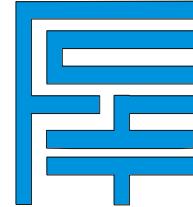


UNIVERSIDAD MAYOR DE SAN SIMÓN
FACULTAD DE CIENCIAS Y
TECNOLOGÍA
CARRERA DE INGENIERÍA DE SISTEMAS



**SISTEMA DE NOTIFICACIÓN EN TIEMPO REAL ORIENTADO A LA
MONITORIZACIÓN DE RECURSOS Y SERVICIOS DE UN SERVIDOR
GNU/LINUX**

Proyecto de Grado, Presentado Para Optar al Diploma
Académico de Licenciatura en Ingeniería De Sistemas.

Presentado por: GABRIEL EDGAR DELGADO ROCHA

Tutor: Ing. Jimmy Villarroel Novillo

Cochabamba – Bolivia

Noviembre, 2015

DEDICATORIA

A mis queridos padres, por ser el impulso, la voluntad y el coraje en mi vida.

AGRADECIMIENTOS

A DIOS por brindarme la felicidad y la dicha de vivir cada día en la grandeza de su incondicional amor.

A mi madre por toda la paciencia, cariño, por ser mi ejemplo de coraje, fuerza, humildad y paciencia.

A mi padre por brindarme las mejores oportunidades, ser el mejor ejemplo de voluntad y dedicación.

A Naira Romero por ser mi compañera, amiga, motivación y comprensión que llena de alegría mis días.

A mi familia por apoyarme, soportarme y fortalecerme todos estos años, con su cariño.

A mis amigos por todas las experiencias y conocimientos que compartieron conmigo.

FICHA RESUMEN

Los eventos que se generan en un servidor suelen tener información que los administradores de dichos servidores necesitan conocer en el momento en que estos ocurren o cuando dichos eventos están por suceder, hoy en día existe un gran número de aplicaciones que monitorizan el estado de los servidores, los cuales se enfocan en obtener bastante información de los eventos generados pero dichas aplicaciones tienden a ser soluciones costosas o tienden a presentar una configuración complicada para usuarios con pocos conocimientos en el área, estos sistemas tienden también a omitir u ofrecer pocas alternativas relacionadas a las notificaciones en tiempo real. Los sistemas de notificaciones en tiempo real aplicados a la monitorización de servidores permiten enviar y recibir información relacionada al estado de los recursos y servicios de un servidor en este caso un servidor del tipo GNU/Linux, por lo expuesto utilizando como tecnologías de comunicación REST y WebSockets para el intercambio de información se desarrolló e integró un conjunto de aplicaciones que ofrecen servicios web que pueden ser consumidas por aplicaciones cliente web y móviles. Dado que para un administrador de servidores es necesario conocer inmediatamente los eventos generados en sus servidores, las notificaciones en tiempo real representan una ayuda significativa en este tipo de tareas, aportando a la mejora de una herramienta tecnológica.

Para el desarrollo del sistema, se utilizaron tecnologías web tanto del lado del servidor como del lado del cliente, lo cual permitió separar el desarrollo del proyecto en diferentes aplicaciones las cuales fueron trabajadas haciendo uso de la metodología ágil SCRUM, por lo que en el transcurso de las diferentes iteraciones fueron integrándose y acoplando cada una de sus características que hacen de este conjunto de aplicaciones un sistema con un objetivo común, generar notificaciones en tiempo real para los usuarios, con información relevante del estado del servidor y sus recursos.

Se concluye que el objetivo principal fue alcanzado a cabalidad, ya que se pudo apreciar que el sistema cumple con todos los requerimientos y características propuestas, al beneficiar favorablemente a los usuarios administradores de servidores con sistemas operativos GNU/Linux.

ÍNDICE

DEDICATORIA.....	2
AGRADECIMIENTOS.....	3
FICHA RESUMEN.....	4
ÍNDICE	5
ÍNDICE DE TABLAS	8
ÍNDICE DE FIGURAS.....	9
CAPÍTULO I: INTRODUCCIÓN	12
1.1. Antecedentes	12
1.2. Definición del problema.....	13
1.3. Alcance	13
1.4. Innovación Tecnológica	13
1.5. Justificación.....	14
1.6. Objetivos	14
CAPÍTULO II: MARCO TEÓRICO O REFERENCIAL.....	15
2.1 Metodologías Ágiles	15
2.1.1 SCRUM.....	15
2.2 Tecnologías Web Servidor	16
2.2.1 NODEJS.....	16
2.2.2 ExpressJS.....	17
2.2.3 REST.....	17
2.2.4 WebSocket.....	19
2.3 Base de datos No Relacionales.....	19
2.3.1 MongoDB	19
2.4. Tecnologías Web Cliente.....	21
2.4.1. HTML5.....	21
2.4.2. JQuery.....	21
2.4.3. JQuery Mobile.....	22
2.4.4. PhoneGap	22
2.4.5. Bootstrap.....	23
2.4.6. AngularJS.....	23
CAPÍTULO III: ÁREA DE APLICACIÓN	26
3.1. Monitorización de Recursos y Servicios	26
3.2. Prevención de riesgos	26
CAPÍTULO IV: METODOLOGÍA Y DESARROLLO DEL PROYECTO	27
4.1. Iteración 1 Andromeda	27
4.1.1. Cuadro de Historias de usuario.....	27
4.1.2. Análisis del Sistema.....	30
4.1.3. Diseño de la Arquitectura del Sistema.....	35
4.1.4. Cuadro de Avance.....	37
4.1.5. Retrospectiva.....	39
4.2. Iteración 2 Cetus	40
4.2.1. Cuadro de Historias de usuario.....	40
4.2.2. Diseño	41
4.2.3. Implementación	43
4.2.4. Pruebas de Funcionalidad.....	48
4.2.5. Cuadro de Avance.....	50
4.2.6. Retrospectiva.....	51
4.3. Iteración 3 Columba.....	52

4.3.1.	Cuadro de Historias de Usuario	52
4.3.2.	Diseño	54
4.3.3.	Implementación	58
4.3.4.	Pruebas de Funcionalidad	61
4.3.5.	Pruebas de Integración	62
4.3.6.	Cuadro de Avance	63
4.3.7.	Retrospectiva	64
4.4.	Iteración 4 Cygnus	65
4.4.1.	Cuadro de Historias de Usuario	65
4.4.2.	Diseño	68
4.4.3.	Cuadro de Avance	79
4.4.4.	Retrospectiva	80
4.5.	Iteración 5 Crater	81
4.5.1.	Cuadro de Historias de Usuario	81
4.5.2.	Diseño	86
4.5.3.	Implementación	87
4.5.4.	Pruebas de Funcionalidad	101
4.5.5.	Pruebas de Integración	103
4.5.6.	Cuadro de Avance	105
4.5.7.	Retrospectiva	106
4.6.	Iteración 6 Eridanus	107
4.6.1.	Cuadro de Historias de Usuario	107
4.6.2.	Diseño	110
4.6.3.	Implementación	112
4.6.4.	Pruebas de Funcionalidad	118
4.6.5.	Pruebas de Integración	120
4.6.6.	Cuadro de Avance	121
4.6.7.	Retrospectiva	122
4.7.	Iteración 7 Centaurus	123
4.7.1.	Cuadro de Historias de Usuario	123
4.7.2.	Implementación	124
4.7.3.	Pruebas de funcionalidad	134
4.7.4.	Pruebas de Integración	136
4.7.5.	Cuadro de Avance	137
4.7.6.	Retrospectiva	138
CAPÍTULO V: CONCLUSIONES Y RECOMENDACIONES		139
REFERENCIAS BIBLIOGRÁFICAS		140
ANEXOS		142
Anexo 1: Manual de Instalación		143
	Aplicación Sensor	143
	MONIT	143
	Aplicación Web de servicios	144
	Aplicación Web cliente	145
	Aplicación móvil cliente	146
Anexo 2: Manual de Usuario		150
	Aplicación Sensor	150
	Aplicación Web de servicios	152
	Aplicación Web cliente	157
	Aplicación móvil cliente	171
Anexo 3: Manual Técnico		176
	Aplicación Sensor	176

Aplicación Web de servicios	177
Aplicación Web cliente	185
Aplicación móvil cliente	187

