Python (2.6 o 2.7). En este caso en la distribución Raspbian ya estaba previamente instalado. Hay que tener en cuenta que Django no funciona con Python 3.0 actualmente, debido a incompatibilidades con el intérprete de Python.

Para comenzar el proceso de instalación es necesaria la utilización del siguiente comando:

```
pip install django
```

Esto instalara Django en el directorio de instalación de Python denominado “site-packages”.

A continuación instalaremos también el modulo “xlwt”, el cual nos permitirá crear una hoja de datos en Excel cuando se decida exportar las tablas de eventos o personal. Igualmente esto será descripto con mayor detalle en una sección posterior de este informe. Para instalar el modulo solo hace falta escribir los siguiente en la consola:

```
pip install xlwt
```

4.3.3.1.2 Inicio del proyecto web

Para crear nuestro primer proyecto, abrimos una terminal, nos ubicamos en la carpeta en donde queremos crear nuestro proyecto y digitamos:

```
django-admin.py startproject controlacceso
```

Esta instrucción creará dos directorios con el nombre del proyecto (en este caso: controlacceso) y 5 archivos distribuidos de la siguiente manera:

- manage.py
- controlacceso
 - __init__.py
 - settings.py
 - urls.py
 - wsgi.py

Para ver que el proyecto está funcionando en la terminal debemos escribir:

```
python manage.py runserver
```

Al ejecutar esa instrucción debemos visualizar un resultado como el siguiente:

```
System check identified no issues (0 silenced).
January 14, 2015 - 15:02:35
Django version 1.7.1, using settings 'login.settings'
Starting development server at http://0.0.0.0:80/
Quit the server with CONTROL-C.
```

Ilustración 13: Inicio del servidor Django

Abrimos el navegador web la dirección <http://0.0.0.0:80> y debemos una pantalla similar a la siguiente:

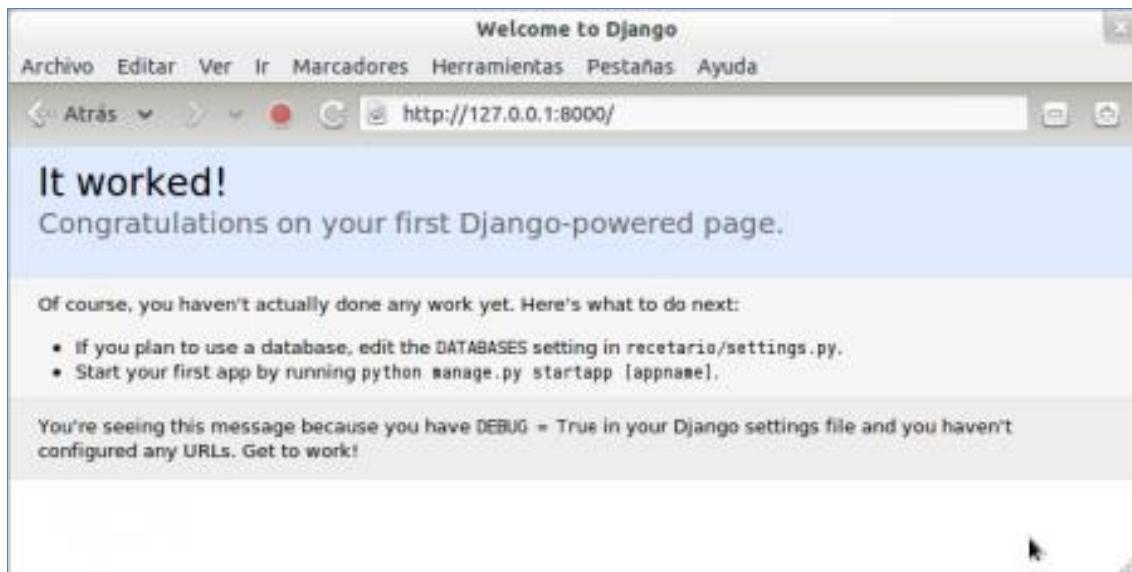


Ilustración 14: Interfaz web Django I

En el caso de que nos salga un error porque el puerto asignado está en uso sólo debemos indicar que puerto usaremos para lanzar el servicio, por ejemplo si se desea usar el puerto 8888 entonces se tendría que digitar:

```
python manage.py runserver 8888
```

En este caso usaríamos la dirección `http://127.0.0.1:8888/` para visualizar en el navegador.

4.3.3.1.3 Inicio de la aplicación

Cada proyecto necesita de aplicaciones donde se puedan gestionar los modelos y las vistas. Para crear nuestra primer aplicación, desde la terminal y en la carpeta del proyecto, debemos digitar:

```
python manage.py startapp control
```

Esto creará un directorio y cuatro archivos más, lo que nos dejaría con una estructura de archivos como esta:

- `manage.py`
- `controlacceso`

- `__init__.py`
- `settings.py`
- `urls.py`
- `wsgi.py`
- control
 - `__init__.py`
 - `models.py`
 - `test.py`
 - `views.py`

4.3.3.1.4 Configuración del proyecto

Una parte muy importante del proyecto es el archivo `settings.py`, este archivo permite configurar la conexión a la base de datos, la zona horaria, el idioma, los directorios principales del proyecto, las aplicaciones del proyecto, entre otras cosas.

Aprender a configurar este archivo permite optimizar el funcionamiento del proyecto, veremos las instrucciones principales a configurar:

4.3.3.1.4.1 Codificación de caracteres

Nuestro idioma está lleno de caracteres especiales como las tildes y las eñes que son las más comunes, la primera sugerencia para manejar esto eficientemente en django debemos agregar la siguiente línea al archivo `settings.py`

```
#encoding:utf-8
```

4.3.3.1.4.2 Ruta del proyecto

Una configuración importante es configurar la ruta del proyecto, esto permitirá lanzar la aplicación desde cualquier directorio y mover el proyecto a cualquier computador con Django instalado. Para ello debemos escribir las siguientes líneas en el archivo `settings.py`:

```
# Identificando la ruta del proyecto
import os
RUTA_PROYECTO = os.path.dirname(os.path.realpath(__file__))
```

Si no se configura la ruta del proyecto, cada vez que se cambia de directorio o de PC, se tendrá que cambiar las rutas de las plantillas, archivos estáticos y directorio de subida de contenido de los usuarios.

4.3.3.1.4.3 Configuración de la base de datos

También podemos configurar la conexión a la base de datos según nuestras necesidades, Django soporta de manera predeterminada la conexión con postgresql, mysql, sqlite3 y oracle. En nuestro proyecto usaremos sqlite3.

Debemos buscar la siguiente sección del archivo:

```
DATABASES ={
    'default': {
        'ENGINE': 'django.db.backends.', # Add 'postgresql_psycopg2', 'mysql', 'sqlite3' or 'oracle'.
        'NAME': '',                      # Or path to database file if using sqlite3.
        'USER': '',                      # Not used with sqlite3.
        'PASSWORD': '',                  # Not used with sqlite3.
        'HOST': '',                      # Set to empty string for localhost. Not used with sqlite3.
        'PORT': ''                       # Set to empty string for default. Not used with sqlite3.
    }
}
```

Y dejarla de la siguiente manera:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3', # Add 'postgresql_psycopg2', 'mysql', 'sqlite3' or
        'oracle'.
```

```
'NAME': '/usr/src/web/database.sqlite',      # Or path to database file if using sqlite3.  
'USER': "",          # Not used with sqlite3.  
'PASSWORD': "",      # Not used with sqlite3.  
'HOST': "",          # Set to empty string for localhost. Not used with sqlite3.  
'PORT': "",          # Set to empty string for default. Not used with sqlite3.  
}  
}
```

4.3.3.1.4.4 Zona Horaria

Django permite configurar la zona horaria del proyecto, la lista de zonas horarias disponibles se pueden encontrar en la wikipedia. Para configurar debemos buscar lo siguiente:

```
TIME_ZONE = 'America/Chicago'
```

Configuraremos el proyecto en la zona horaria de Cordoba/Argentina, así que lo modificare de esta forma:

```
TIME_ZONE = 'America/Argentina/Cordoba'
```

4.3.3.1.4.5 Configuración del idioma

Django también permite configurar el idioma que usará de manera predeterminada para su funcionamiento, para configurar esto debemos buscar lo siguiente:

```
LANGUAGE_CODE = 'en-us'
```

Se puede consultar la lista de idiomas disponibles para adecuarlo a nuestras necesidades. En nuestro caso configuraremos como español de Argentina:

```
LANGUAGE_CODE = 'es-AR'
```

4.3.3.1.4.6 Aplicaciones instaladas

Un proyecto en Django necesita de aplicaciones, algunas ya vienen configuradas de manera predeterminada. En nuestro proyecto usaremos la aplicación de administración y su documentación, estas ya vienen construidas, y también nuestra primera aplicación creada líneas arriba, llamada control. Para habilitar estas aplicaciones debemos buscar la siguiente sección que se encuentra casi al final del archivo **settings.py**

```
INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    # Uncomment the next line to enable the admin:
    # 'django.contrib.admin',
    # Uncomment the next line to enable admin documentation:
    # 'django.contrib.admindocs',
)
```

Y modificarlas de la siguiente manera:

```
INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
```

```
'django.contrib.staticfiles',
'django.contrib.admin',
'django.contrib.admindocs',
'control',
)
```

4.3.3.1.4.7 Directorios de plantillas y static

Es necesario crear un directorio para ubicar los templates como así también un directorio que contenga todos los archivos y datos que sean estáticos (imágenes, fuentes, scripts de java, etc). Para ello nos ubicamos dentro de nuestra carpeta del proyecto y creamos ambos directorios (con el comando mkdir) y damos los permisos necesarios (mediante el comando chmod 777).

Una vez creados ambos directorios debemos avisarle a django de la existencia de estos, por lo tanto nos dirigimos nuevamente al archivo settings.py y agregamos las siguientes líneas:

En el caso del directorio de Templates:

```
TEMPLATE_DEBUG =True

ALLOWED_HOSTS = []

TEMPLATE_DIRS = (os.path.join(ruta_proyecto, 'templates'),)
```

En el caso del directorio Static:

```
STATIC_URL = '/static/'

STATICFILES_DIRS = (os.path.join(ruta_proyecto, 'static'),)
```

4.3.3.1.4.8 Redireccionamiento en Login

Por ultimo debemos configurar la carpeta a la que se va a redireccionar en caso de acceso permitido o negativo cuando se realiza el logueo del usuario. Para ello es necesario agregar las siguientes líneas en el archivo settings.py:

```
# Redirect when login is correct.
```

```
LOGIN_REDIRECT_URL = "/home"

# Redirect when login is not correct.

LOGIN_URL = '/'
```

4.3.3.1.4.9 Direcciones URL del proyecto

Para poder visualizar los cambios que hicimos y la interfaz administrativa de Django, aún falta modificar un archivo más, este es: urls.py. Este archivo contiene lo siguiente:

```
from django.conf.urls import patterns, include, url

# Uncomment the next two lines to enable the admin:
# from django.contrib import admin
# admin.autodiscover()

urlpatterns = patterns('',
    # Examples:
    # url(r'^$', 'control.views.home', name='home'),
    # url(r'^control/', include('control.foo.urls')),

    # Uncomment the admin/doc line below to enable admin documentation:
    # url(r'^admin/doc/', include('django.contrib.admindocs.urls')),

    # Uncomment the next line to enable the admin:
    # url(r'^admin/', include(admin.site.urls)),
```

```
)
```

Debemos dejarlo de esta manera:

```
from django.conf.urls import patterns, include, url  
  
from django.contrib import admin  
  
admin.autodiscover()  
  
urlpatterns = patterns(''  
    url(r'^admin/doc/', include('django.contrib.admindocs.urls')),  
    url(r'^admin/', include(admin.site.urls)),  
)
```

4.3.3.1.5 Ejecución del proyecto

Una vez que tenemos todo listo y configurado, debemos nuevamente iniciar el servidor de desarrollo que tiene el proyecto. Ya hicimos esto al principio, solo debemos digitar desde la terminal nuevamente (dentro del directorio del proyecto):

```
python manage.py runserver
```

Lucirá de esta manera:

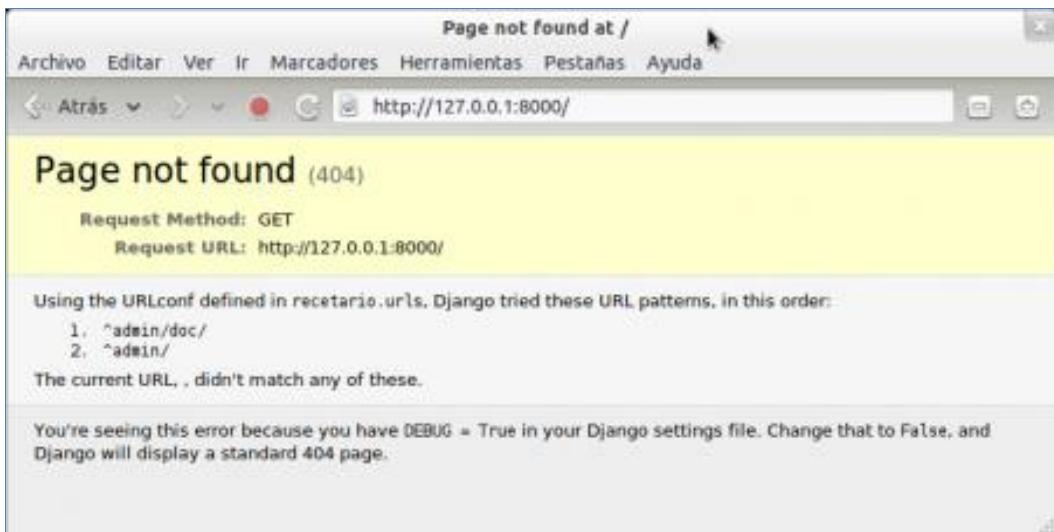


Ilustración 15: Interfaz web Django II

Nos muestra un error 404, luego veremos el por qué. Para ingresar a la interfaz administrativa que ya viene construida con Django, ingresaremos a la dirección: [http://\[IP_RASPBERRY\]/admin](http://[IP_RASPBERRY]/admin), en donde debemos poner el nombre del superusuario y su respectiva contraseña, creados anteriormente (líneas arriba) con la opción syncdb.

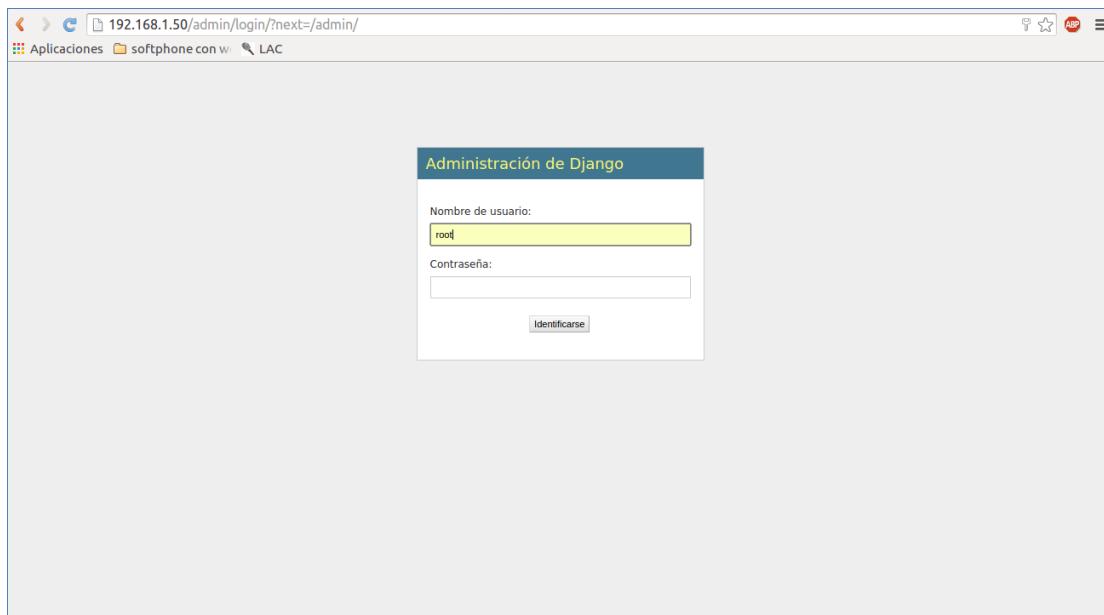


Ilustración 16: Login de la interfaz administrativa de Django

Si todo fue correcto debemos visualizar la interfaz administrativa:

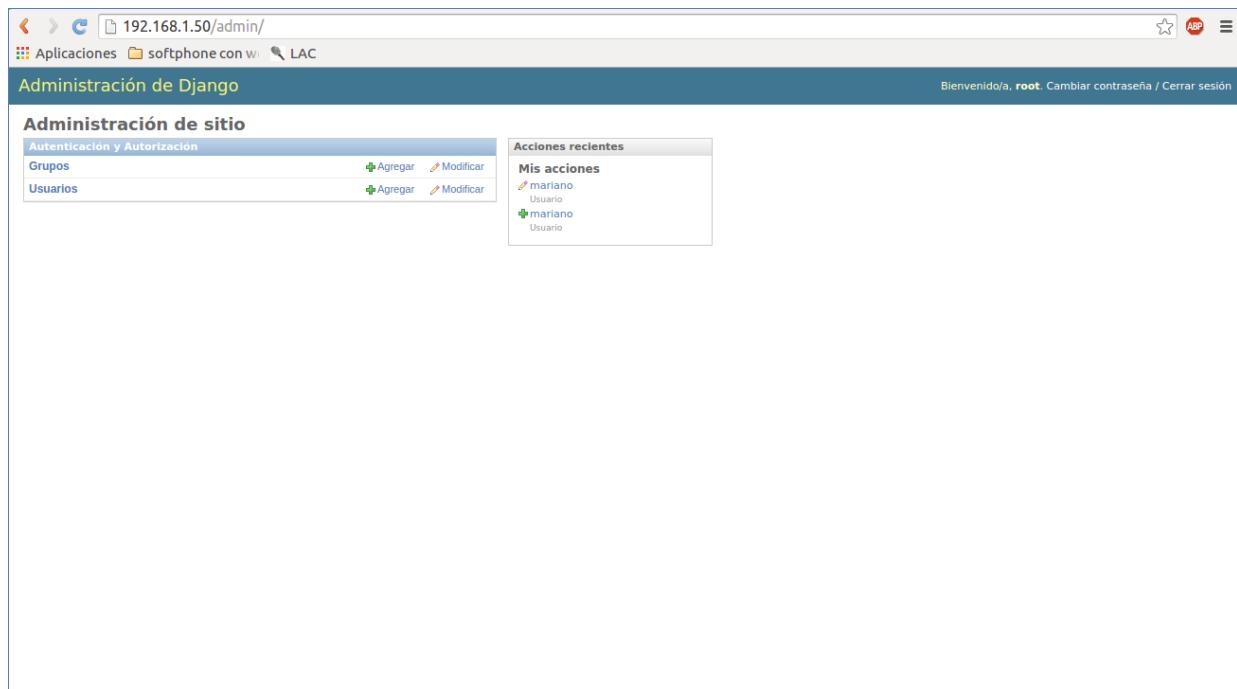


Ilustración 17: Interfaz administrativa de Django

Nuestro proyecto ya se encuentra configurado para continuar con la construcción de los modelos.

4.3.3.1.6 Creación de la base de datos

Hasta el momento no se ha creado la base de datos o las tablas predeterminadas del proyecto, solo se ha configurado los parámetros de conexión. Para crear la base de datos, debemos digitar desde la terminal o ventana de comandos, la siguiente instrucción (recordar que debemos estar en la carpeta de proyecto para que todo se realice correctamente):

```
python manage.py syncdb
```

Esta instrucción deberá mostrar el siguiente resultado:

```

Creating tables ...
Creating table auth_permission
Creating table auth_group_permissions
Creating table auth_group
Creating table auth_user_user_permissions
Creating table auth_user_groups
Creating table auth_user
Creating table django_content_type
Creating table django_session
Creating table django_site
Creating table django_admin_log

You just installed Django's auth system, which means you don't have any superusers defined.
Would you like to create one now? (yes/no): 

```

Ilustración 18: Sincronización con la base de datos

Hay una pregunta que debemos responder, se refiere a la creación de un superusuario (un administrador del proyecto), para lo cual respondemos: yes (en caso de responder negativamente, no podremos usar inmediatamente el administrador predeterminado de Django). Luego de ello completamos la información que nos solicita.

Al finalizar ya estará creada la base de datos, como en este proyecto se usara sqlite3, deberá aparecer un archivo nuevo llamado database.sqlite, este contiene las tablas y los datos iniciales del proyecto.

Una vez realizado este paso podemos empezar a crear nuestras tablas específicas del sistema de control de acceso mediante el archivo “models.py”. Es una de las 3 piezas fundamentales en las que se basa Django (**Model Template View**).

A continuación haremos una descripción de las tablas que serán creadas para este proyecto.

4.3.3.1.6.1 Tabla de usuarios

Esta tabla es generada para contener todos los datos relacionados a los usuarios que van a ser parte del sistema de acceso, es decir aquellos que están o han estado en la lista de usuarios con ingreso permitido al laboratorio.

Cada fila de la tabla será asignada a un usuario específico. A continuación se describen los atributos (columnas) que tendrá cada usuario dentro de nuestra tabla:

- Id: es el número que se le va a asignar a cada usuario. Es auto incremental y único (clave primaria)
- nombres_usuario: En este campo estarán los nombres de pila del usuario. Se completa con caracteres
- apellidos_usuario: En este campo estarán los apellidos o el apellido del usuario. Se completa con caracteres.
- dni_usuario: Se completa con el número del documento de identidad nacional. Es único y el campo es un Integer.
- teléfono_usuario: Aquí se deberá colocar el teléfono de contacto del usuario.

- Dirección usuario: Este campo se completara con la dirección de la vivienda en donde reside el usuario actualmente. El campo es de caracteres.
- localidad_usuario: Aquí se completara con la ciudad donde reside el usuario. El campo es de caracteres.
- email_usuario: En este campo se deberá colocar el email de contacto del usuario. El campo es de caracteres.
- clave_usuario: Este campo será completado con la clave de la tarjeta o llavero RFID que se le dará a cada usuario. Para obtener esta clave será necesario un sistema de captura de clave de tarjeta que será descripto posteriormente, así como también una tabla específica para poder realizarlo. El campo es de caracteres.
- estado_usuario: Este es un campo que podrá tener 2 valores posibles únicamente, ACTIVO o INACTIVO. De esta manera se podrá dar de baja a cualquier usuario que esté presente en la base de datos sin necesidad de eliminarlo, o viceversa. El campo es de caracteres.
- tarjeta_usuario: En este sistema de control de acceso existe la posibilidad de contar tanto con tarjetas RFID como con llaveros RFID, por lo cual este campo sirve para especificar una de las 2 opciones posibles. El campo es de caracteres
- fecha_alta_usuario: Este campo se autocompletara al momento de confirmar el alta del usuario, colocando la fecha actual en la que fue habilitado el usuario. Es un campo DateTime
- categoría_usuario: La categoría hace referencia al tipo de usuario, es decir, a la “especialidad” o cargo que posea. Esto servirá para crear grupos de usuarios y así establecer distintas franjas horarias que afecten a determinados grupos. Es una clave foránea a otra tabla.

Para crear la tabla es necesario ir al archivo **models.py** y agregar las siguientes líneas:

```
class datos_usuarios_dj(models.Model):

    id= models.AutoField(primary_key=True)

    nombres_usuario = models.CharField(max_length=30)

    apellidos_usuario = models.CharField(max_length=30)

    dni_usuario = models.IntegerField()

    telefono_usuario = models.CharField(max_length=30)

    direccion_usuario = models.CharField(max_length=30)

    localidad_usuario = models.CharField(max_length=30)

    email_usuario = models.CharField(max_length=30)
```

```
clave_usuario = models.CharField(max_length=30)

estado_usuario = models.CharField(max_length=30)

tarjeta_usuario = models.CharField(max_length=30)

fecha_alta_usuario = models.DateTimeField()

categoria_usuario = models.ForeignKey(Personal_dj)
```

Por último para realizar la migración de la tabla resta utilizar el siguiente comando:

```
python manage.py syncdb
```

En el caso de realizar modificaciones y querer actualizar la tabla deberá utilizarse también el siguiente comando:

```
python manage.py sqlall control
```

4.3.3.1.6.2 Tabla de captura de claves

Esta tabla servirá para almacenar el código-clave de la tarjeta o llavero RFID cuando es capturado por el lector de radiofrecuencia. También se almacenara el nombre del lugar en donde fue leída dicha clave. En este caso existe solo un establecimiento, que es el Laboratorio de Arquitectura de Computadoras, por lo cual el nombre siempre será el mismo (LAC).

La clave será leída y almacenada por un tiempo determinado, ya que luego de confirmar el alta del usuario esta clave será borrada de la base de datos. Esto permite que sea un sistema más seguro y además que la tabla nunca crezca en tamaño, ya que siempre tendrá como máximo 1 tupla.

Las columnas de la tabla son las siguientes:

- clave_captura: En este campo se almacenara la clave que es capturada por el lector RFID. El campo es de caracteres.
- lugar_captura: En este campo se almacenara siempre el nombre “LAC” ya que es el único establecimiento que maneja el sistema de control en este proyecto.

Para crear la tabla es necesario ir al archivo **models.py** y agregar las siguientes líneas:

```
class Captura_clave(models.Model):  
  
    clave_captura = models.CharField(max_length=50)  
  
    lugar_captura = models.CharField(max_length=50)
```

Por ultimo para realizar la migración de la tabla resta utilizar el siguiente comando:

```
python manage.py syncdb
```

En el caso de realizar modificaciones y querer actualizar la tabla deberá utilizarse también el siguiente comando:

```
python manage.py sqlall control
```

4.3.3.1.6.3 Tabla de categoría de usuario

Esta tabla contiene las diferentes identificaciones que puede tener un usuario, es decir, las diferentes categorías a las cuales puede asignarse un usuario. Esto sirve para categorizar y separar grupos y así poder asignar franjas horarias diferentes a cada uno de ellos.

Por ejemplo al agregar un usuario puede asignársele una categoría de “Estudiante” o “Docente” (entre otros) y cada categoría tendrá distintos privilegios en cuanto a los horarios de entrada al laboratorio.

Pueden existir uno, muchos o ningún usuario dentro de una categoría. Pero un usuario puede estar dentro de una sola categoría.

Las columnas de la tabla son las siguientes:

- id: Este campo es auto incremental y contiene el número de identificación de cada categoría. El campo es clave primaria.
- identificacion_personal: Aquí se guarda el nombre que se le asigna a cada categoría. El campo es de caracteres.

Para crear la tabla es necesario ir al archivo **models.py** y agregar las siguientes líneas:

```
classPersonal_dj(models.Model):  
    id= models.AutoField(primary_key=True)  
    identificacion_personal = models.CharField(max_length=30)
```

Por último para realizar la migración de la tabla resta utilizar el siguiente comando:

```
python manage.py syncdb
```

En el caso de realizar modificaciones y querer actualizar la tabla deberá utilizarse también el siguiente comando:

```
python manage.py sqlall control
```

4.3.3.1.6.4 Tabla de días semanales

Esta tabla esta creada para contener la lista de días que contiene una semana, lo cual será de ayuda para cuando se utilicen franjas horarias para permitir el ingreso de los usuarios.

La tabla solo tendrá 2 columnas y el número de filas será igual a la cantidad de días existentes en una semana, es decir, siete.

Las columnas de la tabla son las siguientes:

- id: Este campo es auto incremental y contiene el número de identificación de cada día de la semana. El campo es clave primaria.
- nombre_dias: Aquí se guarda el nombre de cada día de la semana. El campo es de caracteres.

Para crear la tabla es necesario ir al archivo **models.py** y agregar las siguientes líneas:

```
classDias_semanales_dj(models.Model):  
    id= models.AutoField(primary_key=True)  
    nombre_dias = models.CharField(max_length=10)
```

Por último para realizar la migración de la tabla resta utilizar el siguiente comando:

```
python manage.py syncdb
```

En el caso de realizar modificaciones y querer actualizar la tabla deberá utilizarse también el siguiente comando:

```
python manage.py sqlall control
```

4.3.3.1.6.5 Tabla de franjas horarias

Esta tabla es la que permite manejar distintas franjas horarias para las categorías establecidas en una de las tablas vistas anteriormente (tabla de categoría de usuario).

Se pueden establecer muchas franjas horarias distintas para un mismo día, así como también se puede optar por no habilitar el ingreso de ciertos usuarios para un día completo.

Las columnas de la tabla son las siguientes:

- id: Este campo es auto incremental y contiene el número de identificación de cada franja horaria. El campo es clave primaria.
- Id_personal_franjas: Es una clave foránea a la tabla de categorías de usuarios. Muestra el ID de cada categoría, es decir, la categoría a la que va a ser asignada la franja horaria actual.
- dia_franjas: Es una clave foránea a la tabla de días semanales. Muestra el ID de cada día, es decir, el día al que va a ser asignada la franja horaria actual.
- desde_franjas: Este campo contiene la hora (en formato HH:MM:SS) en la que va a comenzar la franja horaria actual. Es decir el horario de inicio de la franja horaria. El campo es de tipo Time.
- hasta_franjas: Este campo contiene la hora (en formato HH:MM:SS) en la que va a finalizar la franja horaria actual. Es decir el horario de finalización de la franja horaria. El campo es de tipo Time.

Para crear la tabla es necesario ir al archivo **models.py** y agregar las siguientes líneas:

```
class Franjas_horarias_dj(models.Model):  
    id = models.AutoField(primary_key=True)  
    id_personal_franjas = models.ForeignKey(Personal_dj)  
    dia_franjas = models.ForeignKey(Dias_semanales_dj)  
    desde_franjas = models.TimeField()
```

```
hasta_franjas = models.TimeField()
```

Por último para realizar la migración de la tabla resta utilizar el siguiente comando:

```
python manage.py syncdb
```

En el caso de realizar modificaciones y querer actualizar la tabla deberá utilizarse también el siguiente comando:

```
python manage.py sqlall control
```

4.3.3.1.6.6 Tabla de eventos

Esta tabla servirá para almacenar los eventos de ingreso al laboratorio por parte de los usuarios habilitados.

Cada vez que se produzca un ingreso al laboratorio mediante el acercamiento de una tarjeta o llavero RFID, y el resultado sea exitoso, se producirá un evento que será registrado en la tabla en cuestión.

Las columnas de la tabla son las siguientes:

- id: Este campo es auto incremental y contiene el número de identificación de cada evento producido. El campo es clave primaria.
- Id_usuario_eventos: Aquí se almacena el número de identificación del usuario que ingreso al laboratorio. El campo es de tipo Integer.
- nombres_eventos: En este campo se almacenan los nombres del usuario que ingreso al laboratorio. El campo es de caracteres.
- apellidos_eventos: En este campo se almacenan los apellidos del usuario que ingreso al laboratorio. El campo es de caracteres.
- fechayhora_eventos: Este campo contiene la fecha (en formato DD:MM:AAAA) y la hora (en formato HH:MM:SS) en la que el usuario ingreso al laboratorio. El campo es de tipo DateTime.
- lugar_eventos: Este campo contendrá siempre el mismo nombre, que es el del laboratorio de arquitectura de computadoras (LAC). El campo es de caracteres.

Para crear la tabla es necesario ir al archivo **models.py** y agregar las siguientes líneas:

```
class Eventos_dj(models.Model):  
    id= models.AutoField(primary_key=True)  
    id_usuario_eventos= models.IntegerField()  
    nombres_eventos = models.CharField(max_length=50)  
    apellidos_eventos = models.CharField(max_length=50)  
    fechayhora_eventos = models.DateTimeField()  
    lugar_eventos= models.CharField(max_length=50)
```

Por último para realizar la migración de la tabla resta utilizar el siguiente comando:

```
python manage.py syncdb
```

En el caso de realizar modificaciones y querer actualizar la tabla deberá utilizarse también el siguiente comando:

```
python manage.py sqlall control
```

4.3.3.1.6.7 Tabla de eventos de ingresos no permitidos

Esta tabla existe para almacenar los eventos generados cuando existe un intento de ingreso al laboratorio con una tarjeta o llavero RFID que no está habilitado para dicho ingreso.

En este caso no existirá un usuario relacionado con la clave de la tarjeta, ya que si ocurre tal evento quiere decir que no existe esa clave asignada a ningún usuario de nuestra tabla de usuarios. Por lo tanto únicamente se guardara la clave de la tarjeta, así como también la fecha, hora y lugar donde ocurrió el evento.

Las columnas de la tabla son las siguientes:

- id: Este campo es auto incremental y contiene el número de identificación de cada evento producido. El campo es clave primaria.
- clave_eventos_no_permitido: Aquí se almacena la clave de la tarjeta que fue acercada al lector RFID para poder ingresar al laboratorio sin éxito. El campo es de caracteres.
- fechayhora_eventos_no_permitido: Este campo contiene la fecha (en formato DD:MM:AAAA) y la hora (en formato HH:MM:SS) en la que el usuario intento ingresar al laboratorio. El campo es de tipo DateTime.
- lugar_eventos_no_permitido: Este campo contendrá siempre el mismo nombre, que es el del laboratorio de arquitectura de computadoras (LAC). El campo es de caracteres.

Para crear la tabla es necesario ir al archivo **models.py** y agregar las siguientes líneas:

```
class Eventos_no_Permitidos_dj(models.Model):  
  
    id= models.AutoField(primary_key=True)  
  
    clave_eventos_no_permitido = models.CharField(max_length=50)  
  
    fechayhora_eventos_no_permitido = models.DateTimeField()  
  
    lugar_eventos_no_permitido= models.CharField(max_length=50)
```

Por último para realizar la migración de la tabla resta utilizar el siguiente comando:

```
python manage.py syncdb
```

En el caso de realizar modificaciones y querer actualizar la tabla deberá utilizarse también el siguiente comando:

```
python manage.py sqlall control
```

4.3.3.1.6.8 Tabla de eventos web

Esta tabla existe para almacenar los eventos generados cuando se abre la puerta del laboratorio desde la página web.

En este caso quedara registrado el usuario que ingreso al software de control (entorno web) y permitió el ingreso del usuario externo, así como también la fecha, hora y lugar en donde se realizó.

Las columnas de la tabla son las siguientes:

- id: Este campo es auto incremental y contiene el número de identificación de cada evento producido. El campo es clave primaria.
- Id usuario eventos web: Aquí se almacena el número de identificación del usuario que permitió el ingreso del usuario externo al laboratorio. Es decir el usuario que estaba logueado en el software de control cuando se envió la señal de apertura de puerta del laboratorio. El campo es de tipo Integer.
- nombres eventos web: En este campo se almacena el nombre del usuario que estaba logueado en el software de control cuando se envió la señal de apertura de puerta del laboratorio. El campo es de caracteres.
- via eventos web: En este campo contiene siempre la palabra “WEB” haciendo alusión al medio por el cual fue permitido el ingreso del usuario externo. El campo es de caracteres.
- fechayhora eventos web: Este campo contiene la fecha (en formato DD:MM:AAAA) y la hora (en formato HH:MM:SS) en la que el usuario ingreso al laboratorio. El campo es de tipo DateTime.
- lugar eventos web: Este campo contendrá siempre el mismo nombre, que es el del laboratorio de arquitectura de computadoras (LAC). El campo es de caracteres.

Para crear la tabla es necesario ir al archivo **models.py** y agregar las siguientes líneas:

```
classWeb_eventos_dj(models.Model):  
  
    id= models.AutoField(primary_key=True)  
  
    id_usuario_eventos_web= models.IntegerField()
```

```
nombres_eventos_web = models.CharField(max_length=50)

via_eventos_web = models.CharField(max_length=50)

fechayhora_eventos_web = models.DateTimeField()

lugar_eventos_web= models.CharField(max_length=50)
```

Por último para realizar la migración de la tabla resta utilizar el siguiente comando:

```
python manage.py syncdb
```

En el caso de realizar modificaciones y querer actualizar la tabla deberá utilizarse también el siguiente comando:

```
python manage.py sqlall control
```

4.3.3.1.7 Secciones y funcionalidades del Software de control web

En este apartado nos focalizaremos en describir las funcionalidades que presenta el Software de control Web, así como también como fue desarrollada cada una de ellas.

Previamente vimos cómo crear las tablas de la base de datos (**Model**), por lo tanto ya están creados los datos que vamos a necesitar visualizar o modificar en cada una de las secciones del software.

Aquí veremos cómo se diseña la apariencia de la página web (**Templates**), particularmente de cada sección, la cual será a través de plantillas. También veremos cómo se comporta cada página de nuestra Web (**Views**), que será a través de funciones escritas en Python.

Por ultimo veremos cómo se relacionan las plantillas con las funciones en Python, lo cual será mediante un archivo de Django llamado *urls.py*.

4.3.3.1.7.1 Sección Login

Esta sección es la primera que conocemos al ingresar al Software de control Web. Es la pantalla de inicio en la cual se solicita el usuario y contraseña (que fueron generados previamente en la sección *admin* que genera Django automáticamente).

Esta pantalla es particular, ya que para crearla no es necesario hacer nada en el archivo **views.py** (a diferencia de todas las otras secciones que describiremos luego). Lo único que haremos será la plantilla (**login.html**) ya que las funcionalidades de logueo son propias del framework Django.

La apariencia se genera a través de código html y llamando a métodos propios del framework. Solo necesitamos que existan casilleros donde completar con la información de *usuario* y *contraseña* y luego un botón que realice un POST confirmando que se completaron los datos necesarios. Esto se realiza mediante el siguiente formulario:

```
<form class="navbar-form navbar-right" action="{% url 'django.contrib.auth.views.login' %}" method="post" accept-charset="utf-8">

    {% csrf_token %}

    <div class="form-group">

        <input class="form-control" type="text" placeholder="Username" name="username">

    </div>

    <div class="form-group">

        <input class="form-control" type="password" placeholder="Password" name="password">

    </div>

    {% csrf_token %}

    <button type="submit" class="btn btn-default">Login</button>

</form>
```

Las funciones de logueo son heredadas y utilizadas en el template de la siguiente manera:

```
{% if user.is_active %}

    <script language="JavaScript" type="text/javascript">
```

```
window.location="/home";  
  
        </script>  
  
{% endif %}
```

Como vemos en la sección del código html último, utilizamos una función del framework en la que se consulta si el usuario está activo, lo cual devuelve un Boolean (True o False). En caso de que la respuesta sea True entra al script y re direcciona a la página “*home*”, es decir, permite el acceso.

Por ultimo deberemos asignar las funciones de login a una URL. Para ello necesitamos mapearlo mediante el archivo **urls.py** de la siguiente manera:

```
from django.conf.urls import patterns, include, url  
  
from django.contrib.auth.views import login, logout  
  
from django.contrib import admin  
  
urlpatterns = patterns(''  
  
    url(r'^$', login, {'template_name': 'login.html'}, name="login"),  
  
)
```

Solo se muestra la línea que respecta a este apartado. El código fuente completo puede verse en el anexo de “CÓDIGOS” de este informe.

La apariencia de esta sección es la siguiente:



Ilustración 19: Sección login de página web

4.3.3.1.7.2 Sección Home

Esta sección es una de las principales del sistema de control de acceso y vigilancia. En ella se podrán realizar funciones fundamentales tanto para la vigilancia, como para el acceso y la comunicación.

En primer lugar para poder visualizar la página Home es necesario dirigirse al archivo **views.py** y agregar la siguiente función escrita en Python:

```
@login_required()  
  
def home(request):  
  
    return render_to_response('home.html', {'user': request.user},  
    context_instance=RequestContext(request))
```

Lo único que hacemos en esta función será utilizar el shortcut de Django para así facilitarnos la vida a la hora de retornar un web response.

Una vez realizada la función debemos mapearla con el template. Para ello nos dirigimos al archivo **urls.py** y agregamos lo siguiente:

```

from django.conf.urls import patterns, include, url

from django.contrib.auth.views import login, logout

from django.contrib import admin

urlpatterns = patterns('',
    url(r'^home$', 'control.views.home', name='home'),
)

```

A partir de aquí todos los cambios que haremos en la plantilla html serán vistos automáticamente en nuestra página web. Pero en el caso de que se quieran agregar funcionalidades se deberá seguir recurriendo al archivo *views.py* y seguir mapeando (de ser necesario) con el template a través del archivo *urls.py*.

Principalmente en la sección *Home* se podrá visualizar en tiempo real la cámara web. Para ello es necesaria simplemente una línea en la plantilla *home.html* como veremos a continuación:

```
<center><imgsrc="http://192.168.1.50:8081"></center>
```

Esta línea permite visualizar aquello que se encuentre en la ruta 192.168.1.50:8081, siendo:

- 192.168.1.50: Dirección IP de la Raspberry Pi.
- 8081: Puerto asignado para la visualización de la cámara web por parte del software Motion.

Otra de las funcionalidades que presenta esta pantalla es la de poder abrir la puerta del laboratorio mediante el envío de una señal. Aquí se deberá crear un botón en la plantilla *home.html*, lo cual se hará mediante un formulario, ya que es necesario utilizar el método POST:

```
<formaction="/openDoor"method="post">{% csrf_token %}
```

```

<center><input type="submit" name="open" value="Abrir Puerta" class="btn btn-default" style="background:#01DF74"></center>

</form>

```

Como vemos en el formulario, una vez que se pulsa el botón “Abrir Puerta” este nos dirige a través de un método POST a /openDoor. Para detallar lo que se va a hacer una vez presionado dicho botón es necesario dirigirse al archivo *views.py* y agregar las siguientes líneas:

```

@login_required()

def openDoor(request):

    foto = os.system("curl http://localhost:8080/0/action/snapshot")

    GPIO.output(3,True)

    time.sleep(3)

    GPIO.output(3,False)

    username = request.user.username

    id_user= User.objects.get(username=username).pk

    new =
Web_eventos_dj(id_usuario_eventos_web=id_user,nombres_eventos_web=username, via_eventos_web="WEB",
fecha_yhora_eventos_web=datetime.now(),lugar_eventos_web="LAC")

    new.save()

    return HttpResponseRedirect("/home")

```

Como podemos observar en esta función, en primer lugar se realiza una captura de imagen por la cámara web a través de una función del software de vigilancia Motion. Esto es para poder visualizar mediante una imagen a la persona que está por ingresar al laboratorio.

Luego se envía un pulso a un pin digital de la Raspberry Pi para permitir el ingreso del usuario externo (True), se duerme la función por 3 segundos (tiempo que dispone el usuario para ingresar), y luego se envía un pulso al mismo pin digital para bloquear la señal (False).

A continuación obtenemos el nombre del usuario que esta logueado en el software de control web, ya que será necesario para escribirlo luego en una tabla de la base de datos.

Por ultimo grabamos los datos necesarios en la tabla Web_eventos_dj para generar el evento de ingreso de usuario externo mediante envío de señal por Web. Una vez realizado esto se redirecciona nuevamente a la pagina */home*.

Lo único que queda es realizar el mapeo de la función con el template. Para ello nos dirigimos al archivo *urls.py* y agregamos la siguiente linea:

```
from django.conf.urls import patterns, include, url

from django.contrib.auth.views import login, logout

from django.contrib import admin

urlpatterns = patterns('',
    url(r'^openDoor', 'control.views.openDoor', name='openDoor'),
)
```

La apariencia de esta página queda de la siguiente manera:

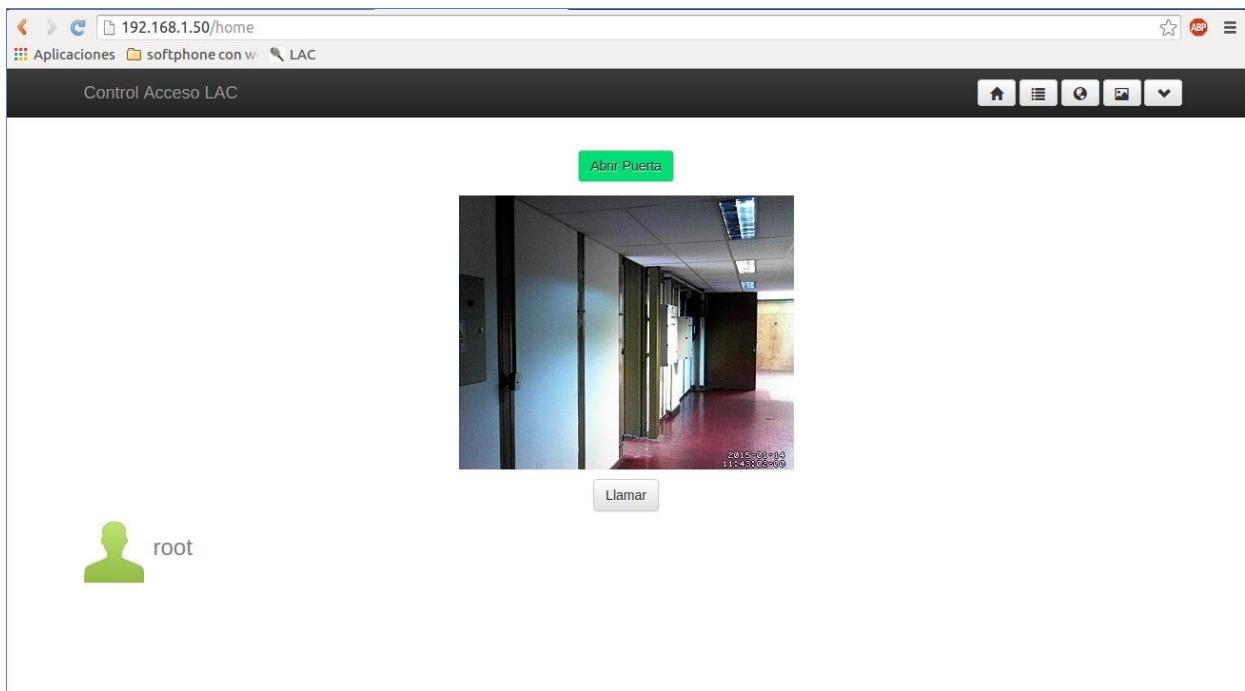


Ilustración 20: Sección home de página web

Como podemos observar en la ilustración 31 existe un botón “Llamar” que será descripto en la siguiente iteración.

4.3.3.1.7.3 Sección Usuarios

En la sección Usuarios podremos observar la lista completa usuarios que están registrados en el sistema de control, como así también los atributos y características de cada uno (Nombre, Apellido, DNI, Estado, Clave asignada, etc). Aquí estarán presentes tanto los usuarios ACTIVOS como los INACTIVOS, es decir, todos los que estén presentes en la base de datos.

En primer lugar veremos la parte de código de la plantilla “tablaUsuarios.html” con la que se definirá la apariencia de la tabla:

```
<table class="table table-hover table-bordered">
<thead>
<tr>
    <th><center>Nombres</center></th>
    <th><center>Apellidos</center></th>
    <th><center>DNI</center></th>
    <th><center>Telefono</center></th>
```

```

<th><center>Direccion</center></th>
<th><center>Localidad</center></th>
<th><center>Email</center></th>
<th><center>Clave</center></th>
<th><center>Estado</center></th>
<th><center>Tarjeta</center></th>
<th><center>Fecha Alta</center></th>
<th><center>Categoria</center></th>
<th><center>Acciones</center></th>
</tr>
</thead>

```

En esta parte de código vemos como creamos el encabezado de la tabla, es decir, los títulos que tendrá cada una de sus columnas. Luego seguimos con la creación del contenido de la tabla de la siguiente manera:

```

<tbody>
{% for usuario in usuarios %}
<tr>
    <tdalign='center'><h6>{{ usuario.nombres_usuario }}</h6></td>
    <tdalign='center'><h6>{{ usuario.apellidos_usuario }}</h6></td>
    <tdalign='center'><h6>{{ usuario.dni_usuario }}</h6></td>
    <tdalign='center'><h6>{{ usuario.telefono_usuario }}</h6></td>
    <tdalign='center'><h6>{{ usuario.direccion_usuario }}</h6></td>
    <tdalign='center'><h6>{{ usuario.localidad_usuario }}</h6></td>
    <tdalign='center'><h6>{{ usuario.email_usuario }}</h6></td>
    <tdalign='center'><h6>{{ usuario.clave_usuario }}</h6></td>
    <tdalign='center'><h6>{{ usuario.estado_usuario }}</h6></td>
    <tdalign='center'><h6>{{ usuario.tarjeta_usuario }}</h6></td>
    <tdalign='center'><h6>{{ usuario.fecha_alta_usuario }}</h6></td>
    <tdalign='center'><h6>{{ usuario.categoria_usuario.identificacion_personal }}</h6></td>
    <td><a href="/editarUsuario/{{usuario.id}}">Editar</a><br/><a href="/borrar/{{usuario.id}}" onclick="return confirm('¿Desea borrar el registro?')">Eliminar</a></td>
</tr>
{% empty %}
<tr><tdcolspan="4">No hay usuarios registrados</td></tr>
{% endfor %}
</tbody>
</table>

```

Aquí vemos como mediante un FOR vamos capturando y mostrando los datos de los usuarios presentes en la base de datos, y de no encontrar usuarios en la base de datos mostrara un cartel de “No hay usuarios registrados”.

A continuación veremos cómo se obtienen los datos de los usuarios mediante una función en el archivo *views.py*.

```
@login_required()  
  
def mostrarTablaUsuarios(request):  
  
    usuarios = datos_usuarios_dj.objects.all()  
  
    return render_to_response('tablaUsuarios.html',{'usuarios': usuarios })
```

Como vemos en la función “mostrarTablaUsuarios” se capturan todos los datos presentes en la tabla *datos_usuarios_dj* y se guardan en una variable denominada *usuarios*, que es la usada en la plantilla “tablaUsuarios.html” para recuperar los datos de los usuarios en la base de datos.

Por ultimo resta mapear la función presente en *views.py* con la URL de la página en cuestión, para ello nos dirigimos al archivo *urls.py* y agregamos la siguiente línea:

```
from django.conf.urls import patterns, include, url  
  
from django.contrib.auth.views import login, logout  
  
from django.contrib import admin  
  
urlpatterns = patterns('',  
  
    url(r'^tablaUsuarios$', 'control.views.mostrarTablaUsuarios', name='tablaUsuarios'),  
  
)
```

También podremos desde esta pantalla eliminar un usuario en particular de la base de datos o modificar sus atributos y características, como se puede ver en el código fuente de la plantilla “tablaUsuarios.html” ubicado más arriba. En la última columna vemos como se agregan las opciones “Editar” y “Eliminar”. Para ver lo que hacen estas opciones debemos dirigirnos nuevamente al archivo *views.py* agregar las siguientes funciones:

En el caso de la función para eliminar usuario:

```
@login_required()
def eliminar_usuario(request, id_usuario):
    usuario=datos_usuarios_dj.objects.get(id=id_usuario)
    usuario.delete()
    return HttpResponseRedirect("/tablaUsuarios")
```

En la función para eliminar usuario tomamos el id del usuario que se pasó como parámetro y luego obtenemos los datos que pertenecen a ese id específico. Luego eliminamos los datos de la base de datos y retornamos a la página “tablaUsuarios”. Por ultimo debemos mapear, como es costumbre, la función con la URL. Nos dirigimos al archivo *urls.py* y agregamos la siguiente línea:

```
from django.conf.urls import patterns, include, url

from django.contrib.auth.views import login, logout

from django.contrib import admin

urlpatterns = patterns('',
    url(r'^borrar/(?P<id_usuario>\d+)$', 'control.views.eliminar_usuario'),
)
```

En el caso de la función para editar usuarios:

```
@login_required()
def editarUsuario(request, idUsuario):
    usuario=datos_usuarios_dj.objects.get(id=idUsuario)
    identPersonals = Personal_dj.objects.all()
    if request.method=="POST":
        datos = request.POST
        nombre = datos['nombre']
        apellido = datos['apellido']
        dni = datos['dni']
        telefono = datos['telefono']
```

```

        direccion = datos['direccion']
        localidad = datos['localidad']
        email = datos['email']
        clave = datos['clave']
        estado = datos['estado']
        tarjeta = datos['tarjeta']
        categoria = datos['categoria']

        usuarioUpdate =
datos_usuarios_dj.objects.filter(id=idUsuario).update(nombres_usuario= nombre,apellidos_usuario=
apellido,dni_usuario=dni,telefono_usuario=telefono,direccion_usuario=direccion,localidad_usuario=l
ocalidad,email_usuario=email,clave_usuario=clave,estado_usuario=estado,tarjeta_usuario=tarjeta,cat
egoria_usuario_id=categoria)

        return HttpResponseRedirect("/tablaUsuarios")

    return
render_to_response("editarUsuario.html", {"formularios":usuario,'identPersonals':identPersonals},co
ntext_instance=RequestContext(request))

```

El caso de la función para editar un usuario es un poco más complejo. En primera instancia obtenemos los datos del usuario y la categoría a la que pertenece (son dos tablas distintas) a partir del id del usuario que fue pasado como parámetro. Una vez que se clickea el botón “Editar” nos dirigimos a una nueva pantalla en la que aparecerá un formulario con los datos actuales y la disponibilidad para modificarlos, como podemos ver a continuación:

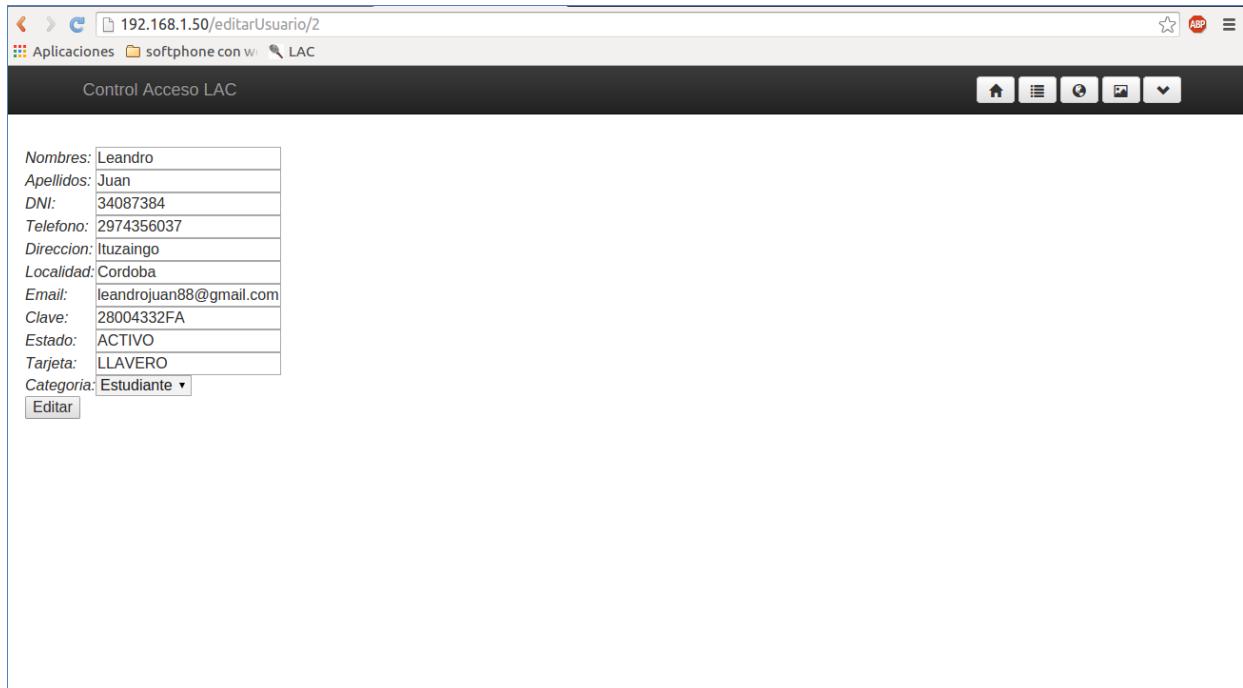


Ilustración 21: Sección editar usuarios de página web

Para poder seguir la descripción de esta función es necesario contar con parte del código fuente de la plantilla “editarUsuario.html”:

```





```

Como vemos aquí se crea una tabla en la cual se dispone de casilleros en donde aparecen los datos actuales que tienen los usuarios dentro de la base de datos. Esos datos son modificables y al momento de clicar en el botón “Editar” se envía una señal POST a la función para confirmar los datos. Si volvemos a observar la función *editarUsuario* dentro del archivo *views.py* que está presente más arriba en este informe vemos como se guardan los datos en diferentes variables y luego se actualiza la base de datos mediante un “update” para que las modificaciones queden realizadas efectivamente.

Por ultimo resta mapear la función *editarUsuario* con la URL en cuestión de la siguiente manera:

```

from django.conf.urls import patterns, include, url

from django.contrib.auth.views import login, logout

from django.contrib import admin

urlpatterns = patterns('',
    url(r'^editarUsuario/(?P<idUsuario>\d+)$', 'control.views.editarUsuario'),
)

```

La última funcionalidad que presenta esta sección es que la tabla de usuarios podrá ser exportada a una hoja de datos de Excel de ser necesario. Para ello agregamos el siguiente *form* en la plantilla “*tablaUsuarios.html*”:

```

<form action="/exportar" method="post">{% csrf_token %}

    <center><button type="submit" class="standar">Exportar a Excel</button></center>

</form>

```

Como podemos observar se trata de un botón llamado “Exportar a Excel” que nos direcciona a través de un método POST a la url */exportar*. Para poder agregarle funcionalidad a ese botón es necesario agregar la siguiente función al archivo *views.py*:

```

@csrf_exempt
def exportar (request):
    headers = ('Book', 'Author')
    data = []
    data = tablib.Dataset(*data, headers=headers)
    books = datos_usuarios_dj.objects.all()
    for book in books:
        data.append(book)
    response = HttpResponse(data.xls, content_type='application/vnd.ms-excel; charset=utf-8')

```

```

response['Content-Disposition'] = "attachment; filename=export.xls"
return response

```

Esta función se encarga de realizar la conversión de datos desde la tabla “datos_usuarios_dj” para almacenarlos en un libro de Excel y guardarlos bajo el nombre de “export.xls”.

En última instancia mapeamos la función con su respectiva URL en el archivo urls.py de la siguiente manera:

```

from django.conf.urls import patterns, include, url

from django.contrib.auth.views import login, logout

from django.contrib import admin

urlpatterns = patterns('',
    url(r'^exportar', 'control.views.exportar', name='exportar'),
)

```

La apariencia de esta sección quedara de la siguiente manera:

Nombres	Apellidos	DNI	Telefono	Direccion	Localidad	Email	Clave	Estado	Tarjeta	Fecha Alta	Categoría	Acciones
Leandro	Juan	34087384	2974356037	Ituzaingo	Cordoba	leandrojuan88@gmail.com	28004332FA	ACTIVO	LLAVERO	26 Nov. 2014 19:45:10	Estudiante	Editar Eliminar
Eduardo	Sufan	34774105	3515918467	Publica A 554	Cordoba	eduardosufan333@gmail.com	5000AA7B9E	ACTIVO	TARJETA	26 Nov. 2014 19:45:10	Estudiante	Editar Eliminar
Gaston	Lucero	36358550	3516637339	Miguel Victoria 2623	Cordoba	gaberrini@hotmail.com	440098F7CB	ACTIVO	LLAVERO	26 Nov. 2014 19:45:10	Estudiante	Editar Eliminar
Denis	Reinoso	34253991	3512650513	Santiago Temple 21 7D	Cordoba	reinoso.denis@gmail.com	52005490EC	ACTIVO	LLAVERO	26 Nov. 2014 19:45:10	Estudiante	Editar Eliminar
Renzo	Pisetta	34441188	156815828	G.Velez 3957	Cordoba	renzopisetta@gmail.com	5000AA2855	ACTIVO	TARJETA	19 Nov. 2014 19:45:10	Estudiante	Editar Eliminar
Sofia	Lujan	34130341	3515919682	Tycho Brahe 5118	Cordoba	solujan@gmail.com	78004580B0	ACTIVO	LLAVERO	19 Nov. 2014 19:45:10	Estudiante	Editar Eliminar
Marcelo	Cebollada	16014016	3515169530	El guardamonte 306	Cordoba	mcebollada@gmail.com	5000AA2887	ACTIVO	TARJETA	19 Nov. 2014 19:45:10	Estudiante	Editar Eliminar
Mariano	Dominguez	34908760	157157848	Cajamarca 1642	Cordoba	mardom4164@gmail.com	310094FC3D	ACTIVO	LLAVERO	5 Dic. 2014 17:55:25	Estudiante	Editar Eliminar

Ilustración 22: Sección ver usuarios de página web

4.3.3.1.7.4 Sección Agregar Usuario

Esta sección permite poder agregar nuevos usuarios al sistema de control de acceso y seguridad, es decir, poder habilitar a un usuario para que ingrese al laboratorio y asignarle una clave para su tarjeta personal de seguridad, entre otras cosas.

En primer lugar debemos crear un *form* con método POST que contenga una tabla y un botón para confirmar los datos. La tabla será con el estilo de formulario como vimos en la tabla para editar usuarios en la sección anterior, con la diferencia que esta vez los casilleros estarán vacíos y deberán ser completados (en algunos casilleros existirán opciones que deberán elegirse, sin posibilidad de completarse manualmente).

A continuación veremos parte del código de la plantilla “agregarUsuario.html” en el que vamos a obviar la creación de la tabla ya que es similar a lo realizado en la plantilla “editarUsuario.html”:

```
<formaction="/agregarUsuario"method="POST"onsubmit="return validacion(this)">{% csrf_token %}

<tablealign="center">

EN ESTE LUGAR IRIA EL CODIGO PARA GENERAL LOS CASILLEROS DE LA TABLA

</table>

<palign="center"><inputtype="submit" value="Grabar"style="margin-top:10px"></p>

</form>
```

Como podemos observar, una vez que clicamos en el botón “Grabar” recurre a una función denominada “validación” y luego nos re direcciona a la url /agregarUsuario. A continuación vemos como fue agregada esta función dentro de la plantilla:

```
<script>

    functionvalidacion(form) {
        var documento = form.dni.value
        var phone = form.telefono.value
        var checkIntDocumento = documento %1
        var checkIntPhone = phone %1

        if (checkIntDocumento !=0 || documento.length<5){
            alert('Formato incorrecto en campo DNI. Complete nuevamente');
            returnfalse;
        }

        if (checkIntPhone !=0){
            alert('Formato incorrecto en campo Telefono. Complete nuevamente');
        }
    }
</script>
```

```

        returnfalse;
    }
}
</script>
```

La función *validación()* es la encargada de comprobar si los datos de los casilleros de DNI y Telefono fueron completados de manera correcta, es decir que sean datos de tipo Integer y que tengan una cantidad de números mínimos de base. De no cumplirse alguno de los dos saldrá una alerta indicando que deben completarse nuevamente y se recargara la página.

Para poder grabar los datos correctamente en la tabla correspondiente de la base de datos es necesario acudir nuevamente al archivo *views.py* y agregar la siguiente función:

```

@csrf_protect
defaddUser(request):
    identPersonals = Personal_dj.objects.all()
    if request.method =='POST':
        datos = request.POST
        nombre = datos['nombre']
        apellido = datos['apellido']
        dni = datos['dni']
        telefono = datos['telefono']
        direccion = datos['direccion']
        localidad = datos['localidad']
        email = datos['email']
        clave = datos['clave']
        estado = datos['estado']
        tarjeta = datos['tarjeta']
        fechaalta = datetime.now()
        categoria = datos['categoria']
        datosusr =
        datos_usuarios_dj(nombres_usuario=nombre,apellidos_usuario=apellido,dni_usuario=dni,telefono_usuario=telefono,direccion_usuario=direccion,localidad_usuario=localidad,email_usuario=email,clave_usuario=clave,estado_usuario=estado,tarjeta_usuario=tarjeta,fecha_alta_usuario=fechaalta,categoria_usuario_id=categoria)
        datosusr.save()
        usuarios = datos_usuarios_dj.objects.all()
        return render_to_response('tablaUsuarios.html',{'usuarios':usuarios},RequestContext(request))
    return render_to_response('agregarUsuario.html',{'identPersonals':identPersonals},RequestContext(request))
```

En esta función vemos como se guardan los datos en sus respectivas variables y luego se almacenan en la tabla “*datos_usuarios_dj*” de nuestra base de datos. Luego la función re direcciona hacia la plantilla “*tablaUsuarios.html*” en donde vamos a poder observar al usuario ya presente en la tabla de usuarios habilitados.

Solo resta mapear la función *addUser* con su respectiva URL en el archivo *urls.py* de la siguiente manera:

```
from django.conf.urls import patterns, include, url

from django.contrib.auth.views import login, logout

from django.contrib import admin

urlpatterns = patterns('',
    url(r'^agregarUsuario', 'control.views.addUser', name='addUser'),
)

)
```

Para poder completar todos los datos al agregar un usuario es necesario asignarle a este una tarjeta o llavero RFID (es decir la clave de dicha tarjeta o llavero). Esta sección también permite realizar esta captura de clave. Para ello agregamos las siguientes líneas en la plantilla “*agregarUsuario.html*”:

```
<a href="/capturar">
    <input type="submit" value="Capturar Clave" class="btn btn-default" title="capturar">
</a>
```

Con esto generamos un botón llamado “*Capturar clave*” el cual nos va a dirigir a la url */capturar*. Luego le agregaremos funcionalidad a dicho botón mediante una función en el archivo *views.py* de la siguiente manera:

```
def capturar(request):
    identPersonals = Personal_dj.objects.all()
    formularios=FormularioUsuarios()
    code =""
    flag =0
    while (code ==""and flag <10):
        try:
            codes = Captura_clave.objects.get(id=1)
```

```

        code = codes.clave_captura
    except:
        flag +=1
        time.sleep(1)
    if code == "":
        code ="No_se_encontro"
    return render(request, 'agregarUsuario.html', {'formularios':
formularios,'identPersonals':identPersonals, 'clave': code})

```

Como ya vimos anteriormente, cada vez que pasamos la tarjeta por el lector de radiofrecuencia se almacena la clave en una tabla especial en la base de datos por un tiempo y luego se borra automáticamente. Aprovechando esto podemos utilizar esta función de python que nos permite comenzar una cuenta 10 segundos en los cuales debemos acercar la tarjeta “nueva” (es decir la que va a ser asignada al usuario a agregar) al lector. Si en esos 10 segundos la tarjeta es detectada, se guardara el código existente en la tabla “*Captura_clave*” de la base de datos en una variable y luego será copiada en el casillero correspondiente en la página de *agregarUsuario*.

Solo resta mapear la función *capturar()* con la URL en el archivo *urls.py* de la siguiente manera:

```

from django.conf.urls import patterns, include, url

from django.contrib.auth.views import login, logout

from django.contrib import admin

urlpatterns = patterns('',
    url(r'^capturar', 'control.views.capturar'),
)

```

A continuación vemos como queda la apariencia de esta página para agregar usuarios:

Para capturar la clave presione el botón "Capturar Clave" y luego pase el llavero por el lector. Si se capturó correctamente, el código deberá aparecer en la casilla de texto CLAVE
NOTA: Usted dispondrá de 10 segundos entre presionar el botón "Capturar" y pasar la tarjeta por el lector.

Ilustración 23: Sección agregar usuarios de página web

4.3.3.1.7.5 Sección Eventos

En primera instancia acudiremos a esta sección para poder visualizar los eventos generados al ingresar al Laboratorio, es decir, es una sección de control. En este apartado se verá la tabla “*Eventos_dj*” completa, mostrando todas las columnas a excepción del *id*.

Lo primero que haremos será ver una parte del código de la plantilla “tablaEventos.html” correspondiente a la creación de la tabla principal:

```
<table class="table table-hover table-bordered">
<thead>
<tr>
<th><center>Nombre</center></th>
<th><center>Apellido</center></th>
<th><center>Fecha y Hora</center></th>
<th><center>Lugar</center></th>
</tr>
</thead>
<tbody>
{%- for usuario in usuarios %}
<tr>
```

```

<tdalign='center'><h5>{{ usuario.nombres_eventos }}</h5></td>
<tdalign='center'><h5>{{ usuario.apellidos_eventos }}</h5></td>
<tdalign='center'><h5>{{ usuario.fechayhora_eventos }}</h5></td>
<tdalign='center'><h5>{{ usuario.lugar_eventos }}</h5></td>
</td>
{% endfor %}
</tbody>
</table>
<form action="/tablaEventos" method="post">{{ csrf_token }}
    <center><button type="submit" class="standar">Ver más</button></center>
</form>

```

Como podemos observar lo primero que aparece es el encabezado de la tabla, en el cual simplemente se escriben los títulos de las columnas de la misma. Luego para crear el contenido del cuerpo de la tabla utilizamos un FOR que muestra los datos que son capturados por una función en el archivo “views.py” y guardados en la variable “usuario”. Y por último existe un botón llamado “Ver más” que a través de un método POST nos permite ver la totalidad de los eventos existentes en la base de datos. A continuación vemos como se realiza esto dirigiéndonos al archivo “views.py” y agregando la siguiente función:

```

@login_required()
def mostrarTablaEventos(request):
    if request.method == "POST":
        eventos = Eventos_dj.objects.order_by('-id')
        return render_to_response('tablaEventos.html', {'usuarios': eventos}, RequestContext(request))
        eventos = Eventos_dj.objects.order_by('-id')[:30]
        return render_to_response('tablaEventos.html', {'usuarios': eventos}, RequestContext(request))

```

Esta función permite diferenciar si se quiere ver la tabla completa o la versión reducida. En el caso de no haber clicado el botón “Ver más” se mostrara la tabla de eventos reducida, en la cual se guardarán los datos de la tabla en una variable (las ultimas 30 filas) y se pasara esa variable para que pueda ser vista por la plantilla “tablaEventos.html” como vimos en el código al principio de la descripción de la sección. En el caso de haber clicado en el botón “Ver más” nos mostrara la tabla completa sin restricciones en cuanto al número de filas.

Resta mapear la función con la URL en el archivo “urls.py” de la siguiente manera:

```
from django.conf.urls import patterns, include, url
```

```

from django.contrib.auth.views import login, logout

from django.contrib import admin

urlpatterns = patterns('',
    url(r'^tablaEventos', 'control.views.mostrarTablaEventos', name='tablaEventos')
)

```

Otra de las funcionalidades que presenta esta sección es la de exportar la tabla a una hoja de datos de Excel. Existen dos posibilidades para exportar los datos, las cuales veremos a continuación en la siguiente sección de código de la plantilla “*tablaEventos.html*”:

```

<td>
    <formaction="/export/eventos"method="post">{% csrf_token %}
        <selectid="cant" name="cantidad" style="background-color:#F2F2F2;color:#1C1C1C;">
            <optionname=uno value=100 selected> Últimos 100 datos </option>
            <optionname=dos value=200> Últimos 200 datos </option>
            <optionname=tres value=500> Últimos 500 datos </option>
            <optionname=cuatro value=1000> Últimos 1000 datos </option>
            <optionname=cinco value=2000> Todos </option>
        </select>
    </td>
    <td>
        <buttonstyle="background-color:#F2F2F2;color:#1C1C1C;" id="exportar" name="exportar" backgroundtype="submit" class="standar">E
xportar</button>
    </form>
    </td>
    <td>
        <divclass="container">
            <divclass="hero-unit">
                <formaction="/export/eventosFecha"method="post">{% csrf_token %}
                    <labelid = "cant3"><i>Seleccione un día:</i></label><inputid="cant2" style="background-
color:#D8D8D8;color:#BDBDBD;" type="text" placeholder="DD-MM-YYYY" name="cantidad" disabled>
                    <buttonstyle="background-color:#D8D8D8;color:#BDBDBD;" id="exportar2" type="submit" name="exportar2" class="standar" disabled>E
xportar</button>
                </form>
            </div>
        </div>
    </td>

```

Lo que hicimos fue crear una tabla con 2 columnas, separando ambas maneras de exportar los datos. En la primera columna creamos a través de un FORM un casillero con opciones para poder elegir exportar una cantidad determinada de filas de la tabla de eventos (existe la opción de exportar 100, 200, 500, 1000 o todas las filas de la tabla). La segunda columna se hizo para crear un casillero del cual se revele un calendario para poder elegir una fecha en particular. Una vez elegida la fecha podremos exportar todos los eventos ocurridos en la misma.

Ambos métodos son POST y sus funcionalidades se describen en el archivo “views.py” mediante la siguiente función:

```
@csrf_exempt
def exportarAExcel (request,name):
    if name <>"usuarios":
        datos = request.POST
        count = datos[ 'cantidad']
        if name =='eventos':
            if count ==2000:
                values_list = Eventos_dj.objects.order_by( '-id').values_list()
            else:
                values_list = Eventos_dj.objects.order_by( '-id')[ :int(count)].values_list()
        if name =='eventosFecha':
            values_list = Eventos_dj.objects.filter(fechayhora_eventos__contains=
count).values_list()
```

En primer lugar diferenciamos la tabla de usuarios a la de eventos. En el caso de ser un exportar Eventos por cantidad primero guardamos en una variable la cantidad que se seleccionó mediante el botón selector de la plantilla. Si el valor es “2000” quiere decir que se seleccionó descargar TODOS los datos de la tabla y se guardan en una variable. Si no es así se guardan en la variable solo la cantidad de datos que se requirió. En el caso de querer exportar los eventos de una fecha específica se buscarán en la base de datos solo aquellos datos que contengan los caracteres exactos de la fecha solicitada y los guarda en una variable.

Luego de quedar guardados los datos se realiza la conversión de los mismos para pasarse a una hoja de datos de Excel como ya se realizó previamente en la sección de Usuarios. Para ver el código completo de la función dirigirse al anexo de “CÓDIGOS” de este informe.

Solo resta mapear la función con la URL (que necesita incluir parámetros) en el archivo “urls.py”:

```

from django.conf.urls import patterns, include, url

from django.contrib.auth.views import login, logout

from django.contrib import admin

urlpatterns = patterns('',
    url(r'^export/(?P<name>[-\w]+)$', 'control.views.exportarAExcel', name='exportarAExcel'),
)

```

Podemos observar como quedara la apariencia de la sección Eventos a través de la siguiente imagen:

Nombre	Apellido	Fecha y Hora	Lugar	
Mariano	Dominguez	14 Enero 2015 08:52:29	LAC	Ver Eventos Ver Foto
Mariano	Dominguez	14 Enero 2015 08:52:16	LAC	Ver Eventos Ver Foto
Agustin	Martina	9 Enero 2015 13:21:43	LAC	Ver Eventos Ver Foto
Leandro	Juan	19 Dic. 2014 15:54:00	LAC	Ver Eventos Ver Foto
Leandro	Juan	19 Dic. 2014 15:53:56	LAC	Ver Eventos Ver Foto

[Ver mas](#)

[Ver intentos de acceso fallidos](#)

Exportar por Cantidad Exportar por día

[Últimos 100 datos](#) [Exportar](#) Seleccionne un dia: [Exportar](#)

Ilustración 24: Sección ver eventos de página web

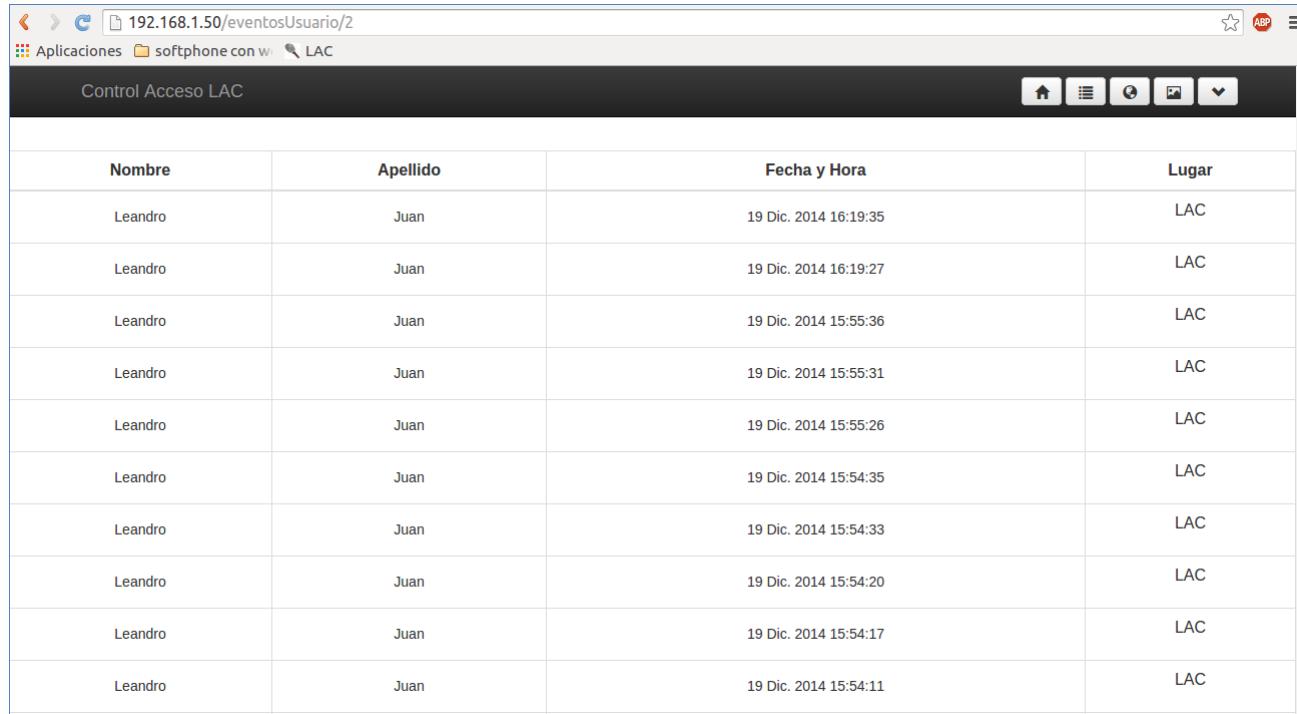
También podemos visualizar los eventos de un usuario específico. Para ello tenemos que agregar unas líneas de código a la plantilla “tablaEventos.html” con las cuales generamos una columna para la tabla de la siguiente manera:

```
{% for usuario in usuarios %}

<tr>
<tdalign='center'><h5>{{ usuario.nombres_eventos }}</h5></td>
<tdalign='center'><h5>{{ usuario.apellidos_eventos }}</h5></td>
<tdalign='center'><h5>{{ usuario.fechayhora_eventos }}</h5></td>
<tdalign='center'><h5>{{ usuario.lugar_eventos }}</h5></td>
<td><a href="/eventosUsuario/{{usuario.id_usuario_eventos}}">Ver Eventos</a><br/>
</td>
{% endfor %}
```

Como vemos solo agregamos una línea respecto a la versión de la plantilla vista al comienzo de esta sección. Esto nos va a permitir que al clicar en el botón “**Ver Eventos**” nos re direccione a otra página para ver los eventos específicos del usuario que está involucrado en la fila correspondiente (por esa razón pasamos como parámetro el id del usuario). La apariencia de esta nueva tabla es igual a la de la tabla Eventos, solo cambiarán los datos ya que solo mostraremos los eventos de un solo usuario. Para ver la apariencia de esta nueva tabla podemos ver la plantilla “eventosUsuario.html” y el archivo “views.py” en el anexo “CÓDIGOS” de este informe.

Podemos observar como quedara la apariencia de la sección Eventos de Usuario a través de la siguiente imagen:



The screenshot shows a web browser window with the URL 192.168.1.50/eventosUsuario/2. The page title is "Control Acceso LAC". The table has four columns: Nombre, Apellido, Fecha y Hora, and Lugar. All entries show "Leandro" in the Nombre column and "Juan" in the Apellido column. The Fecha y Hora column lists various dates and times from December 19, 2014, at 15:54:11 to 16:19:35. The Lugar column shows "LAC" for all rows.

Nombre	Apellido	Fecha y Hora	Lugar
Leandro	Juan	19 Dic. 2014 16:19:35	LAC
Leandro	Juan	19 Dic. 2014 16:19:27	LAC
Leandro	Juan	19 Dic. 2014 15:55:36	LAC
Leandro	Juan	19 Dic. 2014 15:55:31	LAC
Leandro	Juan	19 Dic. 2014 15:55:26	LAC
Leandro	Juan	19 Dic. 2014 15:54:35	LAC
Leandro	Juan	19 Dic. 2014 15:54:33	LAC
Leandro	Juan	19 Dic. 2014 15:54:20	LAC
Leandro	Juan	19 Dic. 2014 15:54:17	LAC
Leandro	Juan	19 Dic. 2014 15:54:11	LAC

Ilustración 25: Sección eventos de usuarios de página web

La ultima funcionalidad que presenta esta sección es la de poder ver los eventos de aquellos intentos de acceso al laboratorio que fueron denegados, es decir, que la tarjeta o llavero acercado al lector no estaba habilitado en la base de datos. Para ello fue necesario agregar el siguiente botón en la plantilla “tablaEventos.html”:

```
<formaction="/tablaNoPermitidoEventos">{% csrf_token %}

    <buttontype="submit" class="standar">Ver intentos de acceso fallidos</button>

</form>
```

Esto nos va a dirigir a una nueva página en la cual podremos observar la nueva tabla de accesos denegados. Para ello contamos con la siguiente plantilla denominada “tablaNoPermitidoEventos.html”:

```
<tableclass="table table-hover table-bordered">
<thead>
<tr>
<th><center>Clave</center></th>
<th><center>Fecha y Hora</center></th>
<th><center>Lugar</center></th>
</tr>
</thead>
<tbody>
{% for evento in eventos %}
<tr>
<tdalign='center'><h5>{{ evento.clave_eventos_no_permitido }}</h5></td>
<tdalign='center'><h5>{{ evento.fechayhora_eventos_no_permitido }}</h5></td>
<tdalign='center'><h5>{{ evento.lugar_eventos_no_permitido }}</h5></td>
<td><a href="/showphoto">Ver Foto</a><br/></td>
{% endfor %}
</tbody>
</table>
```

Esta plantilla es bastante simple y como podemos observar se crea una tabla en la que en la cabecera se ponen los títulos de las columnas (Clave, Fecha y Hora, Lugar) y en el cuerpo de la tabla se muestran los datos que son capturados en la función que veremos a continuación. Para ello tenemos que dirigirnos al archivo “views.py” y agregar la siguiente función:

```
@login_required()

defmostrarTablaNoPermitidoEventos(request):

    eventos = Eventos_no_Permitidos_dj.objects.all().order_by(' -id')

    return render_to_response('tablaNoPermitidoEventos.html',{'eventos': eventos
},RequestContext(request))
```

La función “mostrarTablaNoPermitidoEventos” también es muy simple, solo debemos guardar en una variable aquellos datos presentes en la tabla “Eventos_no_Permitidos_dj” de nuestra base de datos pero de manera invertida y luego pasarlo como parámetros a la plantilla “tablaNoPermitidoEventos.html”.

Por ultimo nos dirigimos al archivo “urls.py” para mapear la función reciente con la URL correspondiente de la siguiente manera:

```
from django.conf.urls import patterns, include, url

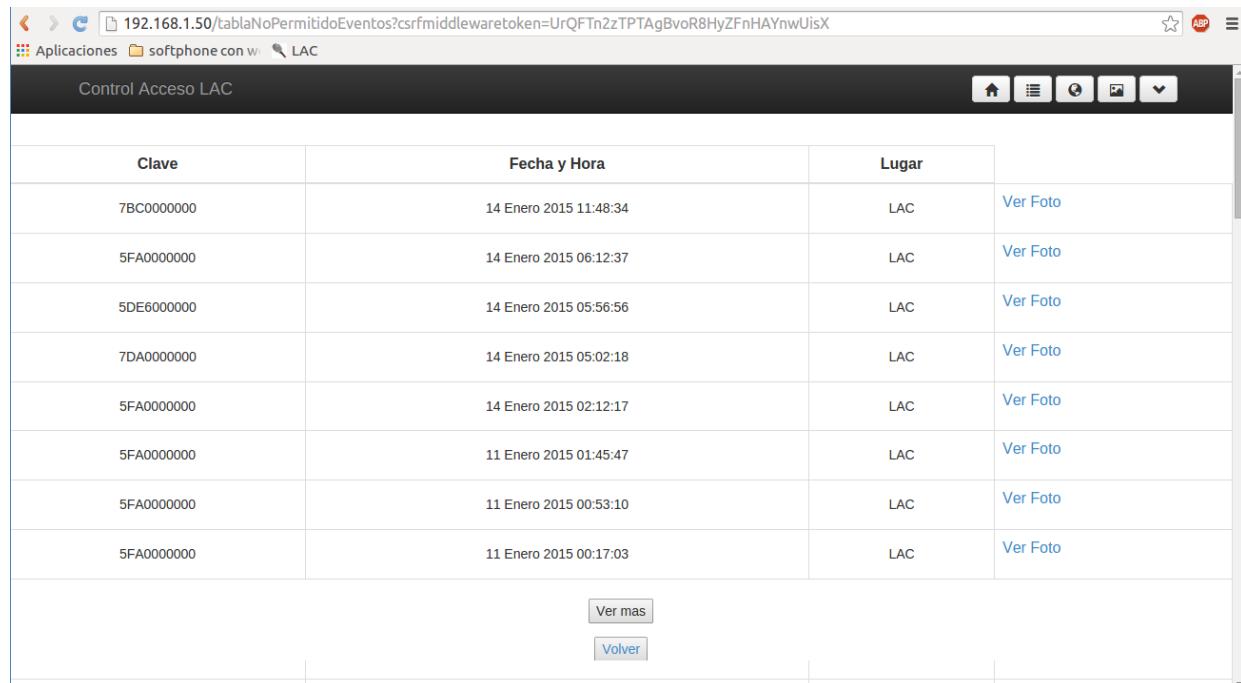
from django.contrib.auth.views import login, logout

from django.contrib import admin

urlpatterns = patterns('',
    url(r'^tablaNoPermitidoEventos', 'control.views.mostrarTablaNoPermitidoEventos',
        name='tablaNoPermitidoEventos'),
)

)
```

Y así se completan todas las funcionalidades presentes en la sección Eventos de nuestro Software de control web. La apariencia de esta página de Eventos de acceso no permitidos quedara de la siguiente manera:



The screenshot shows a web browser window with the URL `192.168.1.50/tablaNoPermitidoEventos?csrfmiddlewaretoken=UrQFTn2zTPTAgBvoR8Hy/ZFnHAYnwUisX`. The page title is "Control Acceso LAC". The main content is a table with the following data:

Clave	Fecha y Hora	Lugar	Ver Foto
7BC0000000	14 Enero 2015 11:48:34	LAC	Ver Foto
5FA0000000	14 Enero 2015 06:12:37	LAC	Ver Foto
5DE6000000	14 Enero 2015 05:56:56	LAC	Ver Foto
7DA0000000	14 Enero 2015 05:02:18	LAC	Ver Foto
5FA0000000	14 Enero 2015 02:12:17	LAC	Ver Foto
5FA0000000	11 Enero 2015 01:45:47	LAC	Ver Foto
5FA0000000	11 Enero 2015 00:53:10	LAC	Ver Foto
5FA0000000	11 Enero 2015 00:17:03	LAC	Ver Foto

At the bottom of the table, there are two buttons: "Ver mas" and "Volver".

Ilustración 26: Sección eventos de accesos no permitidos de página web

4.3.3.1.7.6 Sección EventosWeb

Esta sección, en cuanto a su implementación, es prácticamente idéntica a la sección de Eventos. Aquí se podrán ver aquellos eventos que fueron originados cuando se abrió la puerta del laboratorio mediante el software de control web, quedando registrado en el evento el usuario que estaba logueado en el sistema al momento de enviar la señal para la apertura de la puerta.

Para implementar la sección contamos con la siguiente sección de código de la plantilla “*tablaWebEventos.html*”:

```
<table class="table table-hover table-bordered">
<thead>
<tr>
<th><center>Usuario</center></th>
<th><center>Via</center></th>
<th><center>Fecha y Hora</center></th>
<th><center>Lugar</center></th>
</tr>
</thead>
<tbody>
{% for usuario in usuariosWeb %}
<tr>
<td align='center'><h5>{{ usuario.nombres_eventos_web }}</h5></td>
<td align='center'><h5>{{ usuario.via_eventos_web }}</h5></td>
<td align='center'><h5>{{ usuario.fechayhora_eventos_web }}</h5></td>
<td align='center'><h5>{{ usuario.lugar_eventos_web }}</h5></td>
<td><a href="/eventosUsuarioWeb/{{usuario.id_usuario_eventos_web}}">Ver Eventos Web</a><br/></td>
{% endfor %}
</tbody>
</table>
<form action="/tablaWebEventos" method="post">{% csrf_token %}
<center><button type="submit" class="standar">Ver más</button></center></form>
```

Como podemos observar lo primero que aparece es el encabezado de la tabla, en el cual simplemente se escriben los títulos de las columnas de la misma. Luego para crear el contenido del cuerpo de la tabla utilizamos un FOR que muestra los datos que son capturados por una función en el archivo “**views.py**” y guardados en la variable “*usuario*”. Y por último existe un botón llamado “*Ver más*” que a través de un método POST nos permite ver la totalidad de los eventos existentes en la base de datos. A continuación vemos como se realiza esto dirigiéndonos al archivo “**views.py**” y agregando la siguiente función:

```
@login_required()
```

```

defWebEventos(request):
    if request.method=="POST":
        eventosweb = Web_eventos_dj.objects.order_by('-id')
        return render_to_response('tablaWebEventos.html',{'usuariosWeb': eventosweb},RequestContext(request))

    eventosweb = Web_eventos_dj.objects.order_by('-id')[:30]
    return render_to_response('tablaWebEventos.html',{'usuariosWeb': eventosweb},RequestContext(request))

```

Esta función permite diferenciar si se quiere ver la tabla completa o la versión reducida. En el caso de no haber clicado el botón “Ver más” se mostrara la tabla de eventos web reducida, en la cual se guardaran los datos de la tabla en una variable (las ultimas 30 filas) y se pasara esa variable para que pueda ser vista por la plantilla “tablaEventos.html” como vimos en el código al principio de la descripción de la sección. En el caso de haber clicado en el botón “Ver más” nos mostrara la tabla completa sin restricciones en cuanto al número de filas.

Resta mapear la función con la URL en el archivo “**urls.py**” de la siguiente manera:

```

from django.conf.urls import patterns, include, url

from django.contrib.auth.views import login, logout

from django.contrib import admin

urlpatterns = patterns('',
    url(r'^tablaWebEventos$', 'control.views.WebEventos', name='WebEventos'),
)

```

La apariencia de esta sección será la siguiente:

The screenshot shows a web browser window with the URL 192.168.1.50/tablaWebEventos. The title bar says "Control Acceso LAC". The main content is a table with the following data:

Usuario	Vía	Fecha y Hora	Lugar	
root	WEB	14 Enero 2015 11:44:40	LAC	Ver Eventos Web
root	WEB	14 Enero 2015 11:33:01	LAC	Ver Eventos Web
mariano	WEB	14 Enero 2015 09:17:56	LAC	Ver Eventos Web
root	WEB	19 Dic. 2014 17:04:44	LAC	Ver Eventos Web
root	WEB	19 Dic. 2014 17:02:03	LAC	Ver Eventos Web
root	WEB	19 Dic. 2014 16:44:47	LAC	Ver Eventos Web
root	WEB	19 Dic. 2014 15:56:26	LAC	Ver Eventos Web
mariano	WEB	17 Dic. 2014 09:04:45	LAC	Ver Eventos Web
mariano	WEB	16 Dic. 2014 18:10:15	LAC	Ver Eventos Web

[Ver mas](#)

Ilustración 27: Sección eventos web de página web

Como en la sección de Eventos aquí también podemos ver los eventos de un usuario web específico al hacer clic en el botón “**Ver eventos**” de la última columna de la tabla. Esto nos re direccionara a la plantilla “eventosUsuarioWeb.html” que podemos ver en el anexo “CODIGOS” de este informe. La apariencia de esta nueva página será la siguiente:

The screenshot shows a web browser window with the URL 192.168.1.50/eventosUsuarioWeb/1. The title bar says "Control Acceso LAC". The main content is a table with the following data:

Usuario	Vía	Fecha y Hora	Lugar
root	WEB	14 Enero 2015 11:44:40	LAC
root	WEB	14 Enero 2015 11:33:01	LAC
root	WEB	19 Dic. 2014 17:04:44	LAC
root	WEB	19 Dic. 2014 17:02:03	LAC
root	WEB	19 Dic. 2014 16:44:47	LAC

[Ver mas](#)

Ilustración 28: Sección eventos de usuarios web de página web

De la misma forma que exportamos las tablas de Eventos a una hoja de datos de Excel en la sección anterior, aquí también podremos realizar lo mismo. La implementación es idéntica en ambos casos, pudiendo exportar por cantidad y por fecha específica. Se utiliza la misma función que se encuentra en el archivo “**views.py**” con la excepción que en este caso se pasan los parámetros pertenecientes a esta sección de eventos web como podemos ver a continuación:

```
@csrf_exempt
def exportarAExcel (request,name):
    if name <>"usuarios":
        datos = request.POST
        count = datos[ 'cantidad']
        if name =='eventosweb':
            if count ==2000:
                values_list =Web_eventos_dj.objects.order_by( '-id').values_list()
            else:
                values_list =Web_eventos_dj.objects.order_by( '-id')[ :int(count)].values_list()
            if name =='eventosFechaWeb':
                values_list =Web_eventos_dj.objects.filter(fechayhora_eventos_web__contains=count).values_list()
```

4.3.3.1.7.7 Sección Mostrar Imágenes

Esta sección del software es la que permite visualizar las imágenes que fueron capturadas por la cámara web del control de acceso y seguridad al detectar un intento de ingreso al laboratorio.

Como veremos a continuación las imágenes podrán ser vistas de 2 maneras diferentes:

- Por fecha específica
- En su totalidad, ordenadas por fecha

Para poder ver la implementación de esta funcionalidad es necesario recurrir a la plantilla “showphoto.html”, en la cual veremos la siguiente sección de código para poder entender la parte visual de la misma:

```
<formaction="/showphoto"method="post">{% csrf_token %}
<divclass="container">
    <divclass="hero-unit">
        <i>Seleccione un día: </i><inputtype="text"placeholder="DD-MM-YYYY"name="from"id="example1">
```

```

<button type="submit" class="standar">Cargar</button><i>{{dateFrom}}</i>
</div>
</div>
{% for photo in photos%}
<div class="col-sm-1 col-md-2">
    <a href="/static/photo/{{photo}}" class="thumbnail">
        
        {{photo}}
    </a>
    <center><a href="/file_download/{{photo}}">
        <button type="button" class="btn btn-default" title="Descargar foto {{photo}}">
            <span class="glyphicon glyphicon-download"></span>
        </button></a>
    <br>
</div>
{%endfor%}

```

Como podemos observar en primer lugar se crea el botón que va a permitir cargar las imágenes a través de un método POST el cual nos dirigirá a la url /showphoto. Luego podemos ver un ciclo FOR que permite recuperar las fotos que están presentes en la carpeta "/static/photo" e ir mostrándolas una a una para que aparezcan en la página correspondiente. Por ultimo agregamos la posibilidad de descargar las fotos mediante un botón en la parte inferior de cada imagen.

Para ver como se implementaron dichas funcionalidades debemos ir al archivo "views.py" y agregar las siguientes funciones:

```

@csrf_protect
def showphoto(request):
    if request.method == 'POST':
        dateFrom = request.POST['from']
        pathTotal = "/usr/src/web/login/static/photo/" + dateFrom + "*.jpg"
        photoFiles = glob.glob(pathTotal)
        if dateFrom == '':
            dateFrom = "Todas las fotos"
        else:
            dateFrom = time.strftime("%d-%m-%Y")
            pathTotal = "/usr/src/web/login/static/photo/" + dateFrom + "*.jpg"
            photoFiles = glob.glob(pathTotal)
    photo = []
    for photoFile in photoFiles:
        photoPath = photoFile.split("/")
        photo.append(photoPath[7])
    photo.sort()
    photo.reverse()

```

```

dateFrom = dateFrom +" (Se encontraron "+str(len(photo)) +" fotos)"
return render_to_response('showphoto.html',{'photos': photo, 'dateFrom': dateFrom
},RequestContext(request))

```

En esta función detectamos en primer lugar si existió una solicitud de método POST, de ser correcto es porque se hizo clic en el botón “Cargar”, con lo cual pasa por medio de un parámetro la fecha que fue completada en el casillero correspondiente y luego se busca por fecha en la carpeta /static/photo (debido a que la fecha es parte del nombre de cada imagen). En el caso de que no se haya completado el casillero correspondiente, se cargan todas las fotos que existan en la carpeta. El “ELSE” existe debido a que si nos dirigimos a la sección de visualización de imagen, en un comienzo debería mostrar las imágenes del día actual. Por lo cual guardamos en la variable “dateFrom” la fecha (día-mes-año) actual, y por consiguiente se van a mostrar solo las imágenes que se hayan capturado en ese día.

Luego resta que las imágenes se muestren en orden, para lo cual guardamos el path de las imágenes, las guardamos en una lista (append), ordenamos dicha lista (sort) y por ultimo invertimos esa lista para que las imágenes se muestren desde la más reciente hasta la menos reciente (en orden descendente).

En último lugar pasamos las imágenes dentro de una variable “photo” para que puedan ser manejadas por la plantilla como vimos en la primer parte de esta sección.

La segunda función presente en esta sección de visualización de imágenes es la siguiente:

```

@login_required()
def file_download(request,filename):
    filepath = os.path.join(settings.MEDIA_ROOT, filename)
    wrapper = FileWrapper(open(filepath, "rb"))
    content_type = mimetypes.guess_type(filepath)[0]
    response = HttpResponseRedirect(wrapper, content_type="application/octet-stream")
    response[ 'Content-Disposition' ] ="attachment; filename=%s"% filename
    return response

```

Esta función sirve para poder descargar cada imagen de manera particular a cualquier computadora. Para ello en primer lugar agregamos el pathname y el nombre del archivo a os.path, luego utilizamos un file wrapper para descargar en tamaños de 8kb hasta que el archivo sea descargado por completo, y por ultimo obtenemos el archivo mimetype. Con esta pequeña función ya podemos descargar exitosamente las imágenes a la computadora.

Solo resta mapear ambas funciones con las respectivas URLs, para ello nos dirigimos al archivo “urls.py” y agregamos las siguientes líneas:

```

from django.conf.urls import patterns, include, url

from django.contrib.auth.views import login, logout

from django.contrib import admin

urlpatterns = patterns('',
    url(r'^showphoto', 'control.views.showphoto', name='showphoto'),

    url(r'^file_download/(?P<filename>[^/]+)/$', 'control.views.file_download',
    name='file_download'),
)

```

La apariencia de esta sección será la siguiente:

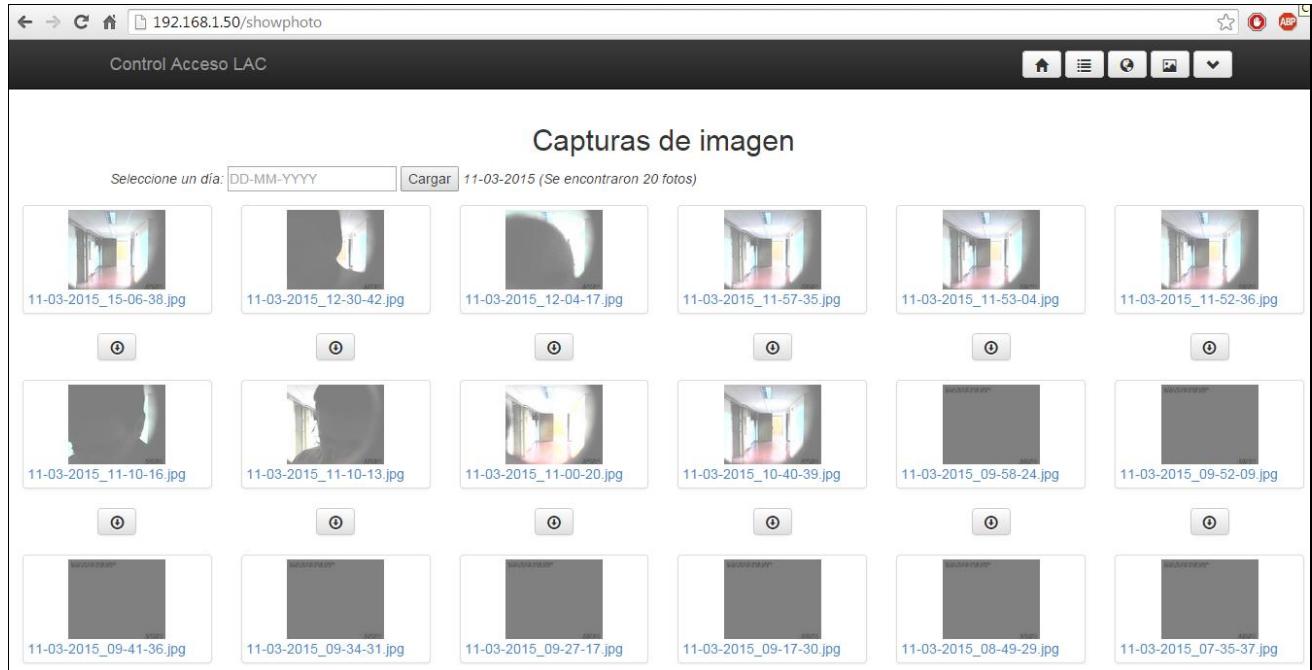


Ilustración 29: Sección capturas de imagen de página web

4.3.3.1.7.8 Sección Administrar Franjas Horarias

Esta sección presenta varias funcionalidades, pero en una visión “macro” se puede decir que su principal función será la de agregar, editar o eliminar franjas horarias a ciertos grupos de usuarios. Es decir, aquí se podrán definir los horarios en los que los usuarios del laboratorio van a poder ingresar, y en qué horarios van a tener el permiso de acceso denegado.

También se podrán agregar o eliminar categorías (denominación de los grupos de usuarios) para que se pueda realizar la administración de las franjas horarias en su totalidad dentro de esta sección.

En primer lugar veremos cómo se implementa la visualización de las franjas horarias en nuestra pantalla principal. Para ello veremos la siguiente sección de código dentro de la plantilla “tablaFranjas.html”:

```
<div class="panel panel-default">
    <div class="panel-heading">
        <i class="panel-title">Seleccione una Categoría</i>
        <span align="right"></span>
    </div>
    <div class="panel-body">
        <table>
            <tr>
                <td>
                    <form action="/tablaFranjas" method="post">{{ csrf_token }}<br/>
                        <select id="cmbPersonal" name="idPersonal" style="background-color:#F2F2F2;color:#1C1C1C;">
                            {% for identPersonal in identPersonals %}
                                {% if identPersonal.id == namePersonal.id %}
                                    <option value="{{identPersonal.id}}>{{identPersonal.identificacion_personal}}</option>
                                {% else %}
                                    <option value="{{identPersonal.id}}>{{identPersonal.identificacion_personal}}</option>
                                {% endif%}
                            {% endfor %}
                        </select>
                        <button style="background-color:#F2F2F2;color:#1C1C1C;" id="ver" name="ver" background-type="submit" class="standar">Ver</button>
                    </form>
                </td>
            </tr>
            {% for franja in Franjas %}<br/>
                <tr>
                    <td align='center'><h5>{{ franja.dia_franjas.nombre_dias }}</h5></td>
                    <td align='center'><h5>{{ franja.desde_franjas }}</h5></td>
                    <td align='center'><h5>{{ franja.hasta_franjas }}</h5></td>
```

```

<td><a href="/editarFranja/{{franja.id}}">Editar</a><br/><a href="/borrarFranja/{{franja.id}}"
"onClick="return confirm('Se va a borrar. ¿Esta Seguro?')>Eliminar</a></td>
</tr>
{% endfor %}

```

En esta sección de código podemos ver como se crea la tabla de visualización de las franjas horarias ya presentes en la base de datos del sistema. Como en todas las tablas creadas en las secciones anteriores, existirá un FORM con un método POST para poder comunicarse con la función en el archivo “views.py”. En primer lugar re direccionamos a la url /tablaFranjas y creamos un casillero “combo box” que nos va a permitir seleccionar entre una lista de categorías para poder visualizar sus franjas horarias específicas. A lo último agregamos un botón “Ver” para poder visualizar, una vez seleccionada la categoría correspondiente, las franjas horarias que están establecidas para dichas categorías.

Para capturar los datos que completan la lista se realiza mediante un ciclo FOR que muestra las variables que son pasadas mediante una función en el archivo “views.py”, por lo cual nos dirigimos a dicho archivo para ver la siguiente función:

```

@login_required()
def mostrarFranjasHorarias(request):
    identPersonals = Personal_dj.objects.all()
    if request.method=="POST":
        idPersonal = request.POST['idPersonal']
        namePersonal = Personal_dj.objects.get(pk=idPersonal)
        Franjas = namePersonal.franjas_horarias_dj_set.all().order_by('dia_franjas')
        return render_to_response('tablaFranjas.html',{'Franjas': Franjas,
,'identPersonals':identPersonals,'namePersonal':namePersonal},
context_instance=RequestContext(request))
    try:
        namePersonal = Personal_dj.objects.first()
        Franjas = namePersonal.franjas_horarias_dj_set.all().order_by('dia_franjas')
        return render_to_response('tablaFranjas.html',{'Franjas': Franjas,
,'identPersonals':identPersonals,'namePersonal':namePersonal},
context_instance=RequestContext(request))
    except:
        return render_to_response('agregarPersonal.html',
context_instance=RequestContext(request))

```

En la primera parte de esta función vemos si se realizó realizar una solicitud POST, lo cual quiere decir que el botón “Ver” fue seleccionado y debemos mostrar las franjas horarias de la categoría elegida. Para ello se guarda el ID de la categoría en una variable (namePersonal) y luego las franjas horarias que pertenecen a esa categoría en otra variable (Franjas). Por último se envían como parámetros para que la plantilla pueda utilizar esos datos y mostrarlos.

La segunda parte de la función es para cuando se ingresa por primera vez a la sección. Lo que se hace en este caso es guardar en una variable (namePersonal) la primera categoría que aparezca en la base de datos y luego guardar en otra variable las franjas horarias referidas a esa categoría. Esto se realiza para poder visualizar las franjas horarias de entrada cuando se ingresa en la sección.

De no existir ninguna categoría el TRY va a fallar, por lo cual se genera una excepción en la cual te permite ir a agregar una categoría nueva automáticamente para después poder agregar las franjas que se quieran a dicha categoría.

Por ultimo mapeamos la función con su respectiva URL en el archivo “urls.py” de la siguiente manera:

```
from django.conf.urls import patterns, include, url

from django.contrib.auth.views import login, logout

from django.contrib import admin

urlpatterns = patterns('',
    url(r'^tablaFranjas', 'control.views.mostrarFranjasHorarias', name='franjasHorarias'),
)
```

La apariencia de esta parte de la sección será la siguiente:

The screenshot shows a web browser window with the address bar displaying "192.168.1.50/tablaFranjas". The title bar says "Control Acceso LAC". The main content area has a header "Seleccione una Categoría" with buttons "Estudiante" and "Ver". Below this is a button "Agregar Categoría". A table titled "Categoría: Estudiante" lists days of the week with their corresponding start and end times. Each row has "Editar" and "Eliminar" links in the last column. At the bottom is a "Agregar Franja" button.

DIA	DESDE	HASTA	
Lunes	00:01:00	23:59:00	Editar Eliminar
Martes	00:01:00	23:59:00	Editar Eliminar
Miercoles	00:01:00	23:59:00	Editar Eliminar
Jueves	00:01:00	23:59:00	Editar Eliminar
Domingo	00:01:00	23:59:00	Editar Eliminar

Ilustración 30: Sección franjas horarias de página web

A continuación veremos como agregar una nueva franja horaria para una categoría en especial. Para ello vemos el siguiente código dentro de la plantilla “tablaFranjas.html”:

```
<center><input type="button" value="Agregar Franja" onclick="mostrar()"></center>
<div style="display:none; margin-left:5px; id="franja">
    <h4> Agregar franja en categoría: {{namePersonal.identificacion_personal}}</h4>
    <form action="/agregarFranja" name="datos" method="post" onsubmit="return validacion(this)">{%
        csrf_token %} <br>
        <p> ID: <input type="text" size="1" name="idPersonal" value="{{namePersonal.id}}></p>
        <p> Día:<br>
            <select id="diaFranja" name="diaFranja" style="background-color:#F2F2F2;color:#1C1C1C;">
                <option value=1>Lunes</option>
                <option value=2>Martes</option>
                <option value=3>Miércoles</option>
                <option value=4>Jueves</option>
                <option value=5>Viernes</option>
                <option value=6>Sábado</option>
                <option value=7>Domingo</option>
            </select></p>
        <p> Hora Desde:<br>
            <input type="text" placeholder="HH:MM" name="desdeFranja"></p>
        <p> Hora Hasta: <input type="text" placeholder="HH:MM" name="hastaFranja"></p>
        <input style="margin-left:75px" type="submit" value="Agregar">
    </form>
</div>
```

En esta sección de código vemos como al hacer clic en el botón “Agregar Franja” llamamos a un javascript “mostrar()” que nos muestra una sección que estaba oculta en la página. Dicha sección será un FORM con un método POST que nos va a permitir comunicarnos con la función específica dentro del archivo “views.py”. Dicho FORM nos redirigirá a la url /agregarFranja, y consta de, en primer lugar, agregar un casillero select que nos da la opción de elegir el día en el cual queremos agregar los horarios en los que los usuarios de la categoría actual estarán habilitados. Una vez seleccionado el día resta completar la hora de inicio y fin de la franja horaria y hacer clic en el botón “Agregar” para completar el proceso. Para que los datos sean completados correctamente existe un javascript “validación()” para comprobar que los datos ingresados en los casilleros tienen el formato correcto.

Para ver el código javascript hay que dirigirse al anexo “CODIGOS” de este informe.

Para ver el comportamiento de esta parte de la sección debemos ir al archivo “views.py” en donde creamos las siguientes funciones:

```

@login_required()
@csrf_protect
def agregarFranja(request):
    if request.user.is_superuser:
        if request.method=="POST":
            diaFranja = request.POST['diaFranja']
            idPersonal = request.POST['idPersonal']
            desdeFranja = request.POST['desdeFranja']
            hastaFranja = request.POST['hastaFranja']
            datoFranja = Franjas_horarias_dj(id_personal_franjas_id=idPersonal,
,dia_franjas_id=diaFranja,desde_franjas=desdeFranja, hasta_franjas=hastaFranja)
            datoFranja.save()
            identPersonals = Personal_dj.objects.all()
            namePersonal = Personal_dj.objects.get(pk=idPersonal)
            Franjas = namePersonal.franjas_horarias_dj_set.all().order_by('dia_franjas')
            return render_to_response('tablaFranjas.html',{'Franjas': Franjas,
,'identPersonals':identPersonals,'namePersonal':namePersonal},
context_instance=RequestContext(request))
        else:
            return HttpResponseRedirect("/error403")

```

Como vemos en las primeras líneas de esta función, este apartado solo está disponible si la persona que está utilizando el software de control es el administrador superusuario. De no ser así aparecerá una pantalla de error.

Esta función es simple y lo que hace es guardar en variables los datos que fueron completados en la plantilla “agregarFranja.html” y luego los graba en la base de datos. Por ultimo guarda el ID de la categoría actual en una variable (namePersonal) y sus respectivas franjas horarias en otra (Franjas) para luego re direccionar a la pantalla principal con los datos de las franjas horarias de la categoría a la cual se le decidió agregar horarios.

Solo resta mapear la función con la respectiva URL en el archivo “urls.py” de la siguiente manera:

```

from django.conf.urls import patterns, include, url

from django.contrib.auth.views import login, logout

from django.contrib import admin

urlpatterns = patterns('',
    url(r'^agregarFranja', 'control.views.agregarFranja', name='agregarFranja'),
)

```

La apariencia de esta parte de la sección será la siguiente:

The screenshot shows a web-based application titled "Control Acceso LAC". At the top, there's a navigation bar with icons for home, search, and other functions. Below it is a table with five rows, each representing a day of the week from Wednesday to Sunday. Each row contains three columns: the day name, a start time (00:01:00), and an end time (23:59:00). To the right of each row is a small menu with "Editar" and "Eliminar" options. Below the table is a button labeled "Agregar Franja". Underneath this button, there's a section for adding a new slot with fields for "ID" (set to 1), "Día" (set to "Lunes"), "Hora Desde" (set to "HH:MM"), and "Hora Hasta" (set to "HH:MM"). A final "Agregar" button is at the bottom of this section.

Ilustración 31: Sección agregar franjas horarias de página web

Las funcionalidades de editar Franjas y eliminar franjas se hacen de la misma manera que se vio en las secciones descriptas previamente donde existían tablas para visualizar. Para poder ver los códigos fuentes completos hay que dirigirse al anexo “CÓDIGOS” de este informe.

La apariencia de la funcionalidad de Editar Franjas será de la siguiente manera:

The screenshot shows a web-based application titled "Control Acceso LAC". At the top, there's a navigation bar with icons for home, search, and other functions. Below it is a section titled "Editar Franja Horaria". This section contains two input fields: "Hora desde" (set to "00:01:00") and "Hora Hasta" (set to "23:59:00"). Below these fields is a single "Editar" button.

Ilustración 32: Sección editar franjas horarias de página web

Como podemos ver solo hará falta completar 2 casilleros: El horario de inicio de la franja horaria y el horario de finalización.

La ultima funcionalidad que presenta esta sección es la de poder agregar o eliminar categorías de usuarios. Dichas categorías servirán para establecer franjas horarias que afecten a un grupo de usuarios (Estudiantes, Docentes, Personal de limpieza, etc). Para poder implementar esta funcionalidad debemos crear un botón en la plantilla “tablaFranjas.html” de la siguiente manera:

```
<td style="margin-right:10px">

<a href="{% url 'agregarPersonal' %}">

    <button type="button" class="btn btn-default" title="Agregar nuevo personal">

        <span class="glyphicon glyphicon-plus"></span>

    </button>

</a><span class="panel-title">Agregar Categoria</span></p>

</td>
```

El botón “Agregar Categoria” nos va a redirigir a una nueva URL /agregarPersonal. Para poder observar la apariencia de esta nueva parte de la sección debemos ver la plantilla “agregarPersonal.html” siguiente:

```
<center>

    <p><input type="button" value="Agregar" onclick="mostrar()"></p>

    <div style="display:none; margin-left:5px; id="agPersonal">

        <form action="/agregarPersonal" name="identificacion" method="post">{% csrf_token %}

            <p> IDENTIFICACION: <input type="text" size="25" name="identificacion"></p>

            <input style="margin-left:75px" type="submit" value="OK">

        </form>

    </div>

</center>
```

En la primera parte del código fuente completo se podrá observar cómo se crea la tabla para visualizar las categorías existentes en la base de datos, lo cual es similar a lo realizado con las secciones previas. Lo particular de esta funcionalidad es que una vez ingresado a la pantalla correspondiente se deberá hacer clic en el botón “Agregar Categoria” con lo cual aparecerá un casillero que se encontraba oculto gracias al javascript “mostrar()”, como podemos observar en la parte de código de esta nueva plantilla. El botón para agregar la categoría estará bajo el formato de un FORM y generara un método POST al hacer clic en él, lo cual permitirá comunicarse con la

función dentro del archivo “views.py”. Por lo tanto nos dirigimos a ese archivo y agregamos la siguiente función:

```
@login_required()
@csrf_protect
def agregarPersonal(request):
    if request.user.is_superuser:
        if request.method=="POST":
            identificacion = request.POST['identificacion']
            IdentificacionPers = Personal_dj(identificacion_personal= identificacion)
            IdentificacionPers.save()
            personal = Personal_dj.objects.all()
            return render_to_response('agregarPersonal.html',{'personal': personal},context_instance=RequestContext(request))
        else:
            return HttpResponseRedirect("/error403")
```

Esta funcionalidad solo está disponible para aquellos administradores que sean superusuarios, y de no ser así aparecerá una pantalla de error. Lo que hace esta función es guardar los datos de la tabla de la base de datos en una variable y luego pasarla como parámetro para que la plantilla pueda mostrarlos. Si existe un método POST fue porque el botón “Agregar Categoría” fue presionado, con lo cual guardamos el contenido del casillero que se completó en una variable (identificación) y luego guardamos dicho valor en la tabla “Personal_dj” de la base de datos del sistema de acceso. Con lo cual queda agregada la categoría de manera exitosa.

Por ultimo mapeamos la función con la URL específica dentro del archivo “urls.py” de la siguiente manera:

```
from django.conf.urls import patterns, include, url
from django.contrib.auth.views import login, logout
from django.contrib import admin
urlpatterns = patterns('',
    url(r'^agregarPersonal', 'control.views.agregarPersonal', name='agregarPersonal'),
)
```

La apariencia de esta nueva parte de la sección quedara de la siguiente manera:

The screenshot shows a web browser window with the URL 192.168.1.50/agregarPersonal. The title bar says 'Control Acceso LAC'. The main content area has a heading 'Agregar Categoría'. Below it is a table with two rows:

ID	IDENTIFICACION	
1	Estudiante	Eliminar
2	Docente	Eliminar

At the bottom center of the page is a small 'Agregar' button.

Ilustración 33: Sección agregar categorías de página web

4.3.3.1.7.9 Sección Ayuda

Esta es una sección que permitirá la consulta de cualquier inquietud que se requiera para la utilización del Software administrador web. Aquí se explicaran las distintas funcionalidades que presenta y como realizar todas las acciones disponibles paso a paso, así como también se encontrara información perteneciente a los autores del sistema de control de acceso y seguridad y bajo qué condiciones fue creado.

Los códigos tanto de la plantilla “Ayuda.html” como de la función “ayuda()” perteneciente a esta sección podrán ser visualizadas en el anexo “CÓDIGOS” de este informe. La implementación es muy simple ya que consta por sobre todas las cosas de texto.

La apariencia de esta sección quedara de la siguiente manera:

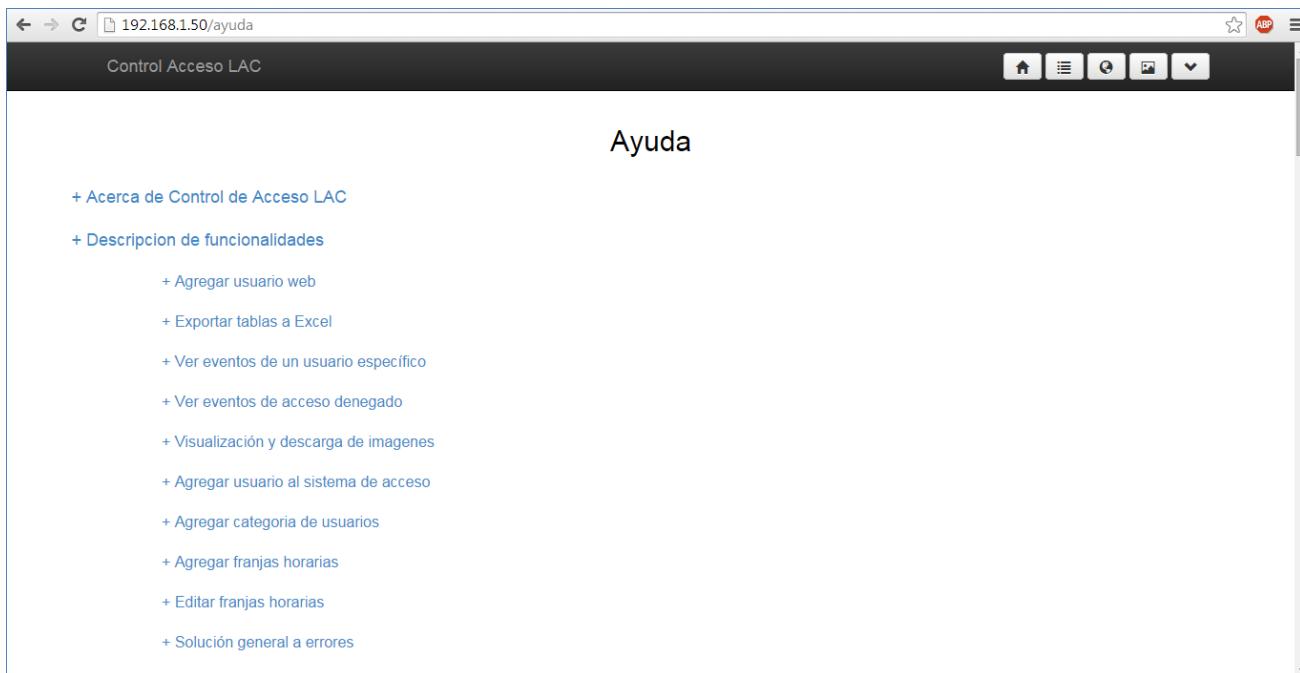


Ilustración 34: Sección ayuda de página web

4.3.4 Testing

4.3.4.1 Registro de usuarios y captura de clave mediante página web

En este testing pondremos a prueba el sistema de captura de clave y registro de usuarios mediante el uso de la página web. La captura de clave se realiza para poder asignar una clave personal a cada usuario (ID de tarjeta o llavero) y así poder registrar al mismo en la base de datos del sistema para luego realizar las consultas necesarias.

4.3.4.1.1 Cuadro de Análisis

Registro de usuarios y captura de clave mediante página web	
Requerimientos	R9 – R10
Propósito	Comprobar que se captura correctamente la clave de las tarjetas de seguridad al habilitar dicha función en la página web y que luego se registra de manera correcta al usuario en la base de datos del sistema
Entorno de Ejecución	Página web del sistema de control de acceso. Lector RFID.
Inicialización	Ejecutar el servidor “ Django ”. Iniciar la página web.
Finalización	Datos del nuevo usuario guardados en la base de datos del sistema.
Acciones	Ejecutar el servidor Django. Ingresar a la sección “ <i>Agregar usuario</i> ” de la página web. Presionar el botón “Capturar clave” y acercar la tarjeta de seguridad al lector RFID en menos de 10 segundos. Completar todos los campos del formulario. Presionar el botón “ <i>Grabar</i> ” para guardar los datos en la base de datos del sistema.
Resultados	
Resultados Esperados	La clave deberá ser capturada en el campo correspondiente del formulario. Los datos deberán ser guardados correctamente en la base de datos del sistema.
Resultados Obtenidos	Al presionar el botón “Capturar clave” y acercar una tarjeta de seguridad al lector RFID la clave se copia automáticamente en el campo correspondiente. Luego de completar todos los campos del formulario y presionar el botón “ <i>Grabar</i> ” se guardaron los datos de manera correcta en la base de datos.

Tabla 26: Cuadro de análisis del registro de usuarios y captura de clave mediante página web

4.3.4.1.2 Evidencia

Se utilizaron las siguientes tarjetas de proximidad RFID:

- Tarjeta 1: 310094FC3D
- Tarjeta 2: 28004332FA

Se agregaron 2 usuarios, a los cuales se les asigno las tarjetas con las claves mostradas anteriormente.

```
[08/May/2015 16:25:37] "GET /capturar HTTP/1.1" 200 6547
Capturando Clave...
Esperando Tarjeta...
Esperando Tarjeta...
Esperando Tarjeta...
Clave encontrada: 310094FC3D
[08/May/2015 16:29:26] "GET /capturar HTTP/1.1" 200 6543
Agregando Usuario Nuevo
Cargando datos de usuario nuevo
Abriendo Base de Datos
Grabando Datos de Usuario Nuevo
Datos Grabados Exitosamente
[08/May/2015 16:30:06] "POST /agregarUsuario HTTP/1.1" 200 28207
_____
[08/May/2015 16:31:29] "GET /agregarUsuario HTTP/1.1" 200 6533
Capturando Clave...
Esperando Tarjeta...
Esperando Tarjeta...
Clave encontrada: 28004332FA
[08/May/2015 16:31:52] "GET /capturar HTTP/1.1" 200 6543
Agregando Usuario Nuevo
Cargando datos de usuario nuevo
Abriendo Base de Datos
Grabando Datos de Usuario Nuevo
Datos Grabados Exitosamente
[08/May/2015 16:32:51] "POST /agregarUsuario HTTP/1.1" 200 28218
```

Ilustración 35: Evidencia registro de usuario y captura de clave mediante página web

4.3.5 Conclusión

Luego de haber desarrollado la tercera iteración hemos logrado administrar el sistema de control de acceso mediante una página web, con la cual cumplimos con todas las funcionalidades deseadas. La velocidad con la cual se cargan las páginas en la red interna es de 2 segundos, lo que permite un manejo del sistema mucho más fluido.

4.4 Iteración 4

4.4.1 Objetivos

En esta iteración se desarrolla el sistema de comunicación de voz vía VoIP, el cual permite al usuario habilitado comunicarse, mediante una llamada realizada desde la computadora, con el usuario que se encuentra en el exterior del laboratorio. Luego de finalizar la comunicación el administrador decide si habilita el ingreso o no mediante la apertura de la puerta de forma remota.

Esta implementación es realizada como antecedente a un trabajo a futuro que será la instalación de un botón en el exterior que inicie la llamada, avisando al usuario interno que éste desea ingresar.

4.4.2 Diseño

Aquí se describe como se realiza la comunicación VoIP entre el usuario interno y el usuario externo al establecimiento. También se puede visualizar como se le permite el ingreso al usuario externo vía web por parte del administrador.

4.4.2.1 Requerimientos

REQUERIMIENTOS INVOLUCRADOS		
Número	Prioridad	Requerimiento
7	Alta	El Software administrador deberá ser capaz de abrir la puerta
8	Alta	Deberá existir una comunicación de voz entre usuario externo e interno del laboratorio

Tabla 27: Requerimientos involucrados en el sistema de comunicación y acceso por web

4.4.2.2 Diagrama de secuencia

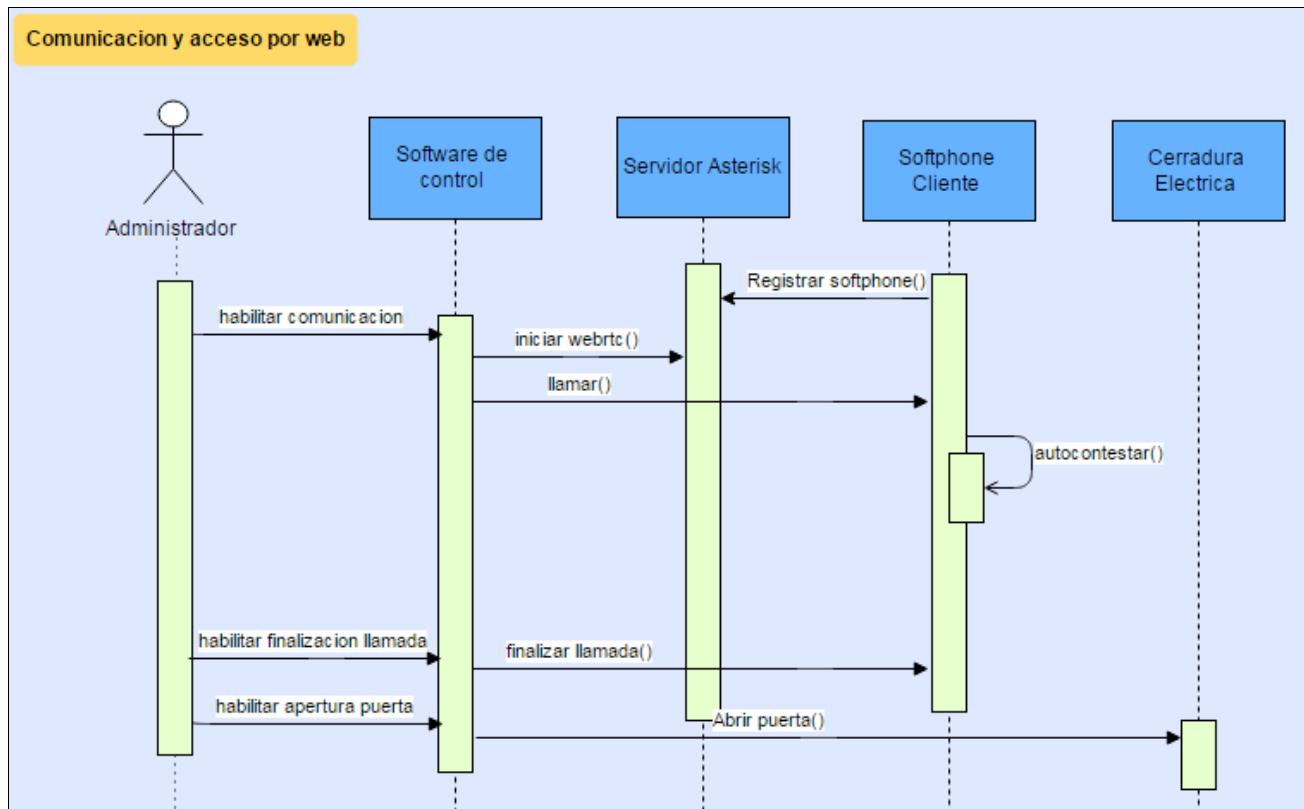


Ilustración 36: Diagrama de secuencia del sistema de comunicación y acceso por web

4.4.3 Implementación

4.4.3.1 Instalación y configuración de ASTERISK

Para lograr la comunicación vía VoIP entre el usuario externo y el usuario interno del laboratorio es necesario en primer instancia contar con un servidor de comunicaciones. En este caso vamos a mostrar como instalar y configurar ASTERISK, que fue el que utilizamos para el desarrollo del sistema.

4.4.3.1.1 Instalación Asterisk

En primer lugar vamos a descargar el paquete de asterisk desde su página oficial de la siguiente manera:

```
cd /usr/src
```

```
wget http://downloads.asterisk.org/pub/telephony/asterisk/asterisk-11-current.tar.gz
```

A continuación descomprimimos el archivo que acabamos de descargar e ingresamos a la carpeta que fue creada:

```
tar xvfz asterisk-11-current.tar.gz  
cd asterisk-*
```

En nuestro caso fue necesario instalar previamente una dependencia (la librería “ncurses”), por lo cual realizamos lo siguiente:

```
apt-get install libncurses5-dev
```

A continuación utilizamos el siguiente comando para realizar la configuración que permite la futura compilación:

./configure

Al terminar el proceso de configuración deberíamos poder observar algo similar a la siguiente imagen. De ser así es porque el proceso concluyó de manera exitosa

Ilustración 37: Configuración Asterisk terminada

Luego iniciamos el proceso de compilación de asterisk mediante la utilización de los siguientes comandos:

sudo make

Al concluir deberíamos observar lo siguiente:

```
+----- Asterisk Build Complete -----+
+ Asterisk has successfully been built, and +
+ can be installed by running:                 +
+                                               +
+           make install                      +
+-----+
+----- Asterisk Build Complete -----+
```

Ilustración 38: Compilación Asterisk terminada I

sudo make install

Al concluir deberíamos observar lo siguiente:

```
+---- Asterisk Installation Complete -----+
+                                         +
+     YOU MUST READ THE SECURITY DOCUMENT   +
+                                         +
+ Asterisk has successfully been installed. +
+ If you would like to install the sample  +
+ configuration files (overwriting any      +
+ existing config files), run:             +
+                                         +
+           make samples                     +
+                                         +
+-----+
+---- Asterisk Installation Complete -----+
```

Ilustración 39: Compilación Asterisk terminada II

sudo make samples

4.4.3.1.2 Instalación Asterisk-GUI

GUI es la Interfaz gráfica para Asterisk creada por Digium, es un plataforma para poder configurar Asterisk de forma gráfica y remota al mismo tiempo.

En primer lugar ingresamos a la carpeta y luego comenzamos con la configuración y compilación de la siguiente manera:

```
cd /asterisk-gui  
.configure  
make  
make install
```

4.4.3.1.3 Configuración Asterisk

Ya tenemos instalado la GUI para Asterisk. Ahora tenemos que configurar dos ficheros que se encuentran en **/etc/asterisk**.

Algunas ya existen y solo hay que quitarles el ";" para des comentarlas.

Primero editamos el fichero **manager.conf** para que quede así:

```
manager.conf  
  
[general]  
  
displaysystemname = yes  
  
enabled = yes  
  
webenabled = yes  
  
port = 5038  
  
bindaddr = 0.0.0.0  
  
[root] (Este será nuestro usuario para entrar a la GUI)  
  
secret = 1234 (Esta será nuestra contraseña para entrar a la GUI)  
  
read = system,call,log,verbose,command,agent,user,config  
  
write = system,call,log,verbose,command,agent,user,config
```

Luego editamos el fichero http.conf para que quede así:

http.conf

```
[general]  
  
enabled = yes  
  
enablestatic = yes  
  
bindaddr = 0.0.0.0  
  
bindport = 8088
```

Una vez compilado Asterisk-GUI, podemos comprobar si todo está correcto ejecutando:

```
#make checkconfig
```

Si todo ha ido bien , nos mostrara un mensaje de que ya podemos acceder:

```
root@raspberrypi:~/asterisk-gui# make checkconfig  
--- Checking Asterisk configuration to see if it will support the GUI ---  
* Checking for http.conf: OK  
* Checking for manager.conf: OK  
* Checking if HTTP is enabled: OK  
* Checking if HTTP static support is enabled: OK  
* Checking if manager is enabled: OK  
* Checking if manager over HTTP is enabled: OK  
--- Everything looks good ---  
* GUI should be available at http://raspberrypi:8088/asterisk/static/config/index.html  
  
* Note: If you have bindaddr=127.0.0.1 in /etc/asterisk/http.conf  
you will only be able to visit it from the local machine.  
  
Example: http://localhost:8088/asterisk/static/config/index.html  
  
* The login and password should be an entry from /etc/asterisk/manager.conf  
which has 'config' permission in read and write. For example:  
  
[admin]  
secret = mysecret4452  
read = system,call,log,verbose,command,agent,config  
write = system,call,log,verbose,command,agent,config  
  
--- Good luck! ---  
root@raspberrypi:~/asterisk-gui#
```

Ilustración 40: Asterisk Checkconfig

4.4.3.2 Instalación y configuración del Softphone

Para poder realizar la llamada desde la PC del usuario interno del laboratorio hacia el exterior del mismo es necesario contar con un softphone instalado en la Raspberry Pi y a su vez que exista un usuario registrado al cual va a ser dirigida la llamada. Para ello utilizamos el Softphone TWINKLE.

4.4.3.2.1 Instalación Twinkle

La instalación del Softphone Twinkle es muy fácil y rápida (esta fue una de las razones por las cuales se decidió utilizar este softphone en particular).

Solo basta con utilizar el siguiente comando en la consola de nuestra Raspberry Pi:

#apt-get install twinkle

Una vez finalizada la instalación ya podremos acceder a configurar el mismo y registrar el usuario.

4.4.3.2.2 Registrar usuario

Para registrar un usuario en el Softphone Twinkle debemos, en primer lugar, iniciar el software mediante la interfaz gráfica del sistema operativo Raspbian (puede hacerme mediante el VNC Server). Una vez iniciado el programa veremos una apariencia como la siguiente:

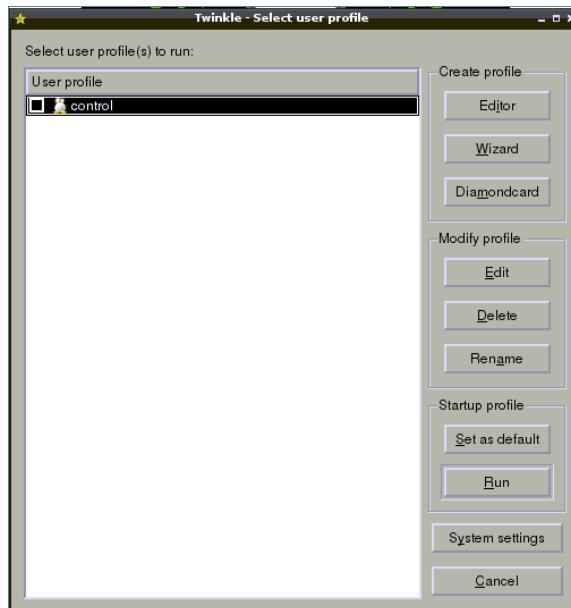


Ilustración 41: Registro de usuario I

A continuación tenemos que dirigirnos al apartado “*Create profile*” y clicamos en el botón “*Wizard*” el cual, en primer lugar, nos permitirá colocar un nombre al perfil:

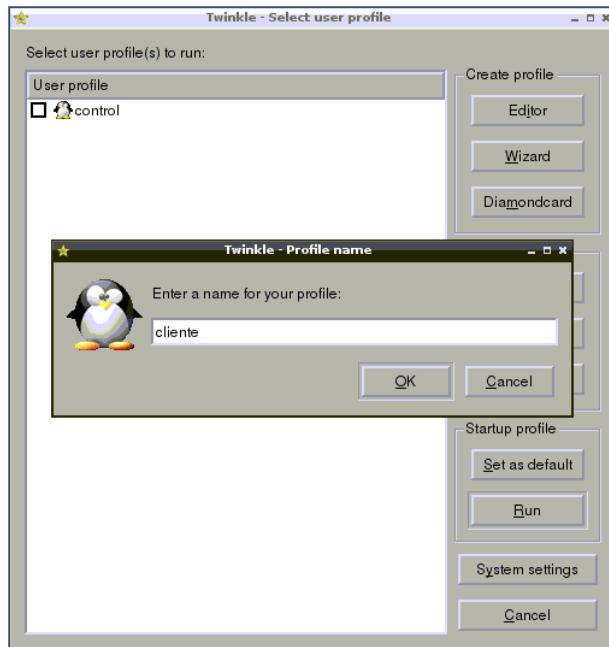


Ilustración 42: Registro de usuario II

Una vez que aceptamos nos aparecerá una pantalla en la cual debemos completar los datos del usuario (que tiene que estar creado en Asterisk previamente) como vemos a continuación:

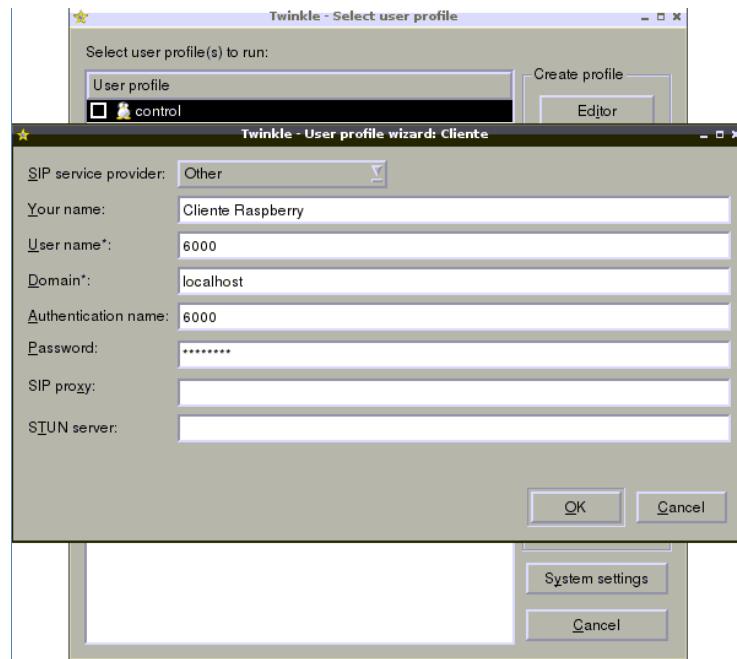


Ilustración 43: Registro de usuario III

Ya tenemos el usuario creado, solo resta confirmar el registro del mismo en el softphone. Para ello primero seleccionamos el perfil de usuario en la pantalla principal y nos dirigimos al apartado

“Startup profile” para clicar en el botón “Run”, con lo cual el usuario quedara registrado automáticamente.

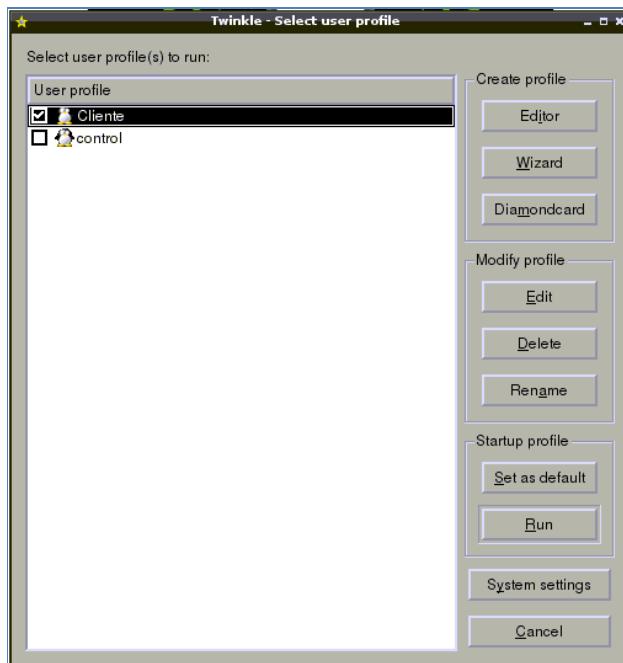


Ilustración 44: Registro de usuario IV

4.4.3.2.3 Configurar audio

Para que el sonido se escuche de manera clara y sin cortes ni saturación es necesario configurar el audio del Softphone. Para ello tenemos que iniciar el software mediante la interfaz gráfica del sistema operativo Raspbian como ya se realizó previamente para registrar el usuario. A continuación iremos a la pestaña “Edit” y luego a “System Settings”.

Dentro de esta ventana nos dirigimos a la sección “Audio” y cambiamos los valores por defecto por los siguientes valores:

ALSA play period size: 256

ALSA capture period size: 64

4.4.3.3 Instalación y configuración de JanusWebRTC

Para poder realizar la comunicación de voz (VoIP) desde el navegador web entre usuario interno y externo del laboratorio es necesario contar con una plataforma que sirva de pasarela y soporte este protocolo. Para ello contamos con el software JANUS.

4.4.3.3.1 Instalación de dependencias

Esta pasarela de comunicación web en tiempo real puede ser instalada únicamente en sistemas operativos Linux, lo cual no es un problema ya que en nuestra Raspberry contamos con el Sistema operativo Raspbian, que está basado en Debian.

La instalación puede llevar varios minutos ya que es necesario instalar varias dependencias y pueden surgir algunos problemas mientras se intenta realizar ese proceso de instalación.

Las dependencias necesarias que debemos instalar son las siguientes:

- libmicrohttpd: es una pequeña librería C que intenta hacer más fácil correr un servidor HTTP como parte de otra aplicación.
- libini-config (INI configurations): Es una librería para procesar archivos de configuración de formato “ini” en una estructura de datos “libcollection”.
- Jansson: Es una librería C para codificar, decodificar y manipular data JSON
- libnice: Sirve para aplicaciones que necesitan establecer flujos de datos UDP peer-to-peer. Automatiza el proceso de trasversar NATs y provee seguridad ante algunos ataques. También permite a aplicaciones crear flujos confiables utilizando capas TCP en vez de capas UDP.
- OpenSSL (at least v1.0.1e): Contiene las librerías de desarrollo, archivos de cabecera y documentación SSL (Secure Sockets Layer).
- libsrtsp: Es una implementación open source del protocolo de transporte de seguridad en tiempo real (Secure Real-time Transport Protocol – SRTP).
- Sofia-SIP: Es una librería SIP User-Agent de código abierto. Puede ser usada como un bloque de compilación para software de cliente SIP para usos como VoIP, IM, y muchos otros servicios de comunicaciones en tiempo real y persona a persona.

Adicionalmente debemos instalar las siguientes librerías y herramientas:

- GLib: GLib proporciona los bloques básicos para construir aplicaciones y bibliotecas escritas en C. Proporciona el sistema de objetos básico usado en GNOME, la implementación del bucle principal, y un gran conjunto de funciones de utilidad para cadenas y estructuras de datos comunes.
- pkg-config: es una herramienta usada cuando se compilan librerías y aplicaciones. Ayuda a insertar las opciones correctas del compilador en la línea de comandos para que una

aplicación pueda usar gcc -o test test.cpkg-config --libs --cflags glib-2.0 por ejemplo, en vez de valores de difícil codificación en donde encontrar glib u otras librerías. Puede ser utilizado para definir la localización de herramientas de documentación.

- gengetopt: Este programa genera una función en C que usa la función getopt_long para parsear las opciones de la línea de comandos, para validarlas y llenar una estructura. También puede generar una función para grabar las opciones de la línea de comandos en un archivo, y una función para leer las opciones de la línea de comandos desde un archivo.

Para instalar dichas dependencias es necesario utilizar los siguientes comandos en la consola de nuestro Raspbian:

```
aptitude install libmicrohttpd-dev libjansson-dev libnice-dev \
    libssl-dev libsrtpp-dev libsofia-sip-ua-dev libglib2.0-dev \
    libopus-dev libogg-dev libini-config-dev libcollection-dev \
    pkg-config gengetopt libtool automake
```

4.4.3.3.2 Compilación de Janus

Una vez instaladas las dependencias debemos, en primer lugar, descargar el código fuente de Janus. Para ello utilizamos el siguiente comando:

```
git clone https://github.com/meetecho/janus-gateway.git
cd janus-gateway
```

Una vez que entramos a la carpeta recién descargada debemos utilizar un comando para preparar la compilación automática:

```
sh autogen.sh
```

Una vez terminado este proceso obtendremos el archivo de configuración. Luego ya podremos realizar el proceso de configuración y compilación de la siguiente manera:

```
./configure --disable-websockets --disable-data-channels --disable-rabbitmq --
--disable-docs
make
```

```
make install
```

Como podemos observar en el comando de configuración utilizado recientemente, deshabilitamos websockets, data channels y rabbitmq ya que en el caso de nuestro sistema de acceso no los vamos a utilizar. También deshabilitamos la documentación ya que no fue instalada debido a que ocupa mucho espacio en nuestra tarjeta de memoria.

Para instalar automáticamente los archivos de configuración por defecto debemos utilizar lo siguiente:

```
make configs
```

Hay que tener en cuenta que Janus contiene una serie de plugins que no vamos a utilizar. En el caso del desarrollo del sistema de comunicación por VoIP solo utilizaremos el plugin “SIP Gateway”, por lo cual también podemos deshabilitar los demás plugins de la misma manera que se hizo anteriormente con la documentación, websockets, etc. Para ello podemos utilizar el comando –help y así obtener la información para poder deshabilitarlos.

4.4.3.3 Configuración e inicio de Janus

Existen varias cosas que pueden ser configuradas a partir del archivo de configuración siguiente:

```
<installdir>/etc/janus/janus.cfg
```

O en la línea de comandos:

```
<installdir>/bin/janus --help

janus 0.0.6

Usage: janus [OPTIONS]...