ÍNDICE DE TABLAS

Tabla 1. Cuadro de historias de usuario.....	30
Tabla 2. Cuadro de avance iteración 1.....	38
Tabla 3. Retrospectiva iteración 1.....	39
Tabla 4. Cuadro de historias de usuario, iteración 2.	41
Tabla 5. Pruebas de funcionalidad, iteración 2.	49
Tabla 6. Cuadro de avance, iteración 2.....	50
Tabla 7. Retrospectivas de iteración 2.	51
Tabla 8. Cuadro de historias de usuario, iteración 3.	54
Tabla 9. Definición de rutas para alertas, usuarios y sensores.	58
Tabla 10. Pruebas de funcionalidad realizadas en iteración 3.	61
Tabla 11. Pruebas de integración realizadas en iteración 3.	62
Tabla 12. Cuadro de avance, iteración 3.	63
Tabla 13. Retrospectiva iteración 3.	64
Tabla 14. Cuadro de historias de usuario, iteración 4.	68
Tabla 15. Cuadro de avance, iteración 4.	79
Tabla 16. Retrospectiva iteración 4.	80
Tabla 17. Cuadro de historias de usuario, iteración 5.	86
Tabla 18. Pruebas de funcionalidad, iteración 5.	103
Tabla 19. Pruebas de integración, iteración 5.	104
Tabla 20. Cuadro de avance, iteración 5.	105
Tabla 21. Retrospectiva, iteración 5.	106
Tabla 22. Cuadro de historias de usuario, iteración 6.	109
Tabla 23. Pruebas de funcionalidad, iteración 6.	119
Tabla 24. Pruebas de integración, iteración 6.	120
Tabla 25. Cuadro de avance, iteración 6.	121
Tabla 26. Retrospectiva iteración 6.	122
Tabla 27. Cuadro de historias de usuario, iteración 7.	124
Tabla 28. Pruebas de funcionalidad, iteración 7.	136
Tabla 29. Pruebas de integración, iteración 7.	137
Tabla 30. Cuadro de avance, iteración 7.	137
Tabla 31. Retrospectiva iteración 7.	138
Tabla 32. Descripción parámetros, aplicación sensor.	151
Tabla 33. Descripción rutas, aplicación servicio web.	155

ÍNDICE DE FIGURAS

Figura 1. Diagrama SCRUM.....	16
Figura 2. Thread modeling NodeJS	17
Figura 3. Diagrama REST API.....	19
Figura 4. Diagrama de comparación Base de datos relacionales vs no relacionales.	20
Figura 5. Diagrama Funcionamiento Phonegap.	23
Figura 6. Diagrama MVC AngularJS.	24
Figura 7. Diagrama de funcionamiento del sistema.	31
Figura 8. Diagrama de funcionamiento del servicio Web.	32
Figura 9. Diagrama de funcionamiento de la aplicación Web.	33
Figura 10. Diagrama de funcionamiento aplicación móvil.	34
Figura 11. Diagrama de funcionamiento aplicación Sensor.	35
Figura 12. Diagrama del diseño de la arquitectura del sistema.	36
Figura 13. Diagrama de Avance de historias de usuario, Iteración 1.	39
Figura 14. Diagrama Diseño Sensor.	42
Figura 15. Diagrama de Diseño Notifier.	42
Figura 16. Estructura Alerta.....	43
Figura 17. Diagrama de Secuencia Generar Alerta.	43
Figura 18. Diagrama del código para construir objetos JSON.	44
Figura 19. Código para reconocer parámetros.	45
Figura 20. Creación de la respuesta HTTP.....	46
Figura 21. Estructura archivo de configuración.....	46
Figura 22. Método Constructor Configurador.	47
Figura 23. Método para llenar configuraciones desde un archivo.	47
Figura 24. Diagrama de Avance de historias de usuario, Iteración 2.	50
Figura 25. Diagrama de secuencia, Alerta en servicio web.	54
Figura 26. Modelo de objetos JSON.	55
Figura 27. Modelo del objeto Settings.	57
Figura 28. Estructura proyecto aplicación web servicios.	58
Figura 29. Cabecera ruta nueva alerta.	59
Figura 30. Función crear alerta.....	60
Figura 31. Diagrama de avance Iteración 3.....	63
Figura 32. Diseño página inicio, aplicación web cliente.	69
Figura 33. Diseño página tablero, aplicación web cliente.	70
Figura 34. Diseño página alertas, aplicación web cliente.	71
Figura 35. Diseño cuadro dialogo editar alerta, aplicación web cliente.	72
Figura 36. Diseño listado de sensores, aplicación web cliente.	73
Figura 37. Diseño cuadro dialogo edición sensor, aplicación web cliente.	74
Figura 38. Diseño creación sensores, aplicación web cliente.	75
Figura 39. Diseño página listado usuarios, aplicación cliente web.....	76
Figura 40. Diseño creación de usuarios, aplicación cliente web.	77
Figura 41. Diseño página configuraciones, aplicación web cliente.	78
Figura 42. Diagrama de avance de historias de usuario, Iteración 4.....	79
Figura 43. Diagrama de secuencia autenticación de usuarios.	86
Figura 44. Diagrama de secuencia modificar alerta.	87
Figura 45. Diagrama de secuencia eliminar alerta.	87
Figura 46. Diagrama estructura proyecto aplicación web cliente.	88
Figura 47. Librerías utilizadas por la página de inicio.	88
Figura 48. Implementación página de inicio, aplicación web cliente.	89

Figura 49. Implementación cabecera, página de inicio.	90
Figura 50. Diagrama página de inicio, aplicación web cliente.....	90
Figura 51. Diagrama de la estructura de la aplicación servicios web.	91
Figura 52. Implementación autenticación usuario, servicios web.	92
Figura 53. Implementación cambio de estado usuario, aplicación servicios web.....	92
Figura 54. Página web de prueba, autenticación de usuarios.	93
Figura 55. Pantalla de inicio, con error en la autenticación de usuarios.	94
Figura 56. Implementación rutas, aplicación web cliente.	94
Figura 57. Implementación de la función para cargar datos a las gráficas.	96
Figura 58. Página tablero, aplicación web cliente.	96
Figura 59. Implementación de la función para cargar alertas.	97
Figura 60. Diagrama página listar alertas, aplicación cliente web.	97
Figura 61. Implementación servicio modificar alertas.	98
Figura 62. Diagrama cuadro de dialogo editar Alerta, aplicación web.	99
Figura 63. Implementación eliminar alertas, servicio web.	100
Figura 64. Diagrama eliminar alertas, aplicación web cliente.	100
Figura 65. Diagrama página configuración, aplicación cliente web.	101
Figura 66. Diagrama de avance de historias de usuario iteración 5.	105
Figura 67. Diagrama página autenticación usuarios, aplicación móvil.	110
Figura 68. Diagrama página listado de alertas, aplicación móvil.	111
Figura 69. Gráfica estructura proyecto aplicación móvil.	112
Figura 70. Diagrama estructura folder "www".....	113
Figura 71. Implementación página autenticación usuarios, aplicación móvil.	114
Figura 72. Diagrama página de autenticación, aplicación móvil.	115
Figura 73. Implementación autenticación usuarios, aplicación móvil.	116
Figura 74. Implementación página autenticación usuarios, aplicación móvil.	117
Figura 75. Implementación recepción alertas, aplicación móvil.	117
Figura 76. Diagrama página listado de alertas, aplicación móvil.	118
Figura 77. Diagrama de avance iteración 6.....	122
Figura 78. Implementación listar alertas, aplicación servicio web.	125
Figura 79. Implementación listar alertas, aplicación cliente web.	125
Figura 80. Implementación crear sensores, aplicación servicios web.	126
Figura 81. Implementación modificar sensores, aplicación servicios web.	127
Figura 82. Implementación eliminar sensores, aplicación servicios web.	128
Figura 83. Implementación listar usuarios, aplicación servicios web.	128
Figura 84. Implementación crear usuario, aplicación servicio web.	129
Figura 85. Implementación crear usuario, aplicación servicio web.	130
Figura 86. Implementación modificar usuario, aplicación servicio web.	131
Figura 87. Implementación eliminar usuario, aplicación servicio web.	131
Figura 88. Implementación guardar configuraciones, aplicación servicio web.	132
Figura 89. Implementación configuraciones, aplicación cliente web.	133
Figura 90. Implementación filtrar alertas, aplicación web cliente.	134
Figura 91. Cuadro de avance iteración 7.	138
Figura 92. Configurar URL servicio y URL de la aplicación cliente web.	145
Figura 93. URL aplicación cliente web.	146
Figura 94. Diagrama copiar instalador, aplicación móvil.	146
Figura 95Diagrama instalación aplicación móvil.	147
Figura 96. Diagrama instalación aplicación móvil.	147
Figura 97. Diagrama instalación completa, aplicación móvil.	148
Figura 98. Diagrama abrir aplicación móvil.	148
Figura 99. Diagrama pantalla de inicio, aplicación móvil.	149