-h, --help                  Print help and exit
-V, --version                Print version and exit
-i, --interface=ipaddress    Interface to use (will be the public IP)
-p, --port=portnumber         Web server HTTP port (default=8088)
-s, --secure-port=portnumber  Web server HTTPS port (default=no HTTPS)
-n, --no-http                 Disable insecure HTTP web server (default=off)
-b, --base-path=basepath      Base path to bind to in the web server
```

	(default=/janus)
-w, --ws-port=portnumber	WebSockets server port (default=no WebSockets)
-W, --ws-secure-port=portnumber	Secure WebSockets server port (default=no secure WebSockets)
-N, --no-websockets	Disable insecure WebSockets server (default=off)
-m, --admin-port=portnumber	Admin/monitor web server HTTP port (default=7088)
-M, --admin-secure-port=portnumber	Admin/monitor web server HTTPS port (default=no HTTPS)
-O, --no-admin	Disable insecure HTTP admin/monitor web server (default=off)
-B, --admin-base-path=basepath	Base path to bind to in the HTTP/HTTPS admin/monitor web server (default=/admin)
-Q, --admin-secret=randomstring	Admin/monitor secret all requests need to pass in order to be accepted by Janus (useful a crude form of authentication, none by default)
-L, --admin-acl=list	Comma-separated list of IP addresses allowed to use the Admin/monitor; partial strings are supported (e.g., 192.168.0.1,10.0.0.1 or 192.168., default=no restriction)
-P, --plugins-folder=path	Plugins folder (default=./plugins)
-C, --config=filename	Configuration file to use
-F, --configs-folder=path	Configuration files folder (default=./conf)
-c, --cert-pem=filename	HTTPS/DTLS certificate
-k, --cert-key=filename	HTTPS/DTLS certificate key
-S, --stun-server=filename	STUN server(:port) to use, if needed (e.g., gateway behind NAT, default=none)
-X, --ice-ignore-list=list	Comma-separated list of interfaces or IP addresses to ignore for ICE gathering; partial strings are supported (e.g., vmnet8,192.168.0.1,10.0.0.1 or vmnet,192.168., default=vmnet)
-e, --public-ip=ipaddress	Public address of the machine, to use in SDP
-6, --ipv6-candidates	Whether to enable IPv6 candidates or not (experimental) (default=off)
-l, --libnice-debug	Whether to enable libnice debugging or not (default=off)
-q, --max-nack-queue=number	Maximum size of the NACK queue per user for retransmissions
-r, --rtp-port-range=min-max	Port range to use for RTP/RTCP (only available if the installed libnice supports it)
-d, --debug-level=1-7	Debug/logging level (0=disable debugging, 7=maximum debug level; default=4)
-a, --apisecret=randomstring	API secret all requests need to pass in order to be accepted by Janus (useful when wrapping Janus API requests in a server, none by default)
-R, --enable-rabbitmq	Enable RabbitMQ support (default=off)
-H, --rabbitmq-host=string	Address (host:port) of the RabbitMQ server to use (default=localhost:5672)

```
-t, --rabbitmq-in-queue=string
                                Name of the RabbitMQ queue for incoming
                                messages (no default)
-f, --rabbitmq-out-queue=string
                                Name of the RabbitMQ queue for outgoing
messages (no default)
```

Las opciones pasadas a través de la línea de comandos tienen precedencia de aquellos especificados en el archivo de configuración. Para iniciar el Gateway simplemente utilizamos el siguiente comando:

```
<installdir>/bin/janus
```

Esto inicializara el Gateway, y mirara el archivo de configuración. Por defecto, solo un webserver HTTP es iniciado. Para habilitar el soporte HTTPS hay que editar el archivo de configuración o usar la línea de comandos. El webserver hará uso de los mismos certificados provistos por DTLS. También se puede cambiar la base del path que el webserver usa: por defecto este es /janus, pero se puede cambiar a cualquiera que se quiera y con los subdirectorios que se quieran (por ejemplo /mypath, /my/path, o /my/really/nested/path). Esto se hace para permitir reglas más fáciles de customización en cualquier frontend que se pueda tener.

En ausencia de un archivo de configuración, la únicas opciones que deben especificarse en la línea de comandos son aquellas relacionadas con el certificado DTLS. Un certificado por defecto es provisto con el paquete en el directorio “certs”, el cual se puede utilizar iniciando el ejecutable con los siguientes parámetros:

```
<installdir>/bin/janus -c /path/to/mycert.pem -k /path/to/mycert.key
```

De esta última manera fue como se realizó para el desarrollo de nuestro sistema, ya que no utilizamos el archivo de configuración.

En este punto, el Gateway escuchara el puerto 8088 (o cualquier puerto que se haya elegido) de la Raspberry Pi. Para testear que esté funcionando correctamente se pueden usar los demos provistos en el directorio “html” del paquete descargado. Solo hace falta copiar el archivo contenido en un webserver, y abrir la página index.html en Chrome o Firefox.

4.4.3.4 Integración de la comunicación via VoIP con la página web

La última de las funcionalidades (y la más compleja) que presenta esta sección es la de iniciar y finalizar una comunicación vía VoIP a través de la Web. Para ello utilizamos el Gateway Janus el cual requiere el uso de javascripts y plugins. Los archivos *js* serán los encargados de otorgar las funcionalidades necesarias (no será necesario modificar ni agregar nada al archivo *views.py*).

A continuación vemos las líneas que fueron agregadas a la plantilla *home.html* para permitir la comunicación y también para finalizarla:

```
<script type="text/javascript" src="static/js/jquery.min.js"></script>
<script type="text/javascript" src="static/js/jquery.blockUI.js"></script>
<script type="text/javascript" src="static/js/spin.min.js"></script>
<script type="text/javascript" src="static/js/janus.js"></script>
<script type="text/javascript" src="static/js/siptest.js"></script>

<form action="/home" method="post">
<div class="container">
    <div class="row">
        <div class="col-md-12">
            <p><center><button name="callpage" class="btn btn-default" id="start">Llamar</button></center></p>
            <div class="container hide" id="sipcall">
                <div class="row">
                    <div class="col-md-12">
                        <div class="col-md-6 container hide" id="login">
                            <button class="btn btn-success margin-bottom-sm" id="register">Register</button>
                        </div>
                        <div class="col-md-6 container hide" id="phone">
                            <div class="input-group margin-bottom-sm hide">
                                <span class="input-group-addon"><i class="fa fa-phone fa-fw"></i></span>
                                <input class="form-control" type="text" placeholder="SIP URI to call (e.g., sip:1000@example.com)" id="peer" onkeypress="return checkEnter(this, event);">
                            </div>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>
</div>
```

```
</div>
</div>
</div>
</div>
</div>
</div>
</div>
</form>
```

Como se observa en la sección de la plantilla viste anteriormente, primero cargamos los javascripts necesarios para que la comunicación funcione correctamente. Luego creamos un formulario en el que en primera medida creamos un botón para iniciar la llamada “**Llamar**”. Este botón se comunica con los javascripts a través de su id “start” y allí es donde se inicia la comunicación.

A continuación se agrega código referente al login y registro del usuario SIP, los cuales se comunicaran con los javascripts también a través de sus id “login” y “register” respectivamente. Esto se va a generar automáticamente y estará oculto para la vista del usuario web.

Los archivos javascripts como así también el código fuente completo del *home.html* estarán disponibles en el anexo “CÓDIGOS” de este informe.

4.4.4 Testing

4.4.4.1 Visualización en tiempo real y comunicación de voz vía VoIP

En este testing pondremos a prueba el sistema de comunicación entre el usuario interno y el externo. El usuario externo es aquella persona que desea ingresar al establecimiento y no posee identificación. El usuario interno es aquel sujeto que está dentro del establecimiento y mediante la página web (**Framework Django**) se comunica por VoIP con el usuario externo. Además se hará la prueba de la visualización de la cámara web en tiempo real junto con la apertura de la puerta remotamente mediante la página web.

Para la comunicación de voz utilizamos una pasarela de comunicación web en tiempo real (WebRTC) denominada “**janus**”. Por ultimo debemos utilizar un servidor de telefonía IP llamado “**Asterisk**” para establecer la llamada VoIP desde la página web hacia un softphone (**Twinkle**) que se encuentra instalado en la placa Raspberry Pi.

4.4.4.1.1 Cuadro de Análisis

Visualización en tiempo real y comunicación de voz vía VoIP	
Requerimientos	R1 – R7 – R8
Propósito	Comprobar que se visualiza la cámara web en tiempo real y que se establece exitosamente la comunicación de voz entre usuario interno y externo del laboratorio
Entorno de Ejecución	La cámara web (micrófono) y parlantes están conectados a la placa Raspberry Pi. El framework Django nos permite interactuar con la cámara web y la comunicación.
Inicialización	Ejecutar el servidor “ janus ”. Ejecutar el servidor “ motion ”. Ejecutar el servidor “ asterisk ”. Ejecutar el servidor “ Django ”. Inicializar el softphone “ Twinkle ”. Conectar la cámara web al puerto USB de la placa Raspberry Pi. Conectar parlantes en la placa Raspberry Pi.
Finalización	Comunicación de voz establecida. Visualización de la cámara web en tiempo real.
Acciones	Ejecutar todos los servidores. Ingresar a la sección “ <i>home</i> ” de la página web, en donde se visualizara la cámara web. Presionar el botón “ <i>Llamar</i> ” para iniciar la comunicación de voz. Presionar el botón “ <i>Stop</i> ” para finalizar la comunicación de voz.
Resultados	
Resultados Esperados	Los servidores deberán ser ejecutados e inicializados correctamente para permitir la visualización de la cámara web en tiempo real y la comunicación de voz vía VoIP.
Resultados Obtenidos	Al ingresar a la sección “ <i>home</i> ” de la página web se inicializa la visualización de la cámara web correctamente. Luego se inicia una sesión WebRTC janus. A continuación se registra el softphone en el servidor ASTERISK para luego establecer la comunicación VoIP.

Tabla 28: Cuadro de análisis de visualización en tiempo real y comunicación de voz vía VoIP

4.4.4.1.2 Evidencia

```
##### Visualización de cámara web en tiempo real #####
```

Resource interpreted as Image but transferred with MIME type multipart/x-mixed-replace:
"http://192.168.1.50:8081/".

janus.js:38 Initializing library

```
##### Inicializacion de sesion en pasarela WebRTC JANUS #####
```

Library static/js/adapter.js loaded

janus.js:94 Library initialized: true

janus.js:118 Using REST API to contact Janus

janus.js:120 http://192.168.1.50:8090/janus

janus.js:390 Create session:

janus.js:391 Object {janus: "success", transaction: "avdHB4EY3coH", data: Object}

janus.js:400 Created session: 3381987266

janus.js:163 Long poll...

janus.js:595 Create handle:

janus.js:596 Object {janus: "success", session_id: 3381987266, transaction: "wLu5cFWLljd2", data: Object}

janus.js:603 Created handle: 3413622820

siptest.js:89 Plugin attached! (janus.plugin.sip, id=3413622820)

janus.js:673 Sending message to plugin (handle=3413622820):

janus.js:674 Object {janus: "message", body: Object, transaction: "iVMhewRESM8m"}

siptest.js:350 This is a SIP audio call (dovideo=false)

```
##### Registro del Softphone con ASTERISK #####
```

```
siptest.js:134 :: Got a message :::  
siptest.js:135 {"sip":"event","result":{"event":"registered","username":"6001"}}  
siptest.js:154 Successfully registered as 6001!  
  
##### Comunicacion VoIP #####  
siptest.js:134 :: Got a message :::  
siptest.js:135 {"sip":"event","result":{"event":"accepted","username":"sip:6000@192.168.1.50"}}  
siptest.js:225 sip:6000@192.168.1.50 accepted the call!  
janus.js:1224 Remote description accepted!  
janus.js:933 Handling Remote Stream:  
janus.js:934 MediaStreamEvent {stream: MediaStream, path: NodeList[0], cancelBubble: false, returnValue: true, srcElement: RTCPeerConnection...}  
janus.js:938 Starting bitrate monitor
```

4.4.5 Conclusión

Al implementar el sistema de comunicación se nos presentaron diversos problemas. El principal fue la comunicación de voz, la cual no fue simple debido a que el usuario interno del laboratorio debía poder transmitir voz y recibir audio mediante la utilización de un navegador web. Mientras que el usuario externo debía hacer uso de voz y audio sin ninguna interacción de un teclado o botón para activar este mecanismo.

Por esta razón decidimos que el usuario externo debía comunicarse mediante un softphone con auto-contestación, en donde la voz la captura el micrófono de la cámara web y el audio es reproducido por la placa Raspberry Pi.

Al decidir usar un softphone fue necesario implementar un servidor de VoIP, lo cual nos llevó al servidor Asterisk. Así mismo, éste debía tener una interconexión con el servidor Django para que el usuario interno pueda lograr la comunicación con el usuario externo. Dicha interconexión se realizó con la pasarela JANUS.

Una vez interconectados ambos usuarios se logró cumplir con una de las partes críticas de esta iteración. La calidad de audio que presenta el sistema no es óptima, pero se resolverá en trabajos futuros (Ver Capítulo 6).

La otra funcionalidad de esta iteración, abrir la puerta remotamente, se logró sencillamente a través de un script hecho en python dentro del servidor web Django.

4.5 Iteración 5

4.5.1 Objetivos

En esta iteración se desarrolla todo lo relacionado con la seguridad del sistema. Se realiza un script para borrar las capturas de imagen antiguas y así no sobrecargar la memoria del sistema y también se genera un script para levantar los archivos y servicios que tuvieron fallos y dejaron de funcionar. Por último se enviarán e-mails a una casilla de correo específica para avisar al administrador del sistema que hubo un fallo en el mismo.

4.5.2 Diseño

En este apartado se describe como, ante una falla en el sistema, se envía un aviso a una casilla de mail determinada para comunicar el fallo en tiempo real y así poder tomar las medidas necesarias para resolverlo.

4.5.2.1 Requerimientos

REQUERIMIENTOS INVOLUCRADOS		
Numero	Prioridad	Requerimiento
11	Media	Ante fallos deberán existir avisos vía email

Tabla 29: Requerimientos involucrados en el sistema de seguridad con aviso de fallos

4.5.2.2 Diagrama de secuencia

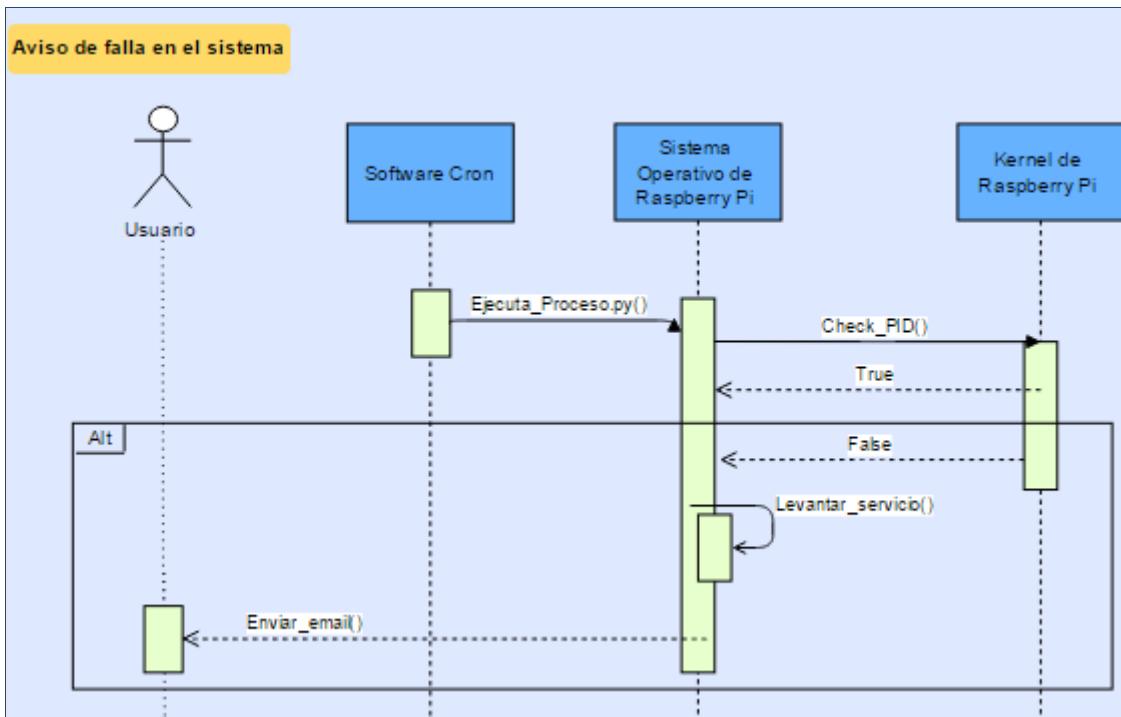


Ilustración 45: Diagrama de secuencia del sistema de seguridad con aviso de fallos

4.5.3 Implementación

4.5.3.1 Eliminación de capturas de imagen antiguas

Debido a que las imágenes son guardadas en una carpeta dentro de la memoria SD de la placa Raspberry Pi es necesario no sobrecargar la misma, ya que posee una capacidad limitada (8GB). Para ello se creó un sistema de buffer circular que permite el borrado de imágenes antiguas, dejando un límite de 2000 imágenes para que queden guardadas en la memoria.

El código del script se llama “cleanupbuffer.py” y es el siguiente:

```
#!/usr/bin/python

import os
import fnmatch
search_dir = "/usr/src/web/login/static/photo"
os.chdir(search_dir)
files = filter(os.path.isfile, os.listdir(search_dir))
```

```

files = [os.path.join(search_dir, f) for f in files] # add path to each file

files.sort(key=lambda x: os.path.getmtime(x))

files.reverse()

for i,items in enumerate(files):
    if i > 2000:
        os.remove(items)

```

En primer lugar se busca la carpeta en donde están almacenadas las capturas de imagen para luego agregar un path a cada archivo que esta presente en dicha carpeta. Luego ordenamos los archivos de acuerdo a su fecha de creación lo cual nos generará una lista de los mismos ubicándolos desde el mas antiguo hasta el mas nuevo.

Por último debemos ordenarlos en forma inversa ya que, para borrarlos, se debe recorrer cada archivo, y una vez que se supera la cantidad de archivos máxima empezará a borrar las capturas de imagen excedentes, las cuales deberán ser las mas antiguas.

4.5.3.2 Script para levantar servicios caídos

Este script cumple una función vital para la seguridad del sistema de control de acceso, así como también para la vigilancia y el control mediante el entorno web, ya que se encarga de levantar aquellos servicios que puedan haberse caído por algún error que se puede haber generado en el sistema.

El script se llama “proceso.py” y no permite que un servicio este caído por más de 1 minuto (gracias al software CRON podremos ejecutar este script en ese intervalo de tiempo), pudiendo arreglarse solo y automáticamente sin la necesidad de que intervenga un usuario administrador del sistema.

Cuenta con varias funciones las cuales pasaran a ser descriptas a continuación, pero en primer lugar vemos el código del Main que permite realizar las llamadas a dichas funciones:

```

if __name__ == '__main__':
    checkPy("control")
    checkPy("manage")
    checkOther("janus")
    checkOther("motion")
    checkOther("twinkle")

```

4.5.3.2.1 Levantar archivos python

Para levantar aquellos servicios que fueron escritos en lenguaje Python y tienen una extensión .py se utiliza la siguiente función:

```
def checkPy(archivo):
    separarEnter = []
    archivoBuscar = "ps aux | grep "+ archivo +".py"
    busqueda = commands.getoutput(archivoBuscar)
    separarEnter = busqueda.split('\n')
    if len(separarEnter) < 3:
        if archivo == "manage":
            archivo = "djangoservice"
            data = time.strftime("%c") + ' - +' + " El proceso " + archivo + " esta caido." + "\n"
            writeFile(data)
            levantarArchivo = "service "+ archivo +" start"
            sendEmail(archivo)
            time.sleep(3)
```

Esta función en primer lugar recibe el parámetro “archivo” que es el nombre del servicio que se quiere controlar. En las primeras líneas vemos como se genera una variable que contiene el comando necesario para mostrar si el servicio está siendo ejecutado en el sistema. Si ese comando devuelve menos de 3 líneas de texto quiere decir que el servicio está caído y hay que levantarlo.

Por consiguiente vemos si el script que está caído es el “manage” y de ser así cambiamos el nombre del archivo a “djangoservice” ya que así se llama el servicio. De no ser ese el nombre del archivo quiere decir que el que está caído era el “control” al cual no se le debe cambiar el nombre de archivo ya que el script y el servicio se denominan de la misma manera.

Por último generamos una variable que va a contener el comando necesario para levantar el servicio en el sistema y lo ejecutamos con una llamada a *os.system*.

También cómo podemos ver en el código existe un comando que permite enviar un mail avisando que un servicio estaba caído y se intentó levantar, lo cual será explicado luego en otra función y también con mayor profundidad en otra sección de este informe.

4.5.3.2.2 Levantar servicios

Esta función permite levantar los servicios que no se generan a través de scripts en lenguaje Python. El código de la misma es el siguiente:

```
def checkOther(archivo):
    if archivo == 'motion':
        levantarArchivo = "motion -n"
    elif archivo == 'janus':
        levantarArchivo = "service janus start"
    else:
        levantarArchivo = "twinkle -c"
    separarEnter = []
    archivoBuscar = "ps aux | grep " + archivo
    busqueda = commands.getoutput(archivoBuscar)
    separarEnter = busqueda.split('\n')
    if len(separarEnter) < 3:
        data = time.strftime("%c") + ' - +' + " El proceso " + archivo + " esta caido \n"
        writeFile(data)
        a = os.system(levantarArchivo)
        time.sleep(3)
```

En primer lugar esta función recibe como parámetro el nombre del servicio que se quiere controlar su estado, y luego se realizan consultas para ver cambiar el nombre del servicio a levantar dependiendo de cuál sea el nombre recibido. Ese nombre del servicio que será usado en el comando a ejecutar será guardado en una variable denominada “levantarArchivo”.

A continuación se hace el mismo procedimiento que con la función descripta anteriormente, viendo la cantidad de líneas de texto que son devueltas por parte del comando “ps aux | grep”. Si el resultado es menor a 3 quiere decir que el servicio esta caído y debe levantarse mediante el llamado al comando “os.system” junto con la variable “levantarArchivo” que fue generada previamente.

4.5.3.3 Aviso de servicio caído por email

Esta función lo único que realiza es la generación de variables necesarias para luego poder llamar a otra función más general que será la encargada de realizar el envío de un correo electrónico avisando de una falla existente.

El código es muy simple y es el siguiente:

```
defsendEmail(archivo):  
    mensaje ="Proceso "+ archivo +" dejo de funcionar. Verificar"  
    asunto ="Error en proceso "+ archivo  
    mailfrom =vigilancialac@gmail.com  
    responseEmail(mailfrom,mensaje, asunto).send()
```

Como vemos en primer lugar creamos el texto que va a ir dentro del correo electrónico que será enviado y lo guardamos en la variable “mensaje”.

Luego guardamos en la variable “asunto” el título del correo electrónico que será enviado, el cual contiene el nombre del servicio que tuvo la falla.

A su vez generamos una tercera variable que es constante y contiene la dirección de email a la cual se le debe enviar el mensaje. En este caso utilizamos una cuenta de email creada especialmente para la recepción de estos mensajes y que es administrada por el superusuario administrador del software de control web.

Por último debemos llamar a la función “responseEmail” enviando como parámetros aquellas variables que generamos recientemente. Esta nueva función puede ser vista en el anexo “CÓDIGOS” de este informe.

4.5.4 Testing

4.5.4.1 Levantar servicios caídos y aviso de falla vía Email

En este testing pondremos a prueba el sistema de seguridad para levantar servicios caídos y el aviso de fallas mediante el envío de email. Para realizar esto se utiliza el script “**proceso.py**” que se encarga de levantar los servicios que tuvieron algún fallo y dejaron de funcionar y el software “**Cron**” que es el encargado de activar periódicamente el script mencionado anteriormente para que los servicios caídos puedan levantarse.

4.5.4.1.1 Cuadro de Análisis

Levantar servicios caídos y aviso de falla vía Email	
Requerimientos	R11 – R21
Propósito	Comprobar que se levantan automáticamente los servicios que tuvieron algún tipo de fallo y dejaron de funcionar y que ante dicho fallo se envía exitosamente un Email de aviso de falla.
Entorno de Ejecución	Sistema operativo de la Raspberry Pi.
Inicialización	Script “ proceso.py ”. Ejecutar el software “ Cron ”.
Finalización	Servicios activos y mail enviado a la casilla de correo correspondiente.
Acciones	Colocar la dirección del script “ proceso.py ” dentro de la lista de ejecución del software “ Cron ”. Permitir que “ Cron ” se ejecute cada 1 minuto automáticamente.
Resultados	
Resultados Esperados	Los servicios que dejaron de funcionar inesperadamente deberán activarse de manera automática en un intervalo de 1 minuto. Si algún servicio no estaba funcionando al momento de activarse el script “proceso.py” deberá enviarse un email a una casilla de correo específica.
Resultados Obtenidos	Los servicios que no funcionaban fueron activados de manera correcta y los emails fueron enviados a la casilla de correo esperada.

Tabla 30: Cuadro de análisis de levantar servicios caídos y aviso de falla vía email

4.5.4.1.2 Evidencia

Se utilizaron los siguientes servicios y/o procesos para chequear:

- Control: Servicio de control de acceso.
- Manage: Servicio de pagina web Django.
- Janus: Servicio WebRTC para comunicación de voz (VoIP).
- Motion: Servicio de vigilancia mediante cámara web.
- Twinkle: Softphone en Raspberry receptor de llamada.

En primera instancia se chequeó el funcionamiento de todos los servicios y procesos estando activos. Luego se fueron desactivando los primeros 4 procesos en el orden previamente establecido para analizar la detección de falla y posterior envío de email.

El script “**proceso.py**” fue ejecutado mediante el software “**Cron**” cada 1 minuto.