Figura 100. Diagrama ventana de configuración.....	150
Figura 101. Diagrama obtener TOKEN.....	151
Figura 102. Página de autenticación, aplicación web.....	157
Figura 103. Página autenticación, cambiar idioma.....	158
Figura 104. Diagrama ir a página listar usuarios, aplicación web.....	158
Figura 105. Página listado de usuarios, aplicación web.....	159
Figura 106. Diagrama ir a página crear usuarios, aplicación web.....	159
Figura 107. Página crear usuarios, aplicación web.....	160
Figura 108. Página modificar usuario, aplicación web.....	161
Figura 109. Eliminar usuarios, aplicación web.....	161
Figura 110. Diagrama ir a página listar sensores, aplicación web.....	162
Figura 111. Página listar sensores, aplicación web.....	162
Figura 112. Diagrama crear sensores, aplicación web.....	163
Figura 113. Página modificar sensores, aplicación web.....	163
Figura 114. Diagrama eliminar sensores, aplicación web.....	164
Figura 115. Diagrama ir a página listar alertas, aplicación web.....	164
Figura 116. Página listar alertas, aplicación web.....	165
Figura 117. Diagrama filtrar alertas, aplicación web.....	165
Figura 118. Cuadro de diálogo, modificar alertas, aplicación web.....	166
Figura 119. Diagrama eliminar alertas, aplicación web.....	166
Figura 120. Diagrama ir a página configuraciones, aplicación web.....	167
Figura 121. Página configuraciones, habilitar gráficos, aplicación web.....	168
Figura 122. Página configuraciones, habilitar alertas, aplicación web.....	169
Figura 123. Página configuraciones, cambiar idioma.....	170
Figura 124. Diagrama cerrar sesión, aplicación web.....	170
Figura 125. Pantalla autenticación, aplicación móvil.....	171
Figura 126. Pantalla listar alertas, aplicación móvil.....	172
Figura 127. Pantalla filtrar alertas, aplicación móvil.....	173
Figura 128. Diagrama ir a pantalla configuraciones, aplicación móvil.....	174
Figura 129. Pantalla configuraciones, aplicación móvil.....	174
Figura 130. Diagrama cerrar aplicación, aplicación móvil.....	175

CAPÍTULO I: INTRODUCCIÓN

Los servidores son ordenadores que permiten a varios clientes compartir información e interactuar entre ellos, los cuales son componentes vitales de la infraestructura de TI de una empresa, por lo que constantemente se utilizan herramientas para monitorizar aplicaciones, servicios, registros de eventos, CPU, memoria, disco duro, etc.

La monitorización de servidores consiste en la vigilancia de todos los servicios activos que un ordenador ofrece. Los servicios pueden ser aplicaciones web, correo electrónico, mensajería instantánea, base de datos, etc.

La monitorización puede ser tanto interna como externa. En el caso que sea interna, la vigilancia se realiza desde la misma red donde está instalado el servidor. Cuando la monitorización es externa, se utiliza una plataforma de un proveedor de servicios que se encuentra fuera de la red local.

Debido a la importancia de los servidores en el mundo actual, estos ordenadores requieren ser monitorizados todo el tiempo, por lo cual el personal encargado de su funcionamiento debe contar con aplicaciones o herramientas que les permitan conocer el estado actual de dichos servidores, ya sea de manera local o remota, para prevenir posibles fallos y en caso de que se produzcan ser notificados inmediatamente para solucionarlos.

Hoy en día las aplicaciones web utilizan muchas notificaciones en tiempo real, desde chats como Facebook, notificaciones de nuevos elementos Twitter o actualizaciones en progreso, este tipo de tecnologías son conocidas como aplicaciones en tiempo real, debido a que los usuarios conectados reciben información actualizada constantemente, sin realizar solicitudes a cada instante.

La primera aproximación para alcanzar esta tecnología fue utilizar llamadas asíncronas en AJAX, solicitando datos de manera periódica, pero ese tipo de tecnologías no era fácil de escalar, debido a que una mala implementación provocaba que la aplicación sufriera denegaciones de servicio y en algunos casos este tipo de problemas causaba congestionamiento en la red debido a las constantes solicitudes de muchos usuarios.

“Las aplicaciones en tiempo real permiten realizar una comunicación bidireccional entre el cliente y el servidor, proporcionando respuestas inmediatas a cualquier interacción, como la conexión entre cliente y servidor es persistente, el intercambio de datos es muy rápido y por esta razón se la suele llamar Tiempo real” [12].

1.1. Antecedentes

Hoy en día Internet se ha transformado en una herramienta muy requerida en áreas de formación, desarrollo, comunicación, entretenimiento, etc. Produciendo al mismo tiempo el crecimiento de servicios para usuarios de todo tipo y con distintas necesidades.

Debido al crecimiento de las redes a nivel mundial, se vio la necesidad de compartir aplicaciones y recursos entre usuarios, por lo que se optó por centralizar servicios utilizando una arquitectura cliente servidor, donde el servidor comparte sus recursos entre los distintos clientes. Dado que el servidor está provisto de una capacidad limitada de recursos, estos deben ser monitorizados constantemente para evitar cualquier tipo de problema o perjuicio para sus clientes, es por esta razón que un administrador de servidores necesita de una aplicación que le permita conocer el estado actual de los servicios y recursos de sus servidores, para que pueda prevenir cualquier tipo de inconveniente o para iniciar la pronta solución a cualquier problema.

En la actualidad los sistemas operativos orientados a la administración de servidores cuentan con una gran variedad de herramientas para la monitorización de recursos y servicios, estas herramientas en su mayoría no cuentan con un sistema de notificación que permita a los administradores de servidores estar inmediatamente informados de los eventos producidos en sus servidores y sus aplicaciones, esta carencia puede llevar a producir fallas en los servicios y a su vez desconformidad en los usuarios, Actualmente existen aplicaciones de monitorización de recursos que proporcionan notificaciones, las cuales utilizan SMS o correo electrónico para llevarlas a cabo, desafortunadamente estas aplicaciones no permiten difundir notificaciones hacia cualquier dispositivo o tampoco permiten crear alertas personalizadas para aplicaciones específicas. Hoy en día existen muchas aplicaciones que permiten monitorizar recursos de servidores los cuales cuentan con módulos de notificación, que por lo general no fueron diseñados para ser extendidos o utilizados por otras aplicaciones externas a la misma.

1.2. Definición del problema

La excesiva cantidad de información dispersa que es generada por aplicaciones de monitorización de servidores, incide en la demora y pérdida de información al momento de generar notificaciones.

1.3. Alcance

El sistema a desarrollar estará enfocado al área de monitorización de servidores, por lo que será posible enviar notificaciones con información relacionada a recursos y servicios de un servidor tales como uso de memoria, discos duros, procesador, base de datos, aplicaciones Web, etc.

Las notificaciones serán enviadas a los usuarios utilizando WebSockets como medio de comunicación, por lo que el servidor y los dispositivos clientes utilizarán la librería SocketIO, para realizar esta comunicación.

1.4. Innovación Tecnológica

En el desarrollo del sistema se utilizarán un conjunto de tecnologías relacionadas a Java y JavaScript, así como también se manejarán tecnologías relacionadas a la programación Web.

La aplicación de notificaciones será implementada utilizando el lenguaje de programación JAVA, el cual es un lenguaje orientado a objetos cuenta con gran cantidad de documentación, ejemplos, y actualmente es muy estable.

Por la parte del servidor se utilizará el framework EXPRESSJS el cual abstraerá varias tareas que realiza NodeJS, y nos permitirá organizar el código de mejor manera, también nos brindará la posibilidad de trabajar con REST y Websockets con cierta libertad. La implementación de WebSockets será a través de SocketIO una librería que se encargará de controlar el intercambio de información desde los servidores hacia los dispositivos cliente.

Los dispositivos clientes, utilizarán PhoneGap como framework de desarrollo debido a la facilidad que representa utilizarlo. La aplicación encargada de realizar configuraciones en el servidor Web utilizará Bootstrap y AngularJS para adaptar en lo posible la interfaz de usuario en distintos navegadores y plataformas.

1.5. Justificación

El proyecto tiene la finalidad de permitir a sus usuarios recibir notificaciones con información relacionada al estado de los recursos y servicios de un servidor GNU/Linux, este intercambio de información se realizará utilizando como puente de comunicación las tecnologías REST y WebSockets las cuales facilitarán las tareas de recibir y enviar notificaciones en tiempo real.

1.6. Objetivos

Integrar información de aplicaciones de monitorización de servidores a través de un servicio de comunicación, para reducir demora y pérdida de información.

Como objetivos específicos el proyecto plantea:

- Recolectar y adaptar información del estado actual de recursos monitorizados, para enviar alertas que son originadas por eventos de aplicaciones instaladas en un servidor, hacia un servicio Web.
- Procesar y validar información de los recursos monitorizados para generar y enviar notificaciones en tiempo real hacia dispositivos clientes asociados a una cuenta de usuario.
- Mediante una aplicación web y con información de los recursos monitorizados, realizar operaciones de registro, modificación y eliminación de usuarios, dispositivos y servidores asociados a una cuenta de usuario.
- Haciendo uso de la interfaz de programación de un servicio web, recibir y procesar notificaciones en tiempo real, mediante una aplicación cliente para dispositivos móviles, con el fin de permitir a los usuarios conocer el estado actual de servicios y recursos en un servidor.

CAPÍTULO II: MARCO TEÓRICO O REFERENCIAL

2.1 Metodologías Ágiles

2.1.1 SCRUM

Es un modelo de referencia que define un conjunto de prácticas y roles, y que puede tomarse como punto de partida para definir el proceso de desarrollo que se ejecutará durante un proyecto. Los roles principales en SCRUM son:

- SCRUM Master mantiene los procesos y trabaja de forma similar al director de proyecto.
- Product Owner que representa a los interesados externos o internos del proyecto.
- Product Team Está conformado por todas las personas que intervienen directamente en el desarrollo del proyecto.