```
*****  
Chequeando si el proceso control esta funcionando  
El proceso control se encuentra activo  
Chequeando si el proceso manage esta funcionando  
El proceso manage se encuentra activo  
Chequeando si el proceso janus esta funcionando  
El proceso janus se encuentra activo  
Chequeando si el proceso motion esta funcionando  
El proceso motion se encuentra activo  
Chequeando si el proceso twinkle esta funcionando  
El proceso twinkle se encuentra activo  
*****  
*****  
Chequeando si el proceso control esta funcionando  
El proceso control esta caido  
[ ok ] Starting system control daemon..  
Reiniciando el proceso control  
Proceso control funcionando nuevamente  
Envio de email exitoso por falla del proceso control  
Chequeando si el proceso manage esta funcionando  
El proceso manage se encuentra activo  
Chequeando si el proceso janus esta funcionando  
El proceso janus se encuentra activo  
Chequeando si el proceso motion esta funcionando  
El proceso motion se encuentra activo  
Chequeando si el proceso twinkle esta funcionando  
El proceso twinkle se encuentra activo  
*****
```

```
*****
Chequeando si el proceso control esta funcionando
El proceso control se encuentra activo
Chequeando si el proceso manage esta funcionando
El proceso djangoservice esta caido
Starting Django Service
Reiniciando el proceso djangoservice
Proceso djangoservice funcionando nuevamente
Envio de email exitoso por falla del proceso djangoservice
Chequeando si el proceso janus esta funcionando
El proceso janus se encuentra activo
Chequeando si el proceso motion esta funcionando
El proceso motion se encuentra activo
Chequeando si el proceso twinkle esta funcionando
El proceso twinkle se encuentra activo
*****
*****  
*****  
Chequeando si el proceso control esta funcionando
El proceso control se encuentra activo
Chequeando si el proceso manage esta funcionando
El proceso manage se encuentra activo
Chequeando si el proceso janus esta funcionando
El proceso janus esta caido
Starting Janus Service
Reiniciando el proceso janus
Proceso janus funcionando nuevamente
Envio de email exitoso por falla del proceso janus
Chequeando si el proceso motion esta funcionando
El proceso motion se encuentra activo
Chequeando si el proceso twinkle esta funcionando
El proceso twinkle se encuentra activo
*****
*****  
*****  
Chequeando si el proceso control esta funcionando
El proceso control se encuentra activo
Chequeando si el proceso manage esta funcionando
El proceso manage se encuentra activo
Chequeando si el proceso janus esta funcionando
El proceso janus se encuentra activo
Chequeando si el proceso motion esta funcionando
El proceso motion esta caido
Reiniciando el proceso motion
Proceso motion funcionando nuevamente
Envio de email exitoso por falla del proceso motion
Chequeando si el proceso twinkle esta funcionando
El proceso twinkle se encuentra activo
*****
```

Ilustración 46: Evidencia de levantar servicios caídos y aviso de fallas vía email

4.5.5 Conclusión

Luego de haber desarrollado la quinta iteración concluimos que los scripts en python nos permitieron de una manera muy simple y efectiva controlar los archivos y servicios del sistema y tomar acciones a partir del estado de los mismos. La integración de éstos con el software Cron nos ayudó a ejecutarlos en períodos de tiempo específicos, lo cual nos aseguró que las fallas no van a estar presentes durante largos intervalos de tiempo o informarnos que existe un problema que impide la ejecución correcta de algún servicio.

4.6 Hardware del sistema

4.6.1 Objetivos

En este apartado se describe el hardware necesario para la implementación de este proyecto integrando todas las iteraciones mencionadas anteriormente.

Para poder alimentar a todo el sistema es necesario contar con dos niveles de tensión diferentes. Necesitamos 12v para poder alimentar la electrocerradura, el lector RDIF, el Relé y la potencia de audio. A su vez, necesitamos de 5v para alimentar la Raspberry Pi y un integrado llamado MAX232 que se encarga de convertir los niveles de tensión del puerto serial entre el lector RFID y la entrada de datos Rx de las Raspberry. El sistema en su conjunto consume alrededor de 1.3A por lo cual hemos decidido usar una fuente de 12v 2A como mínimo.

Para separar 5v de los 12v utilizamos un integrado llamado LM7805 con el encapsulado T0-3, como lo describiremos a continuación.

4.6.1 Circuito reductor de 12v a 5v

Como mencionamos anteriormente, en nuestro sistema necesitamos 5v para alimentar distintos dispositivos, esos 5v los obtenemos de los 12v de la fuente de alimentación general. Para eso utilizamos como componente principal el integrado LM7805. Dicho integrado admite una entrada entre 7v a 48v y tiene como salida 5v constantes. Utilizamos el encapsulado TO-3 que soporta hasta 3A. El circuito es el siguiente:

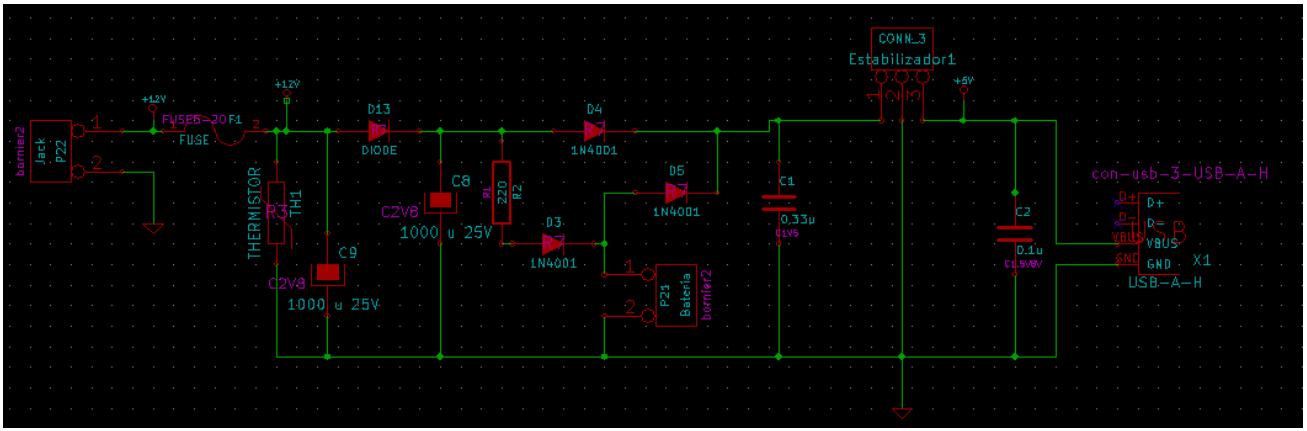


Ilustración 47: Esquemático del circuito reductor de 12V a 5V

En dicho circuito tenemos una bornera (P22) que es la entrada de 12v, un fusible (F1) para proteger el sistema, luego tenemos un termistor (TH1) que nos protege contra los picos de tensión. Después contamos con un capacitor (C9) utilizado para filtrar el ruido en la potencia de audio. Luego disponemos de un conjunto de diodos (D3, D3, D5 y D13), una resistencia (R2), un capacitor (C8) y una bornera (P21) que están implementados para colocar una batería de 12v como sistema de resguardo ante un corte de energía 220v. Y por último agregamos otra bornera de tres pines (Estabilizador1) junto a dos capacitores (C1 y C2) que terminan conectándose con una ficha USB. En dicha bornera se conecta el integrado LM7805 para regular la tensión, de esa forma al puerto USB le llegarán los 5v necesarios.

4.6.2 MAX 232

Debido a que los pines de la Raspberry Pi funcionan con 3.3V y el lector de tarjetas envía pulsos de 5V, fue necesario realizar un circuito para estabilizar la tensión y así asegurar la correcta comunicación de lo que envía el lector y lo que recibe la Raspberry. Para lograr esto se utilizó el integrado MAX232 con sus correspondientes componentes.

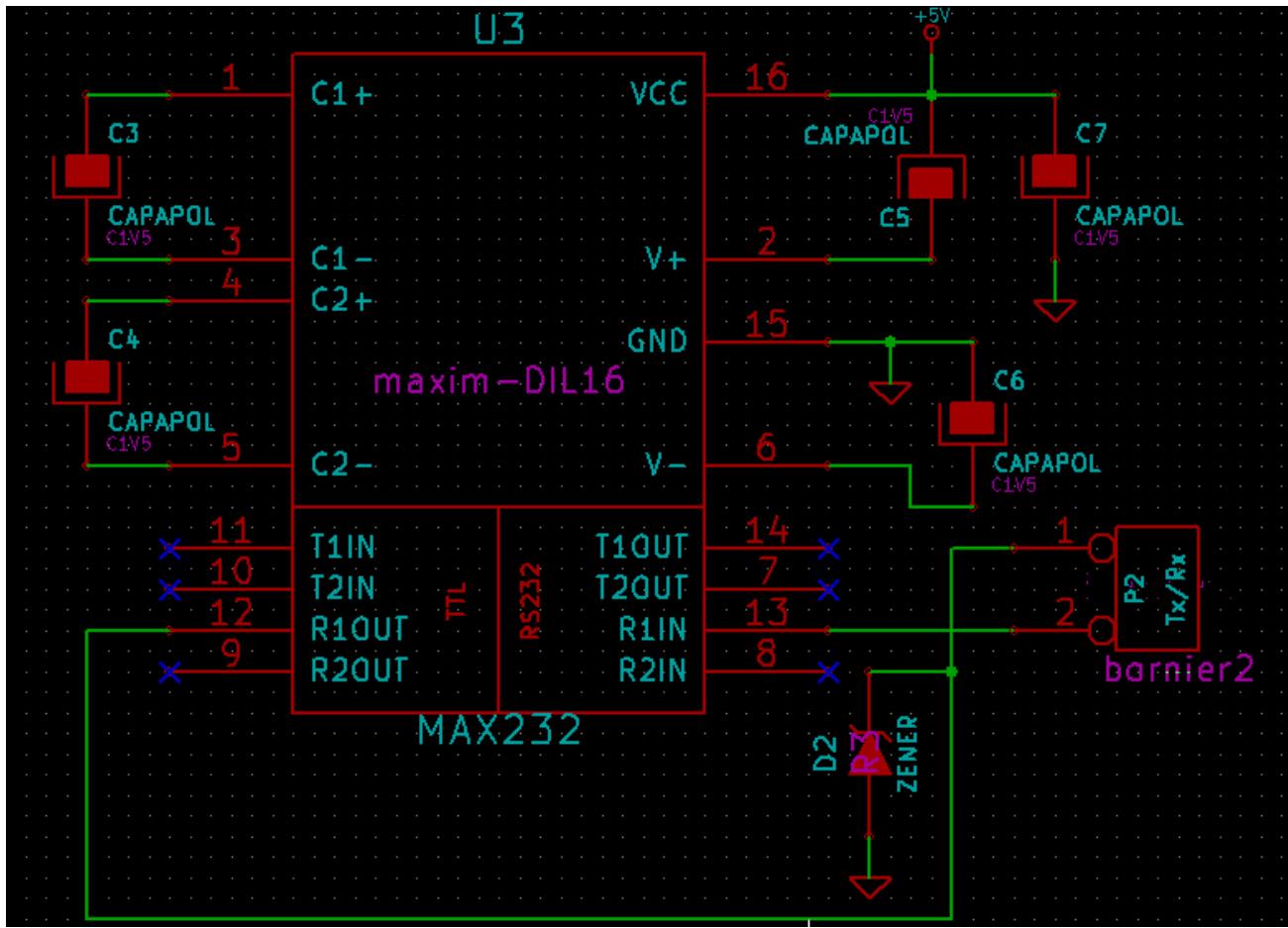


Ilustración 48: Esquemático del circuito del MAX232

Para alimentar este integrado es necesario 5v que lo tomamos de la salida del circuito anterior. Además se conectaron distintos capacitores (C4, C5, C6 y C7) que son propios para el integrado. Luego conectamos un diodo zener (D2) de 3.3v directo a la bornera (P2) que sirve para proteger la entrada de datos (Rx) a la raspberry. En dicha bornera (P2) se conectó la transmisión del dato del lector y la recepción del dato de la raspberry.

4.6.3 Alimentación electrocerradura

La electrocerradura necesita entre 9v y 12v para su correcto funcionamiento, además consume alrededor de 800mA por lo cual un pin de la raspberry no puede excitar la bobina para lograr abrir la puerta. Es por eso que utilizamos un optocoplador H11L1 para separar potencias y un Rele de 12V para alimentar la electrocerradura.

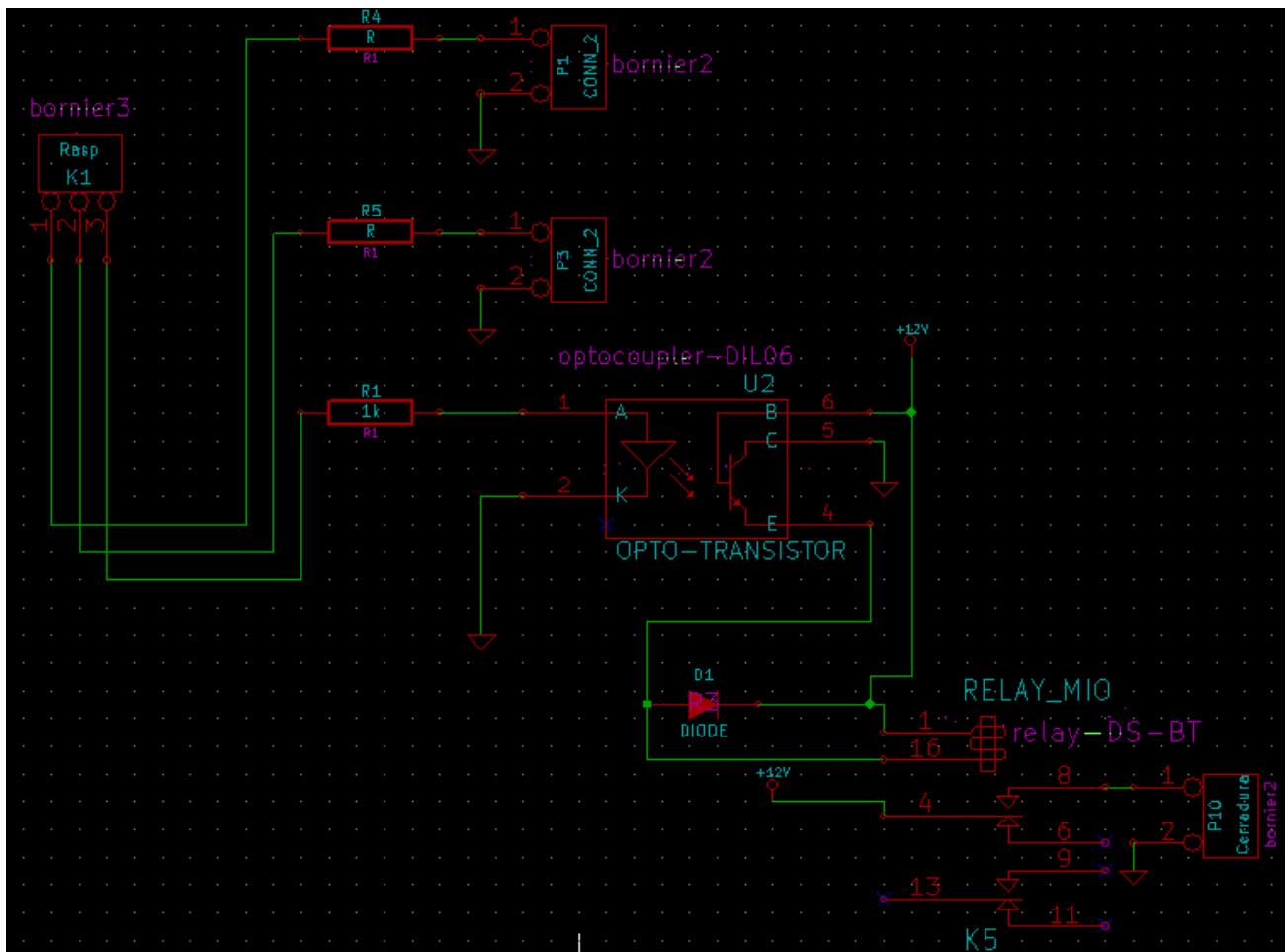


Ilustración 49: Esquemático del circuito de la electrocerradura

Básicamente cuando el script control.py decide abrir la puerta, la raspberry envía un pulso por el GPIO y éste llega al optocoplador (U2). Éste, al activarse con muy poca corriente, permite el paso de la tensión hacia el relé (K5) en donde lo activa. Al exitarse la bobina del relé, le entrega 12v a la bornera (P10) en donde allí está conectada la electrocerradura.

Por último agregamos dos borneras más (P1 y P3) conectadas a dos resistencias y luego a dos pines del GPIO de la raspberry. Esto está diseñado para poder conectar dos leds de control para poder visualizar de manera externa el estado en el que se encuentra el sistema.

4.6.4 Esquemático final

Luego de haber explicado detalladamente los circuitos utilizados, mostramos una imagen general del esquemático utilizado en este sistema.

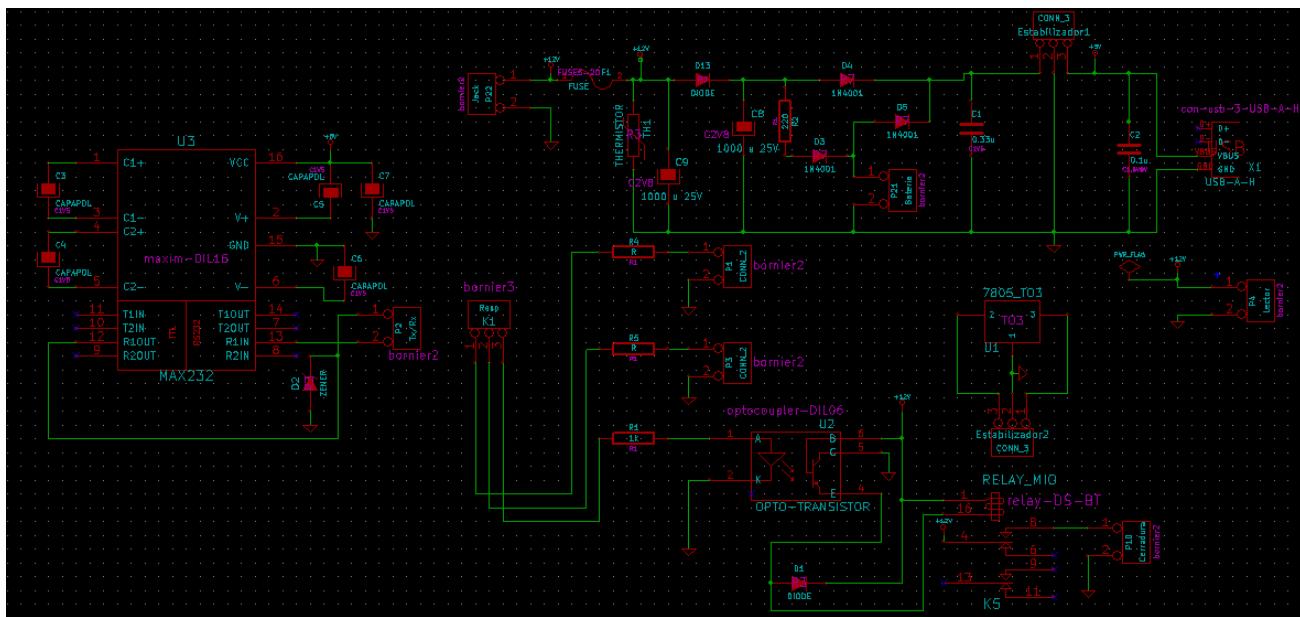


Ilustración 50: Esquemático del circuito general

4.6.4.1 Componentes

- MAX232
 - USB hembra
 - Jack para 12V
 - Optoacoplador H11L1
 - Relé de 12V
 - Resistencia de 1Kohm
 - Diodo Zener de 3.3V
 - Diodo rectificador 2N2222
 - Integrado L7805 TO-3
 - Capacitor cerámico de 1uF y de 33uF
 - Fusible

Capítulo 5 – Testing Integral del sistema

En este capítulo se realiza un testing completo del sistema, integrando todas las iteraciones, para comprobar el correcto funcionamiento del Sistema Integral de control de acceso inteligente.

5.1 Cuadro de análisis

Testing integral del sistema	
Requerimientos	R1 – R2 – R3 – R4 – R5 – R6 – R7 – R8 – R9 – R11
Propósito	Comprobar el funcionamiento del sistema integral de seguridad cuando intenta ingresar un usuario que no está habilitado en la base de datos del sistema. Se controlará el funcionamiento del sistema de acceso, la vigilancia, la página web, la comunicación de voz y los scripts de seguridad del sistema.
Entorno de Ejecución	Sistema operativo Raspbian de la placa Raspberry Pi. Servidor web Django.
Inicialización	Establecer la comunicación entre la placa Raspberry Pi y el lector RFID mediante el circuito estabilizador de tensión. Ejecutar el script “ control.py ”. Ejecutar el servicio de base de datos “ SQLite3 ”. Conectar la cámara web al puerto USB de la placa Raspberry Pi. Ejecutar el software “ motion ”. Ejecutar el servidor “ Django ”. Iniciar la página web. Ejecutar el servidor “ janus ”. Ejecutar el servidor “ asterisk ”. Inicializar el softphone “ Twinkle ”. Conectar parlantes en la placa Raspberry Pi. Script “ proceso.py ”. Ejecutar el software “ Cron ”.
Finalización	Usuario externo ingresa al Laboratorio a través de una habilitación del administrador utilizando la página web del sistema.
Acciones	Usuario externo intenta ingresar al Laboratorio con una tarjeta de proximidad que ha sido dada de baja en el sistema. Al denegarse su acceso, el administrador visualiza el exterior del laboratorio por la cámara web en la sección “Home” de la página web.

	<p>El administrador del sistema establece una comunicación de voz con el botón “Llamar” de la sección “home” de la página web.</p> <p>Por último el administrador permite el ingreso del usuario externo abriendo la puerta de forma remota a través del botón “Abrir Puerta” en la sección “Home” de la página web.</p>
Resultados	
Resultados Esperados	<p>Cuando el usuario externo intenta ingresar al laboratorio con la tarjeta de proximidad que no esta habilitada en el sistema, el control de acceso debe denegar su ingreso.</p> <p>El administrador debe poder visualizar el exterior del Laboratorio mediante la cámara web para comprobar que existe un usuario externo intentando ingresar.</p> <p>La comunicación de voz debe iniciarse y finalizarse de forma correcta.</p> <p>La puerta se debe abrir para permitir el acceso una vez que el administrador envie la señal de apertura de la misma de forma remota.</p> <p>El script de control de acceso no debe fallar y debe estar activo en todo momento.</p>
Resultados Obtenidos	<p>El control de acceso denegó el ingreso al usuario no autorizado.</p> <p>El exterior del laboratorio se visualizó correctamente mediante la cámara y la página web del sistema.</p> <p>La comunicación de voz se inicio y finalizó correctamente.</p> <p>La puerta fue abierta de manera remota por parte del administrador.</p> <p>El script del control de acceso no tuvo fallas, por lo tanto no se reportaron errores ni se reiniciaron los servicios.</p>

Tabla 31: Cuadro de análisis de testing integral del sistema

5.2 Evidencia

5.2.1 Intento de ingreso de usuario externo

El usuario externo intento ingresar al Laboratorio con la siguiente tarjeta de proximidad RFID:

- Tarjeta 2: 310094FC3D -----> REGISTRADA E INACTIVA

La tarjeta fue aproximada al lector RFID.

```
#####
Clave capturada: 310094FC3D
Conexion con Base de Datos database.sqlite exitosa
Usuario Registrado y No Activo
```

Ilustración 51: Evidencia intento de ingreso de usuario externo

5.2.2 Visualización de cámara web en tiempo real

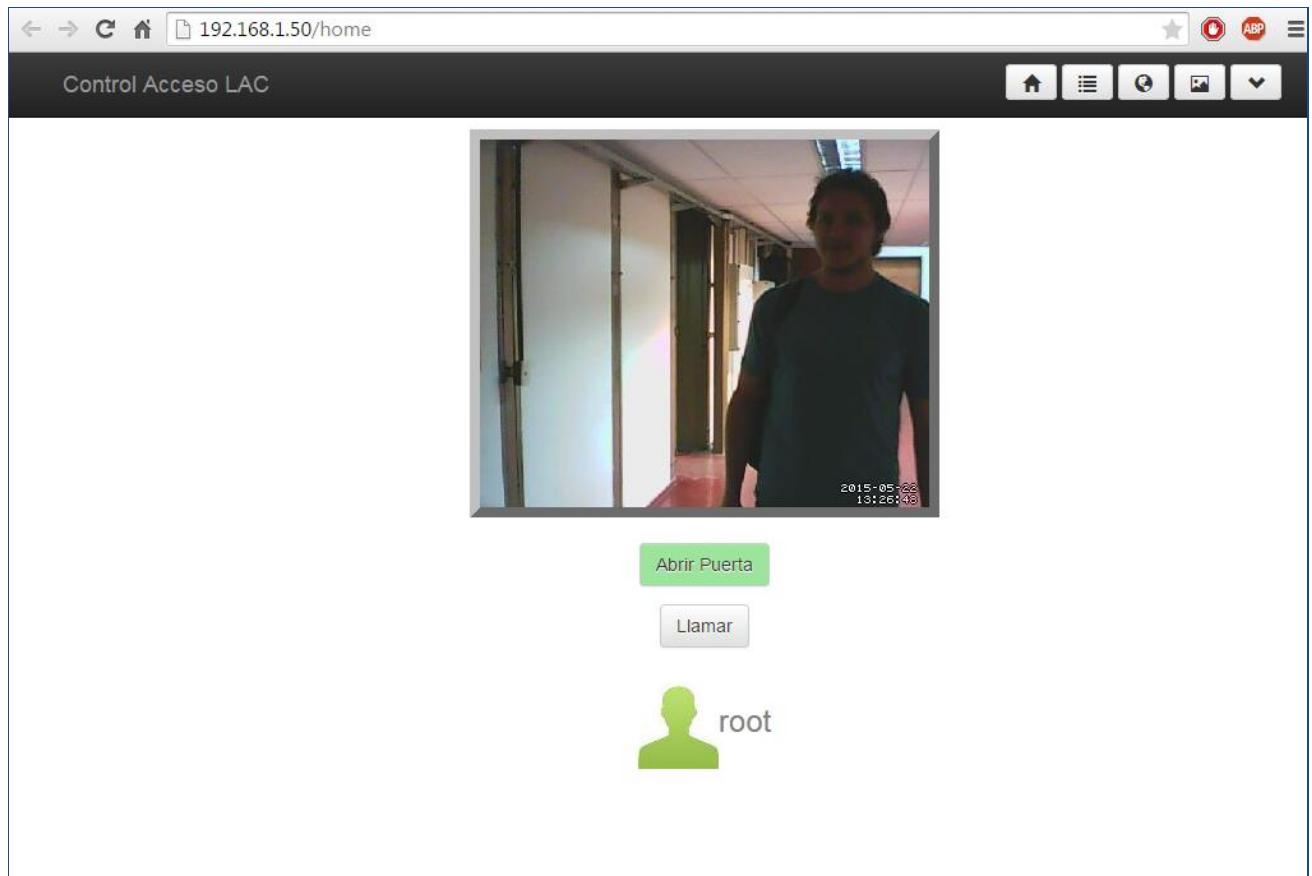


Ilustración 52: Evidencia visualización de cámara web en tiempo real

5.2.3 Comunicación de voz via VoIP

```
##### Inicializacion de sesion en pasarela WebRTC JANUS #####
```

```
Library static/js/adapter.js loaded
```

```
janus.js:94 Library initialized: true
```

```
janus.js:118 Using REST API to contact Janus
```

```
janus.js:120 http://192.168.1.50:8090/janus
```

```
janus.js:390 Create session:
```

```
janus.js:391 Object {janus: "success", transaction: "avdHB4EY3coH", data: Object}
```

```
janus.js:400 Created session: 3381987266
```

```
janus.js:163 Long poll...
janus.js:595 Create handle:
janus.js:596 Object {janus: "success", session_id: 3381987266, transaction: "wLu5cFWLjd2", data: Object}
janus.js:603 Created handle: 3413622820
siptest.js:89 Plugin attached! (janus.plugin.sip, id=3413622820)
janus.js:673 Sending message to plugin (handle=3413622820):
janus.js:674 Object {janus: "message", body: Object, transaction: "iVMhewRESM8m"}
siptest.js:350 This is a SIP audio call (dovideo=false)

##### Registro del Softphone con ASTERISK #####
siptest.js:134 :: Got a message :::
siptest.js:135 {"sip":"event","result":{"event":"registered","username":"6001"}}
siptest.js:154 Successfully registered as 6001!