Durante cada iteración (sprint), un periodo entre dos y cuatro semanas (la magnitud es definida por el equipo), el equipo crea un incremento de software potencialmente entregable (utilizable). El conjunto de características que forma parte de cada sprint viene del Product Backlog también conocido como cuadro de avance, que es un conjunto de requisitos de alto nivel priorizados que definen el trabajo a realizar.

Los elementos del Product Backlog que forman parte de la iteración se determinan durante la reunión de planificación (Sprint Planning). Durante esta reunión, el Product Owner identifica los elementos del Product Backlog que quiere ver completados y los hace del conocimiento del equipo. Entonces, el equipo determina la cantidad de ese trabajo que puede comprometerse a completar durante el siguiente sprint. Durante el sprint, nadie puede cambiar el Sprint Backlog, lo que significa que los requisitos están congelados durante el sprint.

Cada sprint puede ser identificado con cualquier nombre que permita al equipo de producción recordar y tener una idea clara de las tareas llevadas a cabo en cada iteración.

SCRUM permite la creación de equipos auto organizados impulsando la co-localización de todos los miembros del equipo, y la comunicación verbal entre todos los miembros y disciplinas involucrados en el proyecto.

“Un principio clave de SCRUM es el reconocimiento de que durante un proyecto los clientes pueden cambiar de idea sobre lo que quieren y necesitan, y que los desafíos impredecibles no pueden ser fácilmente enfrentados de una forma predictiva y planificada. Por lo tanto, SCRUM adopta una aproximación pragmática, aceptando que el problema no puede ser completamente entendido o definido, y centrándose en maximizar la capacidad del equipo de entregar rápidamente y responder a requisitos emergentes.” [14]

Las características más marcadas que se logran notar en SCRUM serían: gestión regular de las expectativas del cliente, resultados anticipados, flexibilidad y adaptación, retorno de inversión, mitigación de riesgos, productividad y calidad, alineamiento entre cliente y equipo, por último equipo motivado. Cada uno de estos puntos mencionados hace que SCRUM sea utilizado de manera regular en un conjunto de buenas prácticas para el trabajo en equipo y de esa manera obtener resultados posibles.



Figura 1. Diagrama SCRUM.

Fuente: www.argin.com.ar/gestion-agil-con-scrum

Existen varias implementaciones de sistemas para gestionar el proceso de Scrum, que van desde notas amarillas "post-it" y pizarras hasta paquetes de software. Una de las mayores ventajas de Scrum es que es muy fácil de aprender, y requiere muy poco esfuerzo para comenzarse a utilizar.

2.2 Tecnologías Web Servidor

2.2.1 NODEJS

Node.js es un entorno JavaScript de lado del servidor que utiliza un modelo asíncrono y dirigido por eventos, usa el motor de JavaScript V8 de Google: una VM tremadamente rápida y de gran calidad, V8 es actualizado constantemente y es uno de los intérpretes más rápidos que puedan existir en la actualidad para cualquier lenguaje dinámico. Además las capacidades de Node para I/O (Entrada/Salida) son realmente ligeras y potentes, dando al desarrollador la posibilidad de utilizar al máximo la I/O del sistema. Node soporta protocolos TCP, DNS y HTTP. Una aplicación para Node se programa sobre un solo hilo. Si en la aplicación existe una operación bloqueante (I/O por ejemplo), Node creará entonces otro hilo en segundo plano, pero no lo hará sistemáticamente por cada conexión como haría Apache. En teoría Node puede mantener tantas conexiones como número máximo de archivos descriptores (sockets) soportados por el sistema. En un sistema UNIX este límite puede rondar por las 65.000 conexiones, un número muy alto. Sin embargo en la realidad la cifra depende de muchos factores, como la cantidad de información que esté la aplicación distribuyendo a los clientes. Una aplicación con actividad normal podría mantener 20-25.000 clientes a la vez sin haber apenas retardo en las respuestas.

La meta número uno declarada de Node es "proporcionar una manera fácil para construir programas de red escalables. En lenguajes como Java y PHP, cada conexión genera un nuevo hilo que potencialmente viene acompañado de 2 MB de memoria. En un sistema que tiene 8 GB de RAM, esto da un número máximo teórico de conexiones concurrentes de cerca de 4.000 usuarios.

A medida que crece su base de clientes, si usted desea que su aplicación soporte más usuarios, necesitará agregar más y más servidores. Desde luego, esto suma en cuanto a los costos de servidor del negocio, a los costos de tráfico, los costos laborales, y más.

Además de estos costos están los costos por los problemas técnicos potenciales, por ejemplo un usuario puede estar usando diferentes servidores para cada solicitud, así que cualquier recurso compartido debe almacenarse en todos los servidores. Por todas estas razones, el cuello de botella en toda la arquitectura de aplicación Web (incluyendo el rendimiento del tráfico, la velocidad de procesador y la velocidad de memoria) era el número máximo de conexiones concurrentes que podía manejar un servidor. [2]

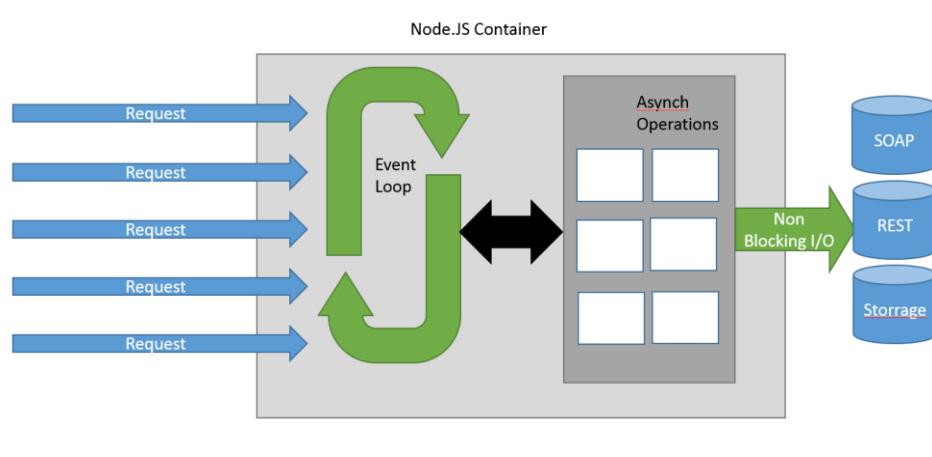


Figura 2. Thread modeling NodeJS.

Fuente: <http://www.ateam-oracle.com/scalability-and-productivity-for-your-mobile-applications-with-mobile-cloud-solutions/>

“Node resuelve este problema cambiando la forma en que se realiza una conexión con el servidor. En lugar de generar un nuevo hilo de OS para cada conexión (y de asignarle la memoria acompañante), cada conexión dispara una ejecución de evento dentro del proceso del motor de Node. Node también afirma que nunca se quedará en punto muerto, porque no se permiten bloqueos y porque no se bloquea directamente para llamados E/S. Node afirma que un servidor que lo ejecute puede soportar decenas de miles de conexiones concurrentes”. [3]

2.2.2 ExpressJS

Es un framework para el desarrollo de aplicaciones web que utiliza como base el lenguaje de programación NodeJS, este framework está orientado a ser minimalista y flexible. Está inspirado en el framework Sinatra, además de ser robusto, rápido, flexible y muy simple entre otras características. En la actualidad Express es utilizado por la mayor parte de desarrolladores Javascript debido a que cuenta con bastante documentación y es posible extender sus funcionalidades. Express está construido sobre Connect, otro framework desarrollado por Sencha Labs que permite el uso de middleware en Node.js. [1]

2.2.3 REST

REST (REpresentational State Transfer), es un tipo de arquitectura de desarrollo web que se apoya totalmente en el estándar HTTP.

REST nos permite crear servicios y aplicaciones que pueden ser usadas por cualquier dispositivo o cliente que entienda HTTP, por lo que es increíblemente más simple y convencional que otras alternativas que se han usado en los últimos diez años como SOAP y XML-RPC.

REST se definió en el 2000 por Roy Fielding, coautor principal también de la especificación HTTP. Podríamos considerar REST como un framework para construir aplicaciones web respetando HTTP.

Por lo tanto REST es el tipo de arquitectura más natural y estándar para crear APIs para servicios orientados a Internet.

Existen tres niveles de calidad a la hora de aplicar REST en el desarrollo de una aplicación web y más concretamente una API que se recogen en un modelo llamado Richardson Maturity Model en honor al tipo que lo estableció, Leonard Richardson padre de la arquitectura orientada a recursos. Estos niveles son:

1. Uso correcto de URIs
2. Uso correcto de HTTP.
3. Implementar Hypermedia.

Además de estas tres reglas, **nunca se debe guardar estado en el servidor**, toda la información que se requiere para mostrar la información que se solicita debe estar en la consulta por parte del cliente.

Al no guardar estado, REST nos da mucho juego, ya que podemos escalar mejor sin tener que preocuparnos de temas como el almacenamiento de variables de sesión e incluso, podemos jugar con distintas tecnologías para servir determinadas partes o recursos de una misma API. [5]

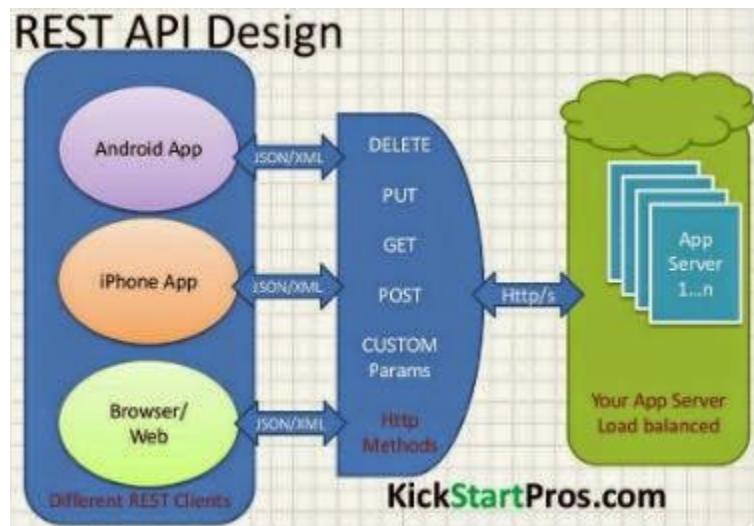


Figura 3. Diagrama REST API.

Fuente: www.kickstartpros.com

2.2.4 WebSocket

Es una tecnología que proporciona un canal de comunicación bidireccional y full-duplex sobre un único socket TCP. Está diseñado para ser implementado en navegadores y servidores web, pero puede utilizarse por cualquier aplicación cliente/servidor. La API de WebSocket está siendo normalizada por el W3C, y el protocolo WebSocket, a su vez, está siendo normalizado por el IETF. Como las conexiones TCP ordinarias sobre puertos diferentes al 80 son habitualmente bloqueadas por los administradores de redes, el uso de esta tecnología proporcionaría una solución a este tipo de limitaciones proveyendo una funcionalidad similar a la apertura de varias conexiones en distintos puertos, pero multiplexando diferentes servicios WebSocket sobre un único puerto TCP (a costa de una pequeña sobrecarga del protocolo).

La especificación del protocolo WebSocket define dos nuevos esquemas de URI, ws y wss para conexiones cifradas y no cifradas. Además del nombre del esquema, el resto de componentes del URI se definen con la sintaxis genérica de URI. [6]

2.3 Base de datos No Relacionales

2.3.1 MongoDB

MongoDB forma parte de la nueva familia de sistemas de base de datos NoSQL. Este manejador de base de datos se caracteriza por no guardar los datos en tablas como se hace en las bases de datos relacionalles. MongoDB guarda estructuras de datos en documentos tipo JSON con un esquema dinámico (MongoDB llama ese formato BSON), haciendo que la integración de los datos en ciertas aplicaciones sea más fácil y rápida. MongoDB tiene la capacidad de realizar consultas utilizando JavaScript, haciendo que estas sean enviadas directamente a la base de datos para ser ejecutadas.

En MongoDB los elementos de los datos son llamados documentos y se guardan en colecciones. Una colección puede tener un número indeterminado de documentos. Comparando con una base de datos relacional, se puede decir que las colecciones son como tablas y los documentos son registros en la tabla. La diferencia es que en una base de datos relacional cada registro en una tabla tiene la misma cantidad de campos, mientras que en MongoDB cada documento en una colección puede tener diferentes campos. En un documento, se pueden agregar, eliminar, modificar o renombrar nuevos campos en cualquier momento, ya que no hay un esquema predefinido.

La estructura de un documento es simple y compuesta por “key-value pairs” parecido a las matrices asociativas en un lenguaje de programación, esto es debido a que MongoDB sigue el formato de JSON. En MongoDB la clave es el nombre del campo y el valor es su contenido, los cuales se separan mediante el uso de “:”. MongoDB da respuesta a la necesidad de almacenamiento de todo tipo de datos: estructurados, semiestructurados y no estructurados.

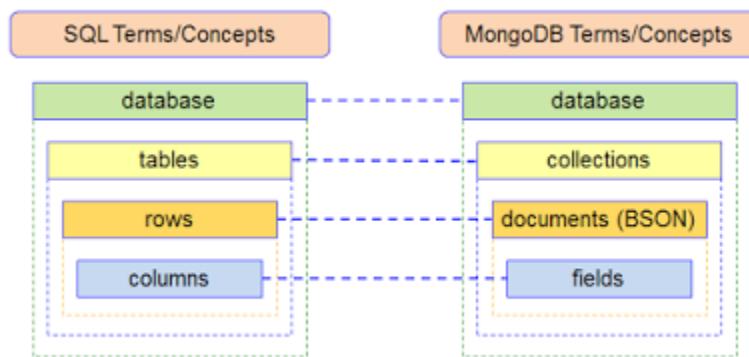


Figura 4. Diagrama de comparación Base de datos relacionales vs no relacionales.

Fuente: sql-vs-nosql.blogspot.com

- Tiene un gran rendimiento en cuanto a escalabilidad y procesado de la información.
- Puede procesar la gran cantidad de información que se genera hoy en día (millones de usuarios utilizando las mismas aplicaciones, redes sociales que crecen sin parar, internet de las cosas, computación en la nube, etc).
- Se adapta a las necesidades actuales de las aplicaciones (millones de usuarios, miles de peticiones por segundo).
- Permite a las empresas ser más ágiles y crecer más rápidamente, crear nuevos tipos de aplicaciones/productos, mejorar la experiencia del cliente y reducir el tiempo de manufacturado o desarrollo del producto, reduciendo así los costes.
- Está orientada a documentos. Lo que quiere decir que en un único documento es capaz de almacenar toda la información necesaria que define un producto, un cliente, etc, aceptando todo tipo de datos (incluidos arreglos y otros subdocumentos). Todo ello sin tener que seguir un esquema predefinido.

- Permite adaptar el esquema de la base de datos a las necesidades de la aplicación rápidamente, disminuyendo el tiempo y coste de la puesta en producción de la misma. Esto es así porque permite modificar el esquema desde el código de la aplicación, sin tener que realizar labores de administración de la base de datos como estaríamos obligados a hacer con las bases de datos relacionales.
- Da a los desarrolladores todas las funcionalidades que tienen las bases de datos relacionales. [7]

2.4. Tecnologías Web Cliente

2.4.1. HTML5

“Es la quinta revisión importante del lenguaje básico de la World Wide Web, HTML. HTML5 (HyperText Markup Language) especifica dos variantes de sintaxis para HTML, la variante conocida como HTML5 y una variante XHTML conocida como sintaxis XHTML5 que deberá ser servida como XML”.[17]

Esta es la primera vez que HTML y XHTML se han desarrollado en paralelo. HTML5 fue diseñado para ser utilizable por todos los desarrolladores de Open Web, se caracteriza por los siguientes aspectos.

- Semántica: lo que le permite describir con mayor precisión cuál es su contenido.
- Conectividad: lo que le permite comunicarse con el servidor de formas nuevas e innovadoras.
- Desconectado y almacenamiento: permite a páginas web almacenar datos, localmente, en el lado del cliente y operar fuera de línea de manera más eficiente.
- Multimedia: permite hacer vídeo y audio de ciudadanos de primera clase en la Web abierta.
- Gráficos y efectos 2D/3D: permite una gama mucho más amplia de opciones de presentación.
- Rendimiento e Integración: proporcionar una mayor optimización de la velocidad y un mejor uso del hardware del equipo.

2.4.2. JQuery

JQuery es una biblioteca de JavaScript, creada inicialmente por John Resig, que permite simplificar la manera de interactuar con los documentos HTML, manipular el árbol DOM, manejar eventos, desarrollar animaciones y agregar interacción a páginas web con técnicas como AJAX.

jQuery es software libre y de código abierto, posee un doble licenciamiento bajo la Licencia MIT y la Licencia Pública General de GNU v2, permitiendo su uso en proyectos libres y privados. jQuery, al igual que otras bibliotecas, ofrece una serie de funcionalidades basadas en JavaScript. jQuery consiste en un único fichero JavaScript que contiene las funcionalidades comunes de DOM, eventos, efectos y AJAX. La característica principal de la biblioteca es que permite cambiar el contenido de una página web sin necesidad de recargarla, mediante la manipulación del árbol DOM y peticiones AJAX. [9]

2.4.3. JQuery Mobile

JQuery Mobile framework toma el concepto “escribe menos, haz más” en lugar de escribir aplicaciones para cada dispositivo móvil, JQuery Mobile te permite diseñar una sola aplicación web que trabajará sobre todos los smartphones, tablets y plataformas desktop.