#####
Comunicacion VoIP #####
siptest.js:134 :: Got a message :::
siptest.js:135 {"sip":"event","result":{"event":"accepted","username":"sip:6000@192.168.1.50"}}
siptest.js:225 sip:6000@192.168.1.50 accepted the call!
janus.js:1224 Remote description accepted!
janus.js:933 Handling Remote Stream:
janus.js:934 MediaStreamEvent {stream: MediaStream, path: NodeList[0], cancelBubble: false, returnValue: true, srcElement: RTCPeerConnection...}
janus.js:938 Starting bitrate monitor
```

5.2.4Apertura remota de puerta

```
[22/May/2015 13:47:06] "GET /home HTTP/1.1" 200 7277
Envio de señal de apertura de puerta
snapshot for thread 0
Done
GPIO.output(3) high...
Puerta abierta satisfactoriamente
Waiting 3 seconds...
GPIO.output(3) Low...
[22/May/2015 13:47:22] "POST /openDoor HTTP/1.1" 302 0
[22/May/2015 13:47:24] "GET /home HTTP/1.1" 200 7277
```

Ilustración 53: Evidencia apertura remota de puerta

Capítulo 6 - Conclusiones y trabajos futuros

6.1 Conclusión

En este trabajo se propuso implementar un sistema integral vigilancia y control de acceso inteligente de bajo costo, con las funcionalidades fundamentales, y adaptable a diversos ambientes (hogares, edificios, oficinas, etc.)

Para lograr esto fue necesario, como primera instancia, realizar una investigación de los sistemas embebidos existentes así como también de los distintos sistemas de control de acceso presentes en el mercado, y así poder ver las fortalezas y debilidades que estos presentan. A continuación se realizó la arquitectura y el diseño del sistema para luego implementarlo en el Laboratorio de Arquitectura de Computadoras de la Facultad de Ciencias Exactas, Físicas y Naturales de la Universidad Nacional de Córdoba.

En base a los resultados obtenidos en el presente trabajo, se puede concluir, que se cumplieron los objetivos del proyecto, ya que se logró un sistema de acceso integral con todas funcionalidades propuestas, de bajo costo y fácil de ocultar.

El cual contiene una interfaz web que es intuitiva y simple de usar. A su vez, el sistema consta de 3 “cajas negras” (Raspberry Pi y circuitos estabilizadores de tensión, Lector RFID, vigilancia y comunicación de voz), logrando un diseño de simple interconexión.

El uso del framework Django permitió administrar el sistema de control acceso mediante una página web. Esto ahorró tiempos y complicaciones que se pueden generar al desarrollar un software de administración.

Se debe destacar la gran integración que se alcanzó entre el control de acceso y la vigilancia, ya que mediante el uso de la web de control y administración se puede acceder a la visualización de la cámara web, establecer una comunicación de voz entre usuarios internos y externos del establecimiento, abrir la puerta remotamente para habilitar el ingreso y también visualizar eventos de ingresos realizados.

La información teórica recopilada en este informe contiene documentación que permite la comprensión del sistema general, desde la introducción a los controles de acceso, pasando por la descripción de los sistemas embebidos utilizados y concluyendo con la información de cada software instalado y utilizado en el proyecto.

6.2 Trabajos futuros

En el transcurso de este proyecto integrador se fueron hallando diversos campos a investigar para realizar mejoras al sistema. Estos se enumeran a continuación:

- Mejorar el procesamiento de audio para así mejorar el rendimiento de la comunicación de voz entre los usuarios internos y externos al establecimiento.
- Implementar nuevas funcionalidades en la sección de Franjas Horarias para programar tiempos de actividad limitados para diferentes usuarios y dar de baja a los mismos cuando se cumpla dicho tiempo límite.
- Implementar control de acceso biométrico (huellas digitales, retina) y por clave en teclado.
- Implementar un botón en el exterior del establecimiento que genere un aviso en las computadoras que tengan activa la página web administradora del sistema de seguridad, para así dar conocimiento de que existe un usuario esperando en el exterior.
- Generar un micro kernel para restaurar una imagen de memoria SD y lograr un sistema tolerante a fallos.
- Sensor magnético colocado en la puerta para detectar que la misma quedó abierta o fue abierta manualmente.
- Redireccionar una llamada del usuario externo (presión del botón) por voz por Ip como si fuese una llamada normal a un teléfono normal.

Bibliografía

1. **Wikipedia.** Información general sobre Raspberry Pi. [En línea]
http://es.wikipedia.org/wiki/Raspberry_Pi.
2. **Social Compare.** Comparación de placas ARM. [En línea]
<http://socialcompare.com/en/comparison/arm-boards>.
3. **Diverteka.** GPIO de Raspberry Pi. [En línea] <http://www.diverteka.com/?p=1370>.
4. **Cubieboard.** Placa de código abierto Cubieboard. [En línea]
<http://docs.cubieboard.org/products/start#a10-cubieboard>.
5. **Wikipedia.** Arduino. [En línea] <http://es.wikipedia.org/wiki/Arduino>.
6. —. Información general del Sistema Operativo Raspbian. [En línea]
<http://es.wikipedia.org/wiki/Raspbian>.
7. **Prototipando.** Información general del Sistema Operativo Pidora. [En línea]
<http://www.prototipando.es/prototipos/raspberry-pi/33-pidora-fedora-for-raspberry-pi>.
8. **Archlinux.** Información general del Sistema Operativo Archlinux ARM. [En línea]
<http://archlinuxarm.org/>.
9. **OpenELEC.** Información general del Sistema Operativo OpenELEC. [En línea]
<http://es.wikipedia.org/wiki/OpenELEC>.
10. **Gizmologia.** Información general del Sistema Operativo Raspbmc. [En línea]
<http://gizmologia.com/2013/02/raspbmc-raspberry-pi-multimedia>.
11. **Wikipedia.** Información general sobre SQLite. [En línea] <http://es.wikipedia.org/wiki/SQLite>.
12. **Usemossoftwarelibre.** Características de SQLite. [En línea]
<https://usemossoftwarelibre.wordpress.com/cc/tutorial-sqlite-en-espanol/>.
13. **Wikipedia.** MySQL. [En línea] <http://es.wikipedia.org/wiki/MySQL>.
14. **Alegsa.** Definición de MySQL. [En línea] <http://www.alegsa.com.ar/Dic/mysql.php>.
15. **Wikipedia.** PostgreSQL. [En línea] <http://es.wikipedia.org/wiki/PostgreSQL>.
16. —. C Lenguaje de programación. [En línea]
http://es.wikipedia.org/wiki/C_%28lenguaje_de_programaci%C3%B3n%29.
17. —. Python. [En línea] <http://es.wikipedia.org/wiki/Python>.

18. —. Java Lenguaje de programación. [En línea]
http://es.wikipedia.org/wiki/Java_%28lenguaje_de_programaci%C3%B3n%29.
19. —. Servidor HTTP Apache. [En línea] http://es.wikipedia.org/wiki/Servidor_HTTP_Apache.
20. —. Nginx. [En línea] <http://es.wikipedia.org/wiki/Nginx>.
21. **Django.** Información general del entorno web Django. [En línea] <http://django.es/>.
22. **Maestrosdelweb.** Instalación y configuración del entorno web Django. [En línea]
<http://www.maestrosdelweb.com/curso-django-instalacion-y-primera-aplicacion/>.
23. **Wikiasterisk.** Introducción a Asterisk. [En línea]
<http://www.wikiasterisk.com/index.php?title=Introducci%C3%B3n>.
24. **Quarea.** Funcionalidades de Asterisk. [En línea]
http://www.quarea.com/es/asterisk_funcionalidades_basicas_avanzadas.
25. **Todoasterisk.** Instalacion y configuracion de Asterisk. [En línea]
<http://todoasterisk.blogspot.com.ar/2009/08/installacion-y-configuracion-de-asterisk.html>.
26. **Wikipedia.** 3CX. [En línea] <http://es.wikipedia.org/wiki/3CX>.
27. —. Elastix. [En línea] <http://es.wikipedia.org/wiki/Elastix>.
28. **Diverteka.** Instalación de Software Motion para Vigilancia. [En línea]
<http://www.diverteka.com/?p=709>.
29. **torpes, Raspberry para.** Instalación y Configuración de Software de vigilancia. [En línea]
<http://raspberryparatorpes.net/installacion/conectar-una-webcam-con-motion-en-raspberry-pi/>.
30. **Lincudo.** Video streaming live con Raspberry Pi. [En línea]
<http://www.lincudo.org.es/2013/12/video-streaming-live-con-raspberry-pi.html>.
31. **Informatica-hoy.** Introducción a Softphones. [En línea] <http://www.informatica-hoy.com.ar/voz-ip-voip/Que-es-un-SoftPhone.php>.
32. **Twinkle.** Información general del softphone Twinkle. [En línea] <http://www.twinklephone.com/>.
33. **Wikipedia.** Linphone. [En línea] <http://es.wikipedia.org/wiki/Linphone>.
34. **3CX.** Información general del Sistema webRTC. [En línea] <http://www.3cx.es/webrtc/>.
35. **Github.** Información general y elementos de instalación del WebRTC Gateway Janus. [En línea]
<https://github.com/meetecho/janus-gateway>.
36. **SENA.** Manual de instalación y configuración de SAMBA. [En línea]
<https://docs.google.com/document/d/1hvLcdTf2U-ihbnR7FamqIV49no8MG5Dttn9Ru40WU8/edit>.

37. **Elboby.** Instalación y configuración de SAMBA en Raspberry Pi. [En línea]

<http://www.elboby.com/2012/10/instalacion-y-configuracion-de-samba-en-raspberry-pi/>.

ANEXOS

A - Configuración Motion

En este apartado nos encargaremos de modificar unos cuantos parámetros en el fichero de configuración de Motion con el siguiente comando:

```
sudonano/etc/motion/motion.conf
```

Y modificamos los parámetros para dejarlos de la siguiente manera:

```
# Rename this distribution example file to motion.conf
#
# This config file was generated by motion 3.2.12

#####
# Daemon
#####

# Start in daemon (background) mode and release terminal (default: off)
daemon on

# File to store the process ID, also called pid file. (default: not defined)
process_id_file /var/run/motion/motion.pid

#####
# Basic Setup Mode
#####

# Start in Setup-Mode, daemon disabled. (default: off)
setup_mode off

#####
# Capture device options
#####

# Videodevice to be used for capturing (default /dev/video0)
# for FreeBSD default is /dev/bktr0
videodevice /dev/video0

# v4l2_palette allows to choose preferable palette to be use by motion
# to capture from those supported by your videodevice. (default: 8)
# E.g. if your videodevice supports both V4L2_PIX_FMT_SBGGR8 and
# V4L2_PIX_FMT_MJPEG then motion will by default use V4L2_PIX_FMT_MJPEG.
# Setting v4l2_palette to 1 forces motion to use V4L2_PIX_FMT_SBGGR8
# instead.
#
# Values :
# V4L2_PIX_FMT_SN9C10X : 0 'S910'
# V4L2_PIX_FMT_SBGGR8 : 1 'BA81'
```

```

# V4L2_PIX_FMT_MJPEG : 2 'MJPEG'
# V4L2_PIX_FMT_JPEG : 3 'JPEG'
# V4L2_PIX_FMT_RGB24 : 4 'RGB3'
# V4L2_PIX_FMT_UYVY : 5 'UYVY'
# V4L2_PIX_FMT_YUYV : 6 'YUYV'
# V4L2_PIX_FMT_YUV422P : 7 '422P'
# V4L2_PIX_FMT_YUV420 : 8 'YU12'
v4l2_palette 8

# Tuner device to be used for capturing using tuner as source (default
/dev/tuner0)
# This is ONLY used for FreeBSD. Leave it commented out for Linux
; tunerdevice /dev/tuner0

# The video input to be used (default: 8)
# Should normally be set to 0 or 1 for video/TV cards, and 8 for USB cameras
input 8

# The video norm to use (only for video capture and TV tuner cards)
# Values: 0 (PAL), 1 (NTSC), 2 (SECAM), 3 (PAL NC no colour). Default: 0
# (PAL)
norm 0

# The frequency to set the tuner to (kHz) (only for TV tuner cards) (default:
# 0)
frequency 0

# Rotate image this number of degrees. The rotation affects all saved images
# as
# well as mpeg movies. Valid values: 0 (default = no rotation), 90, 180 and
# 270.
rotate 0

# Image width (pixels). Valid range: Camera dependent, default: 352
width 352

# Image height (pixels). Valid range: Camera dependent, default: 288
height 288

# Maximum number of frames to be captured per second.
# Valid range: 2-100. Default: 100 (almost no limit).
framerate 2

# Minimum time in seconds between capturing picture frames from the camera.
# Default: 0 = disabled - the capture rate is given by the camera framerate.
# This option is used when you want to capture images at a rate lower than 2
# per second.
minimum_frame_time 0

# URL to use if you are using a network camera, size will be autodetected
# (incl http:// ftp:// or file:///)
# Must be a URL that returns single jpeg pictures or a raw mjpeg stream.
Default: Not defined
; netcam_url value

# Username and password for network camera (only if required). Default: not
# defined
# Syntax is user:password

```

```

; netcam_userpass value

# The setting for keep-alive of network socket, should improve performance on
compatible net cameras.
# 1.0:           The historical implementation using HTTP/1.0, closing the
socket after each http request.
# keep_alive:   Use HTTP/1.0 requests with keep alive header to reuse the same
connection.
# 1.1:           Use HTTP/1.1 requests that support keep alive as default.
# Default: 1.0
; netcam_http 1.0

# URL to use for a netcam proxy server, if required, e.g. "http://myproxy".
# If a port number other than 80 is needed, use "http://myproxy:1234".
# Default: not defined
; netcam_proxy value

# Set less strict jpeg checks for network cameras with a poor/buggy firmware.
# Default: off
netcam_tolerant_check off

# Let motion regulate the brightness of a video device (default: off).
# The auto_brightness feature uses the brightness option as its target value.
# If brightness is zero auto_brightness will adjust to average brightness
value 128.
# Only recommended for cameras without auto brightness
auto_brightness off

# Set the initial brightness of a video device.
# If auto_brightness is enabled, this value defines the average brightness
level
# which Motion will try and adjust to.
# Valid range 0-255, default 0 = disabled
brightness 0

# Set the contrast of a video device.
# Valid range 0-255, default 0 = disabled
contrast 0

# Set the saturation of a video device.
# Valid range 0-255, default 0 = disabled
saturation 0

# Set the hue of a video device (NTSC feature).
# Valid range 0-255, default 0 = disabled
hue 0

#####
# Round Robin (multiple inputs on same video device name)
#####

# Number of frames to capture in each roundrobin step (default: 1)
roundrobin_frames 1

# Number of frames to skip before each roundrobin step (default: 1)
roundrobin_skip 1

```

```

# Try to filter out noise generated by roundrobin (default: off)
switchfilter off

#####
# Motion Detection Settings:
#####

# Threshold for number of changed pixels in an image that
# triggers motion detection (default: 1500)
;threshold 1500

# Automatically tune the threshold down if possible (default: off)
threshold_tune off

# Noise threshold for the motion detection (default: 32)
noise_level 32

# Automatically tune the noise threshold (default: on)
noise_tune off

# Despeckle motion image using (e)rode or (d)ilate or (l)abel (Default: not
defined)
# Recommended value is EedDl. Any combination (and number of) of E, e, d, and
D is valid.
# (l)abeling must only be used once and the 'l' must be the last letter.
# Comment out to disable
despeckle EedDl

# Detect motion in predefined areas (1 - 9). Areas are numbered like that: 1
2 3
# A script (on_area_detected) is started immediately when motion is 4
5 6
# detected in one of the given areas, but only once during an event. 7
8 9
# One or more areas can be specified with this option. (Default: not defined)
; area_detect value

# PGM file to use as a sensitivity mask.
# Full path name to. (Default: not defined)
; mask_file value

# Dynamically create a mask file during operation (default: 0)
# Adjust speed of mask changes from 0 (off) to 10 (fast)
smart_mask_speed 0

# Ignore sudden massive light intensity changes given as a percentage of the
picture
# area that changed intensity. Valid range: 0 - 100 , default: 0 = disabled
lightswitch 0

# Picture frames must contain motion at least the specified number of frames
# in a row before they are detected as true motion. At the default of 1, all
# motion is detected. Valid range: 1 to thousands, recommended 1-5
minimum_motion_frames 1

# Specifies the number of pre-captured (buffered) pictures from before motion
# was detected that will be output at motion detection.

```

```

# Recommended range: 0 to 5 (default: 0)
# Do not use large values! Large values will cause Motion to skip video
frames and
# cause unsmooth mpegs. To smooth mpegs use larger values of post_capture
instead.
pre_capture 0

# Number of frames to capture after motion is no longer detected (default: 0)
post_capture 0

# Gap is the seconds of no motion detection that triggers the end of an event
# An event is defined as a series of motion images taken within a short
timeframe.
# Recommended value is 60 seconds (Default). The value 0 is allowed and
disables
# events causing all Motion to be written to one single mpeg file and no
pre_capture.
gap 60

# Maximum length in seconds of an mpeg movie
# When value is exceeded a new mpeg file is created. (Default: 0 = infinite)
;max_mpeg_time 0

# Always save images even if there was no motion (default: off)
output_all off

#####
# Image File Output
#####

# Output 'normal' pictures when motion is detected (default: on)
# Valid values: on, off, first, best, center
# When set to 'first', only the first picture of an event is saved.
# Picture with most motion of an event is saved when set to 'best'.
# Picture with motion nearest center of picture is saved when set to
'center'.
# Can be used as preview shot for the corresponding movie.
output_normal off

# Output pictures with only the pixels moving object (ghost images) (default:
off)
output_motion off

# The quality (in percent) to be used by the jpeg compression (default: 75)
quality 75

# Output ppm images instead of jpeg (default: off)
ppm off

#####
# FFMPEG related options
# Film (mpeg) file output, and deinterlacing of the video input
# The options movie_filename and timelapse_filename are also used
# by the ffmpeg feature
#####

```

```

# Use ffmpeg to encode mpeg movies in realtime (default: off)
ffmpeg_cap_new off

# Use ffmpeg to make movies with only the pixels moving
# object (ghost images) (default: off)
ffmpeg_cap_motion off

# Use ffmpeg to encode a timelapse movie
# Default value 0 = off - else save frame every Nth second
ffmpeg_timelapse 0

# The file rollover mode of the timelapse video
# Valid values: hourly, daily (default), weekly-sunday, weekly-monday,
monthly, manual
ffmpeg_timelapse_mode daily

# Bitrate to be used by the ffmpeg encoder (default: 400000)
# This option is ignored if ffmpeg_variable_bitrate is not 0 (disabled)
ffmpeg_bps 500000

# Enables and defines variable bitrate for the ffmpeg encoder.
# ffmpeg_bps is ignored if variable bitrate is enabled.
# Valid values: 0 (default) = fixed bitrate defined by ffmpeg_bps,
# or the range 2 - 31 where 2 means best quality and 31 is worst.
ffmpeg_variable_bitrate 0

# Codec to used by ffmpeg for the video compression.
# Timelapse mpegs are always made in mpeg1 format independent from this
option.
# Supported formats are: mpeg1 (ffmpeg-0.4.8 only), mpeg4 (default), and
msmpeg4.
# mpeg1 - gives you files with extension .mpg
# mpeg4 or msmpeg4 - gives you files with extension .avi
# msmpeg4 is recommended for use with Windows Media Player because
# it requires no installation of codec on the Windows client.
# swf - gives you a flash film with extension .swf
# flv - gives you a flash video with extension .flv
# ffvl - FF video codec 1 for Lossless Encoding ( experimental )
# mov - QuickTime ( testing )
ffmpeg_video_codec swf

# Use ffmpeg to deinterlace video. Necessary if you use an analog camera
# and see horizontal combing on moving objects in video or pictures.
# (default: off)
ffmpeg_deinterlace off

#####
# Snapshots (Traditional Periodic Webcam File Output)
#####

# Make automated snapshot every N seconds (default: 0 = disabled)
snapshot_interval 0

#####
# Text Display
# %Y = year, %m = month, %d = date,

```

```

# %H = hour, %M = minute, %S = second, %T = HH:MM:SS,
# %v = event, %q = frame number, %t = thread (camera) number,
# %D = changed pixels, %N = noise level, \n = new line,
# %i and %J = width and height of motion area,
# %K and %L = X and Y coordinates of motion center
# %C = value defined by text_event - do not use with text_event!
# You can put quotation marks around the text to allow
# leading spaces
#####
#
# Locate and draw a box around the moving object.
# Valid values: on, off and preview (default: off)
# Set to 'preview' will only draw a box in preview_shot pictures.
locate off

# Draws the timestamp using same options as C function strftime(3)
# Default: %Y-%m-%d\n%T = date in ISO format and time in 24 hour clock
# Text is placed in lower right corner
text_right %Y-%m-%d\n%T-%q

# Draw a user defined text on the images using same options as C function
strftime(3)
# Default: Not defined = no text
# Text is placed in lower left corner
; text_left CAMERA %t

# Draw the number of changed pixels on the images (default: off)
# Will normally be set to off except when you setup and adjust the motion
settings
# Text is placed in upper right corner
text_changes off

# This option defines the value of the special event conversion specifier %C
# You can use any conversion specifier in this option except %C. Date and
time
# values are from the timestamp of the first image in the current event.
# Default: %Y%m%d%H%M%S
# The idea is that %C can be used filenames and text_left/right for creating
# a unique identifier for each event.
text_event %Y%m%d%H%M%S

# Draw characters at twice normal size on images. (default: off)
text_double off

#####
#
# Target Directories and filenames For Images And Films
# For the options snapshot_, jpeg_, mpeg_ and timelapse_filename
# you can use conversion specifiers
# %Y = year, %m = month, %d = date,
# %H = hour, %M = minute, %S = second,
# %v = event, %q = frame number, %t = thread (camera) number,
# %D = changed pixels, %N = noise level,
# %i and %J = width and height of motion area,
# %K and %L = X and Y coordinates of motion center
# %C = value defined by text_event
# Quotation marks round string are allowed.
#####

```

```

# Target base directory for pictures and films
# Recommended to use absolute path. (Default: current working directory)
target_dir /usr/src/web/login/static/photo

# File path for snapshots (jpeg or ppm) relative to target_dir
# Default: %v-%Y%m%d%H%M%S-snapshot
# Default value is equivalent to legacy oldlayout option
# For Motion 3.0 compatible mode choose: %Y/%m/%d/%H/%M/%S-snapshot
# File extension .jpg or .ppm is automatically added so do not include this.
# Note: A symbolic link called lastsnap.jpg created in the target_dir will
always
# point to the latest snapshot, unless snapshot_filename is exactly
'lastsnap'
snapshot_filename %d-%m-%Y_%H-%M-%S

# File path for motion triggered images (jpeg or ppm) relative to target_dir
# Default: %v-%Y%m%d%H%M%S-%q
# Default value is equivalent to legacy oldlayout option
# For Motion 3.0 compatible mode choose: %Y/%m/%d/%H/%M/%S-%q
# File extension .jpg or .ppm is automatically added so do not include this
# Set to 'preview' together with best-preview feature enables special naming
# convention for preview shots. See motion guide for details
jpeg_filename %v-%Y%m%d%H%M%S-%q

# File path for motion triggered ffmpeg films (mpeg) relative to target_dir
# Default: %v-%Y%m%d%H%M%S
# Default value is equivalent to legacy oldlayout option
# For Motion 3.0 compatible mode choose: %Y/%m/%d/%H%M%S
# File extension .mpg or .avi is automatically added so do not include this
# This option was previously called ffmpeg_filename
movie_filename %v-%Y%m%d%H%M%S

# File path for timelapse mpegs relative to target_dir
# Default: %Y%m%d-timelapse
# Default value is near equivalent to legacy oldlayout option
# For Motion 3.0 compatible mode choose: %Y/%m/%d-timelapse
# File extension .mpg is automatically added so do not include this
timelapse_filename %Y%m%d-timelapse

#####
# Live Webcam Server
#####

# The mini-http server listens to this port for requests (default: 0 =
disabled)
webcam_port 8081

# Quality of the jpeg (in percent) images produced (default: 50)
webcam_quality 80

# Output frames at 1 fps when no motion is detected and increase to the
# rate given by webcam_maxrate when motion is detected (default: off)
webcam_motion off

# Maximum framerate for webcam streams (default: 1)
webcam_maxrate 1

```

```

# Restrict webcam connections to localhost only (default: on)
webcam_localhost off

# Limits the number of images per connection (default: 0 = unlimited)
# Number can be defined by multiplying actual webcam rate by desired number
# of seconds
# Actual webcam rate is the smallest of the numbers framerate and
webcam_maxrate
webcam_limit 0

#####
# HTTP Based Control
#####

# TCP/IP port for the http server to listen on (default: 0 = disabled)
control_port 8080

# Restrict control connections to localhost only (default: on)
control_localhost off

# Output for http server, select off to choose raw text plain (default: on)
control_html_output off

# Authentication for the http based control. Syntax username:password
# Default: not defined (Disabled)
; control_authentication username:password

#####

# Tracking (Pan/Tilt)
#####

# Type of tracker (0=none (default), 1=stepper, 2=iomojo, 3=pwc, 4=generic,
# 5=uvcvideo)
# The generic type enables the definition of motion center and motion size to
# be used with the conversion specifiers for options like on_motion_detected
track_type 0

# Enable auto tracking (default: off)
track_auto off

# Serial port of motor (default: none)
; track_port value

# Motor number for x-axis (default: 0)
track_motorx 0

# Motor number for y-axis (default: 0)
track_motory 0

# Maximum value on x-axis (default: 0)
track_maxx 0

# Maximum value on y-axis (default: 0)
track_maxy 0

```

```

# ID of an iomojo camera if used (default: 0)
track_iomojo_id 0

# Angle in degrees the camera moves per step on the X-axis
# with auto-track (default: 10)
# Currently only used with pwc type cameras
track_step_angle_x 10

# Angle in degrees the camera moves per step on the Y-axis
# with auto-track (default: 10)
# Currently only used with pwc type cameras
track_step_angle_y 10

# Delay to wait for after tracking movement as number
# of picture frames (default: 10)
track_move_wait 10

# Speed to set the motor to (stepper motor option) (default: 255)
track_speed 255

# Number of steps to make (stepper motor option) (default: 40)
track_stepsize 40

#####
# External Commands, Warnings and Logging:
# You can use conversion specifiers for the on_xxxx commands
# %Y = year, %m = month, %d = date,
# %H = hour, %M = minute, %S = second,
# %v = event, %q = frame number, %t = thread (camera) number,
# %D = changed pixels, %N = noise level,
# %i and %J = width and height of motion area,
# %K and %L = X and Y coordinates of motion center
# %C = value defined by text_event
# %f = filename with full path
# %n = number indicating filetype
# Both %f and %n are only defined for on_picture_save,
# on_movie_start and on_movie_end
# Quotation marks round string are allowed.
#####

# Do not sound beeps when detecting motion (default: on)
# Note: Motion never beeps when running in daemon mode.
quiet on

# Command to be executed when an event starts. (default: none)
# An event starts at first motion detected after a period of no motion
defined by gap
; on_event_start value

# Command to be executed when an event ends after a period of no motion
# (default: none). The period of no motion is defined by option gap.
; on_event_end value

# Command to be executed when a picture (.ppm|.jpg) is saved (default: none)
# To give the filename as an argument to a command append it with %f
; on_picture_save value

```

```

# Command to be executed when a motion frame is detected (default: none)
; on_motion_detected value

# Command to be executed when motion in a predefined area is detected
# Check option 'area_detect'.      (default: none)
; on_area_detected value

# Command to be executed when a movie file (.mpg|.avi) is created. (default:
none)
# To give the filename as an argument to a command append it with %f
; on_movie_start value

# Command to be executed when a movie file (.mpg|.avi) is closed. (default:
none)
# To give the filename as an argument to a command append it with %f
; on_movie_end value

# Command to be executed when a camera can't be opened or if it is lost
# NOTE: There is situations when motion doesn't detect a lost camera!
# It depends on the driver, some drivers don't detect a lost camera at all
# Some hang the motion thread. Some even hang the PC! (default: none)
; on_camera_lost value

#####
# Common Options For MySQL and PostgreSQL database features.
# Options require the MySQL/PostgreSQL options to be active also.
#####

# Log to the database when creating motion triggered image file (default:
on)
sql_log_image off

# Log to the database when creating a snapshot image file (default: on)
sql_log_snapshot off

# Log to the database when creating motion triggered mpeg file (default: off)
sql_log_mpeg off

# Log to the database when creating timelapse mpeg file (default: off)
sql_log_timelapse off

# SQL query string that is sent to the database
# Use same conversion specifiers has for text features
# Additional special conversion specifiers are
# %n = the number representing the file_type
# %f = filename with full path
# Default value:
# insert into security(camera, filename, frame, file_type, time_stamp,
text_event) values('%t', '%f', '%q', '%n', '%Y-%m-%d %T', '%C')
sql_query insert into security(camera, filename, frame, file_type,
time_stamp, event_time_stamp) values('%t', '%f', '%q', '%n', '%Y-%m-%d %T',
'%C')

#####
# Database Options For MySQL
#####

```

```

# Mysql database to log to (default: not defined)
; mysql_db value

# The host on which the database is located (default: localhost)
; mysql_host value

# User account name for MySQL database (default: not defined)
; mysql_user value

# User password for MySQL database (default: not defined)
; mysql_password value

#####
# Database Options For PostgreSQL
#####

# PostgreSQL database to log to (default: not defined)
; pgsql_db value

# The host on which the database is located (default: localhost)
; pgsql_host value

# User account name for PostgreSQL database (default: not defined)
; pgsql_user value

# User password for PostgreSQL database (default: not defined)
; pgsql_password value

# Port on which the PostgreSQL database is located (default: 5432)
; pgsql_port 5432

#####
# Video Loopback Device (vloopback project)
#####

# Output images to a video4linux loopback device
# The value '-' means next available (default: not defined)
; video_pipe value

# Output motion images to a video4linux loopback device
# The value '-' means next available (default: not defined)
; motion_video_pipe value

#####
# Thread config files - One for each camera.
# Except if only one camera - You only need this config file.
# If you have more than one camera you MUST define one thread
# config file for each camera in addition to this config file.
#####

# Remember: If you have more than one camera you must have one
# thread file for each camera. E.g. 2 cameras requires 3 files:
# This motion.conf file AND thread1.conf and thread2.conf.
# Only put the options that are unique to each camera in the
# thread config files.

```

```
; thread /usr/local/etc/thread1.conf  
; thread /usr/local/etc/thread2.conf  
; thread /usr/local/etc/thread3.conf  
; thread /usr/local/etc/thread4.conf
```