JQuery Mobile es un Framework optimizado para dispositivos táctiles que está siendo desarrollado actualmente por el equipo de proyectos de jQuery. El desarrollo se centra en la creación de un Framework compatible con la gran variedad de smartphones y tablets, algo necesario en el creciente, pero heterogéneo mercado de tablets y smartphones. El Framework de jQuery Mobile es compatible con otros frameworks móviles y plataformas como PhoneGap y Worklight entre otros. [10]

2.4.4. PhoneGap

PhoneGap es una solución de código abierto, ideal para los desarrolladores web que quieran construir aplicaciones móviles multiplataforma sin tener que aprender un nuevo idioma.

Simplemente usando HTML5, CSS3 y Javascript, PhoneGap permite entrar al universo móvil y desarrollar aplicaciones para iPhone, Android, BlackBerry, Windows, webOS y Symbian. PhoneGap es un framework para el desarrollo de aplicaciones móviles producido por Nitobi, y comprado posteriormente por Adobe Systems. Principalmente, PhoneGap permite a los programadores desarrollar aplicaciones para dispositivos móviles utilizando herramientas genéricas tales como JavaScript, HTML5 y CSS3. Las aplicaciones resultantes son híbridas, es decir que no son realmente aplicaciones nativas al dispositivo ya que el renderizado se realiza mediante vistas web y no con interfaces gráficas específicas de cada sistema, pero no se tratan tampoco de aplicaciones web teniendo en cuenta que son aplicaciones que son empaquetadas para poder ser desplegadas en el dispositivo incluso trabajando con el API del sistema nativo. En la tercera versión de PhoneGap se incorpora el uso de una interfaz de comandos a través de consola, una nueva arquitectura de complementos descentralizados y la posibilidad de utilizar un código web unificado para crear múltiples proyectos.

PhoneGap maneja API que permiten tener acceso a elementos como el acelerómetro, la cámara, los contactos en el dispositivo, la red, el almacenamiento, las notificaciones, etc. Estas API se conectan al sistema operativo usando el código nativo del sistema huésped a través de una Interfaz de funciones foráneas en Javascript.

PhoneGap permite el desarrollo ya sea ejecutando las aplicaciones en nuestro navegador web, sin tener que utilizar un simulador dedicado a esta tarea, y brinda la posibilidad de soportar funciones sobre frameworks como Sencha Touch o JQuery Mobile.

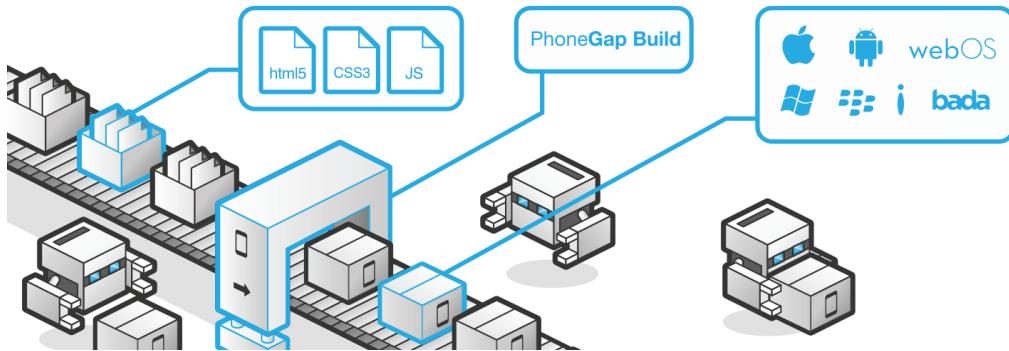


Figura 5. Diagrama Funcionamiento Phonegap.

Fuente: build.phonegap.com

PhoneGap es una distribución de Apache Cordova.5 en un principio fue llamada "PhoneGap", y posteriormente "Apache Callback". Ambos sistemas tienen funciones casi idénticas, la diferencia principal entre Apache Cordova y Phonegap es que el segundo tiene acceso a servicios de compilación en la nube proporcionados por Adobe Creative Cloud. [11]

2.4.5. Bootstrap

"Es un framework o conjunto de herramientas de software libre para diseño de sitios y aplicaciones web. Contiene plantillas de diseño con tipografía, formularios, botones, cuadros, menús de navegación y otros elementos de diseño basado en HTML y CSS, así como, extensiones de JavaScript opcionales adicionales". [12]

Para usar Bootstrap en una página HTML, el desarrollador solo debe descargar la hoja de estilo Bootstrap CSS y enlazarla en el archivo HTML. Bootstrap hace que el desarrollo de interfaces de usuario sea rápido. Bootstrap se creó para personas con todos los niveles de destrezas, dispositivos de todo tipo y proyectos de cualquier tamaño.

2.4.6. AngularJS

AngularJS es un framework MVC de JavaScript para el Desarrollo Web Front End que permite crear aplicaciones **SPA** (Single-Page Applications). Entra dentro de la familia de frameworks como BackboneJS o EmberJS. Su objetivo es aumentar las aplicaciones basadas en navegador con capacidad de Modelo Vista Controlador (MVC), en un esfuerzo para hacer que el desarrollo y las pruebas sean más fáciles. La biblioteca lee el HTML que contiene atributos de las etiquetas personalizadas adicionales, entonces obedece a las directivas de los atributos personalizados, y une las piezas de entrada o salida de la página a un modelo representado por las variables estándar de JavaScript. Los valores de las variables de JavaScript se pueden configurar manualmente, o recuperados de los recursos JSON estáticas o dinámicas.

AngularJS está construido en torno a la creencia de que la programación declarativa es la que debe utilizarse para generar interfaces de usuario y enlazar componentes de software, mientras que la programación imperativa es excelente para expresar la lógica de negocio. Este framework adapta y amplía el HTML tradicional para servir mejor contenido dinámico a través de un data-binding bidireccional que permite la sincronización automática de modelos y vistas. Como resultado, AngularJS pone menos énfasis en la manipulación del DOM y mejora la fase de pruebas y el rendimiento. [15]

Los objetivos de diseño:

- Disociar la manipulación del DOM de la lógica de la aplicación. Esto mejora la capacidad de prueba del código.
- Considerar a las pruebas de la aplicación como iguales en importancia a la escritura de la aplicación. La dificultad de las pruebas se ve reducida dramáticamente por la forma en que el código está estructurado.
- Disociar el lado del cliente de una aplicación del lado del servidor. Esto permite que el trabajo de desarrollo avance en paralelo, y permite la reutilización de ambos lados.
- Guiar a los desarrolladores a través de todo el camino de la construcción de una aplicación: desde el diseño de la interfaz de usuario, a través de la escritura de la lógica del negocio, hasta las pruebas.

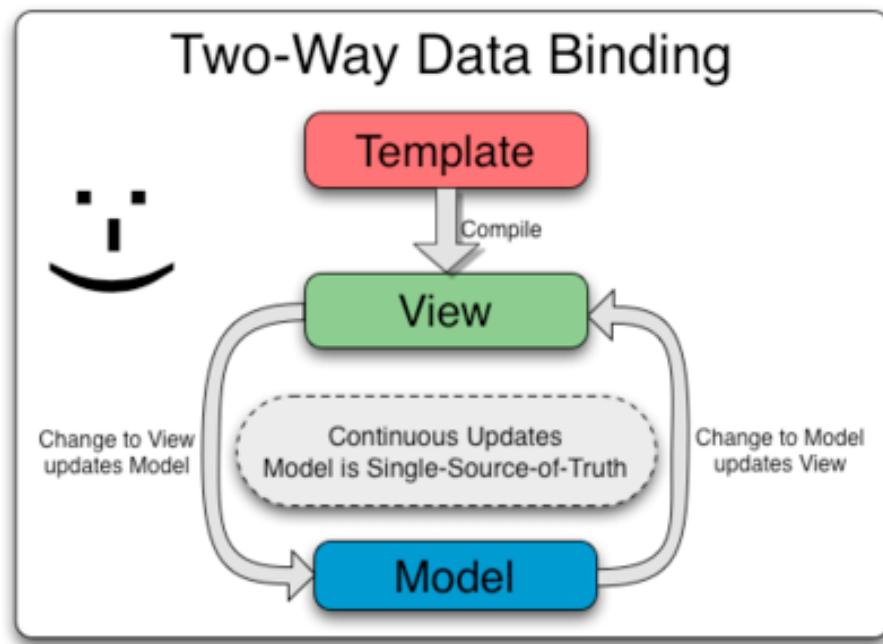


Figura 6. Diagrama MVC AngularJS.

Fuente: tombatossals.github.io/angularjs-tutorial/

Angular sigue el patrón MVC de ingeniería de software y alienta la articulación flexible entre la presentación, datos y componentes lógicos. Con el uso de la inyección de dependencias, Angular lleva servicios tradicionales del lado del servidor, tales como controladores dependientes de la vista, a las aplicaciones web del lado del cliente.

En consecuencia, gran parte de la carga en el backend se reduce, lo que lleva a las aplicaciones web mucho más ligeras. [13]

CAPÍTULO III: ÁREA DE APLICACIÓN

El sistema de notificaciones fue diseñado y desarrollado para enfocarse en servidores que tengan alguna variante del sistema operativo GNU/Linux instalado, en la actualidad una gran parte de los servidores se encuentran proveyendo servicios a través de este sistema operativo. Es por esta razón que los administradores de estos sistemas operativos invierten tiempo y recursos en la Monitorización de servidores y servicios.

3.1. Monitorización de Recursos y Servicios

En los sistemas de monitorización de recursos informáticos se pueden encontrar diferentes soluciones, tanto privadas como de distribución gratuita, que ofrecen gran variedad de posibilidades. Las necesidades de las empresas que las requieren son las que dictan la utilización de unas determinadas herramientas u otras, es muy habitual que las empresas dispongan de servicios externalizados de alojamiento de servidores y servicios (Hosting). Esto implica que los administradores informáticos responsables de gestionar los servicios no dispongan físicamente de los servidores a los que acudir presencialmente si surge cualquier tipo de incidencia. Además, ya sean servidores internos en la propia empresa o externos, muchas veces se hace difícil detectar un problema a nivel de servicio y es el cliente o el usuario final el encargado de alertar de la incidencia.

En la infraestructura informática de una empresa existen diferentes recursos que pueden llegar a monitorizarse con las herramientas adecuadas.

Hardware

- Estado y rendimiento de dispositivos de red o periféricos.
- Temperaturas de los componentes de un servidor.
- Consumo de memoria RAM, procesadores y espacios en discos.

Software

- Servicios instalados en el sistema operativo.
- Aplicaciones y base de datos.

En la actualidad el uso de herramientas de monitorización de recursos informáticos por parte de las empresas del sector de las tecnologías de la información toma mayor fuerza, esto se debe al gran crecimiento de este sector por lo que disponer de un sistema de monitorización centralizado que permita gestionar las incidencias en las infraestructuras informáticas implica un considerable ahorro de tiempo, personal y, por lo tanto, dinero. Además mejora la calidad del servicio que se ofrece y proporciona seguridad a los usuarios, clientes y responsables informáticos que utilizan o gestionan éstos sistemas.

3.2. Prevención de riesgos

Los sistemas de monitorización representan una ventaja a la hora de tomar decisiones relacionadas al comportamiento de los servidores dado que contar con información adecuada en el momento en que se genera dicha información permite a los usuarios analizar, identificar y prevenir posibles riesgos o problemas relacionados al funcionamiento del sistema operativo y los servicios que se ofrecen.

CAPÍTULO IV: METODOLOGÍA Y DESARROLLO DEL PROYECTO

4.1. Iteración 1 Andromeda

En la primera iteración se realizó un Análisis del conjunto de aplicaciones que el sistema representara en general, tomando en cuenta los distintos factores que podrían influir en su desarrollo.

4.1.1. Cuadro de Historias de usuario

A continuación se realizó el cuadro de historias de usuario desde el cual se empezará a seleccionar y trabajar en las distintas características del sistema de notificaciones. Note que para representar Criterio de Aceptación se utilizará la abreviatura CA.

Código	Título	Descripción	Tipo	Estimación	Prioridad
H1	Flujo de trabajo general del sistema de notificaciones	Para tener un concepto claro del funcionamiento del sistema es necesario realizar un análisis del mismo en el cual se deben determinar las herramientas a utilizar así como la forma en la que las distintas aplicaciones interactuarán. CA: El análisis debe estar respaldado por información valida incluyendo imágenes que ayuden a comprender lo explicado.	Diseño	13	1

H2	Flujo de trabajo de la aplicación Web que ofrecerá los servicios Web	<p>Se debe definir cómo será estructurada la aplicación así como también la funcionalidad que debería presentar, se deben tomar en cuenta aspectos relacionados al envío de mensajes en tiempo real, porque los servicios deben ser capaces de permitir notificar a cualquier usuario en cualquier momento</p> <p>CA: El análisis de estar respaldado por información valida incluyendo imágenes que ayuden a comprender lo explicado.</p>	Diseño	8	1
H3	Flujo de trabajo de la aplicación Web cliente	<p>La aplicación Web debe ser analizada como una herramienta administrativa que permita realizar configuraciones sobre los servicios implementados, esta aplicación deberá utilizar herramientas de desarrollo Web avanzadas que permitan abstraer el manejo de datos de la interfaz de usuario.</p> <p>CA: El análisis de estar respaldado por información valida incluyendo imágenes que ayuden a comprender lo explicado.</p>	Diseño	5	2

H4	Flujo de trabajo de la aplicación Móvil	<p>El enfoque de la aplicación móvil estará dirigido a la interfaz de usuario, por lo cual se deben tomar en cuenta aspectos técnicos de las características que pueda brindar el sistema operativo.</p> <p>CA: El análisis de estar respaldado por información válida incluyendo imágenes que ayuden a comprender lo explicado.</p>	Diseño	5	2
H5	Flujo de trabajo de la aplicación Sensor	<p>La aplicación Sensor es la encargada de enviar las alertas hacia los servicios Web, por lo tanto esta aplicación podrá ser utilizada en distintos sistemas operativos y en lo posible debe consumir pocos recursos, así como también debe ser posible realizar configuraciones en la misma que permitan enviar alertas hacia el servicio configurado.</p> <p>CA: El análisis de estar respaldado por información valida incluyendo imágenes que ayuden a comprender lo explicado.</p>	Diseño	5	1

H6	Arquitectura del sistema	Utilizando herramientas de modelación realizar la arquitectura del sistema, la cual permitirá entender de forma clara el funcionamiento del mismo. CA: El diseño de la arquitectura debe ser presentado a través de un gráfico del cual fácilmente se pueda diferenciar las diferentes aplicaciones y componentes del sistema de notificaciones.	Diseño	3	1
H7	Prototipo de la aplicación Sensor	Realizar el primer diseño, implementación y pruebas de funcionalidad de la aplicación sensor la cual deberá ser lanzada desde interfaz de usuario así como también desde línea de comandos. Dado que la aplicación será un prototipo no es necesario contar un diseño específico este será trabajado en otra historia de usuario.	Desarrollo	8	2

Tabla 1. Cuadro de historias de usuario.

Fuente: Elaboración propia.

4.1.2. Análisis del Sistema

La aplicación está enfocada en brindar información originada desde servidores hacia cualquier dispositivo conectado y autenticado, los servicios permitirán enviar y recibir mensajes, utilizando WebSockets por lo que las notificaciones serán enviadas instantáneamente.

Los servicios Web utilizarán REST para encaminar las solicitudes procedentes de los distintos servidores, también se almacenará toda la información en una base de datos de la cual los servicios realizarán consultas.



Figura 7. Diagrama de funcionamiento del sistema.

Fuente: Elaboración propia

En la Figura 7 podemos observar que distintos servidores envían alertas al servicio web con información referente a eventos que ocurren en dichos servidores, los servicios verificarán si la información proporcionada es válida entonces procederá a almacenar esta información y posteriormente enviará notificaciones a los dispositivos conectados.

4.1.2.1. Aplicación Servicios REST

La aplicación REST utilizará como gestor de base de datos MongoDB para almacenar toda la información recibida desde los distintos sensores instalados en los servidores a monitorizar. MongoDB es un manejador de base de datos NoSQL que nos permitirá manejar la información como si se tratase de un objeto JSON, este manejador de base de datos también tiene la característica de consumir pocos recursos respecto al procesador y memoria lo cual significa un gran beneficio para aplicaciones en tiempo real, además de proporcionar rápidamente la información solicitada a través de consultas.

La aplicación de servicios estará constituida de una capa de seguridad que se encargará de la autenticación y aspectos relacionados a la seguridad de la aplicación, también contará con una capa de servicios la que será responsable de encaminar cada una de las solicitudes así como también proporcionar las distintas respuestas a los clientes, finalmente una capa de datos estará encargada de leer y escribir información en la base de datos.

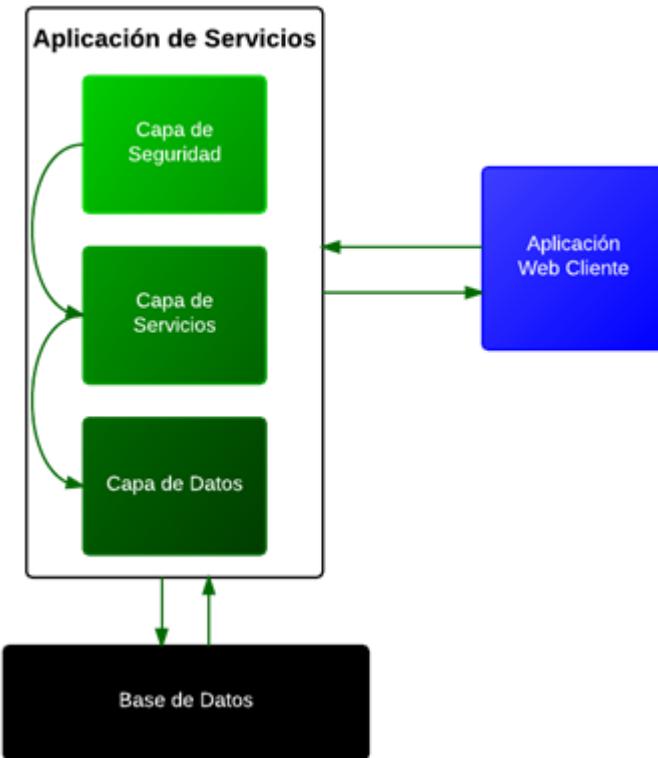


Figura 8. Diagrama de funcionamiento del servicio Web.

Fuente: Elaboración propia

Podemos observar las distintas capas de la aplicación que manejarán los servicios Web. En la capa de seguridad se utilizarán sesiones para poder autenticar a los distintos usuarios los cuales contarán un nombre de usuario y contraseña la cual será encriptada en base de datos.

La capa de servicios es la encargada de procesar la información y encaminarla hacia los dispositivos conectados, en esta capa se definirán “rutas”, las que representarán los puntos de acceso al servicio Web. La configuración e implementación de los WebSockets será realizada en esta capa donde podrá fácilmente acoplar esta característica a los servicios Web.

La capa de datos es la encargada de manejar la información obtenida de los servidores así como también información de los usuarios, en la capa de datos configurará la conexión a la base de datos así como también los puertos por los cuales esta base de datos estará corriendo.

La aplicación de servicios será implementada utilizando el framework ExpressJS el cual utiliza NodeJS como lenguaje de programación, este lenguaje nos permitirá fácilmente implementar los distintos servicios REST así como también la integración con WebSockets y MongoDB, el framework cuenta con mucha información disponible y la curva de aprendizaje es reducida.

4.1.2.2. Aplicación Web (Cliente)

La aplicación web cliente, permitirá realizar cambios en la configuración de notificaciones así como también permitirá obtener información relacionada a usuarios del sistema y de los distintos dispositivos registrados.

La aplicación web utilizará el framework javascript AngularJS el cual maneja el patrón de diseño modelo, vista, controlador, de esta forma existirá una clara separación de capas en el lado del cliente lo que permitirá una mejor implementación del mismo.

Utilizando AngulaJS y Bootstrap para manejar de manera rápida y eficiente las interfaces usuario la aplicación contará con interfaces dinámicas y amigables.

La aplicación Web cliente obtendrá toda la información desde los servicios web, por lo que es necesario realizar las configuraciones necesarias para que apunten a los servicios correctos, también es necesario realizar configuraciones en los permisos de acceso en los servicios web dado que la aplicación Web no necesariamente estará alojada en el mismo dominio.

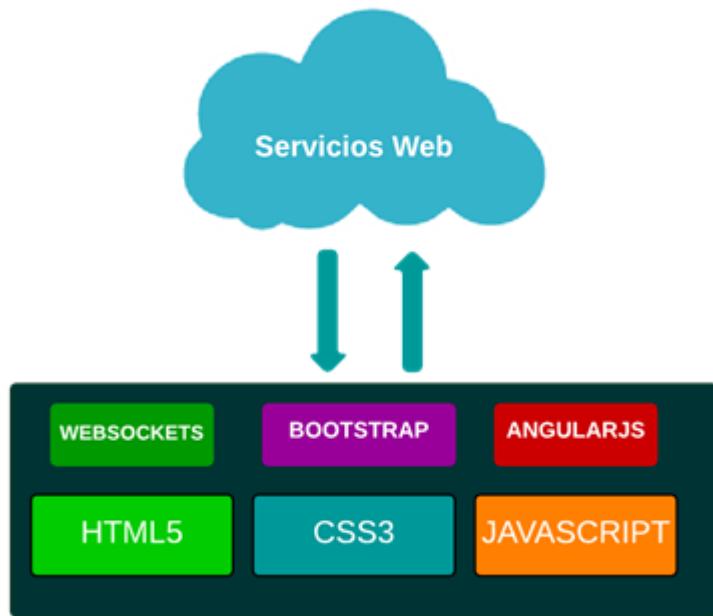


Figura 9. Diagrama de funcionamiento de la aplicación Web.

Fuente: Elaboración propia

La aplicación Web cliente debe permitir listar alertas, usuarios y sensores, también debe poder tener interfaces de usuario para las operaciones básicas como crear, actualizar y eliminar. El objetivo de esta aplicación es poder realizar configuraciones de los servicios web, por lo tanto la página de configuraciones debe permitir realizar este cometido basando sus criterios en las características de los servicios web. Finalmente para los usuarios el hecho de contar con gráficas que permitan entender fácilmente el estado de sus servidores es importante por lo que será necesario implementar graficas usando HighChart.

4.1.2.3. Aplicación para dispositivos Móviles

La aplicación para dispositivos móviles estará encargada de recibir notificaciones enviadas desde el servicio Web implementado, la aplicación permitirá a los distintos usuarios conectados al servidor tener una clara separación de los equipos asociados a sus cuentas, con lo cual el usuario podrá fácilmente identificar el estado de sus equipos.

La aplicación móvil será implementada utilizando el Framework Phonegap, el cual permitirá desplegar la aplicación en distintos dispositivos móviles y distintos sistemas operativos. En este proyecto únicamente se utilizará Android como sistema operativo para dispositivos móviles. Debido a que la aplicación debe realizar notificaciones en segundo plano ya sea del tipo sonoro o visual desde la barra de tareas del dispositivo Android.



Figura 10. Diagrama de funcionamiento aplicación móvil.

Fuente: Elaboración propia

Las características principales de la aplicación móvil estarán enfocadas en proveer una interfaz útil y fácil de manejar, la cual manejará y obtendrá información desde los servicios REST, cada pantalla contará con enlaces a diferentes pantallas con el fin de permitir a los usuarios navegar dentro de la aplicación rápidamente.

4.1.2.4. Aplicación Sensor

La aplicación Sensor es la encargada de enviar alertas desde los distintos equipos asociados hacia el servicio Web, para que este pueda realizar la propagación correspondiente de las notificaciones a los distintos dispositivos.

La aplicación Sensor permitirá configurar la dirección del servicio Web, permitiendo a los usuarios tener mayor control de las alertas que se enviarán. Así también la aplicación podrá ser utilizada desde línea de comandos como desde una interfaz gráfica.

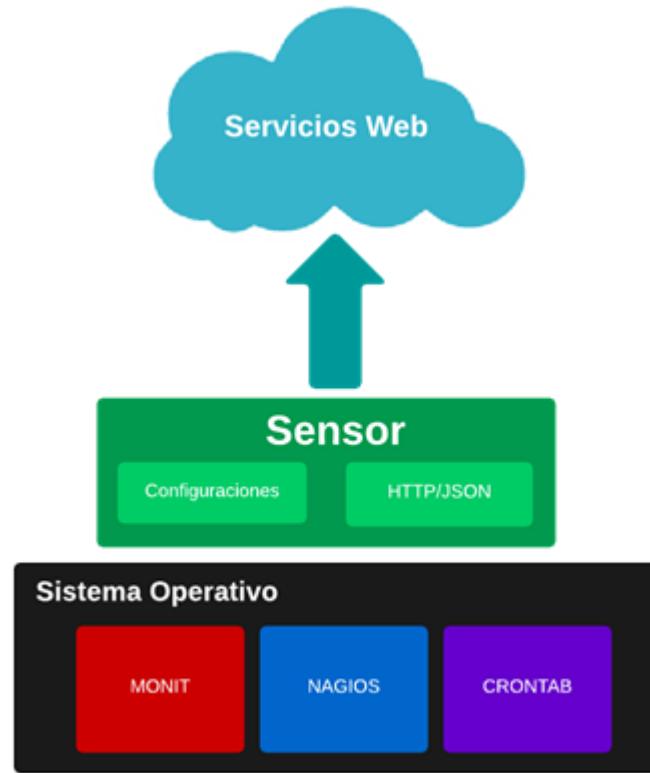


Figura 11. Diagrama de funcionamiento aplicación Sensor.

Fuente: Elaboración propia

El propósito de la aplicación Sensor es utilizar una pequeña aplicación que se encargue de enviar alertas en forma JSON a través del protocolo HTTP, cada una de las aplicaciones de monitorización del sistema podrá invocar a la aplicación Sensor pasando como parámetros información relacionada a un respectivo evento ocurrido en el sistema operativo.

4.1.3. Diseño de la Arquitectura del Sistema

La arquitectura del sistema se encuentra representada en la siguiente figura6 donde se especifican las tecnologías a utilizar así como la composición de aplicaciones en el sistema de notificaciones.

En la arquitectura se pretende remarcar la diferencia entre las diferentes aplicaciones así como también la relación de interacción entre las mismas.

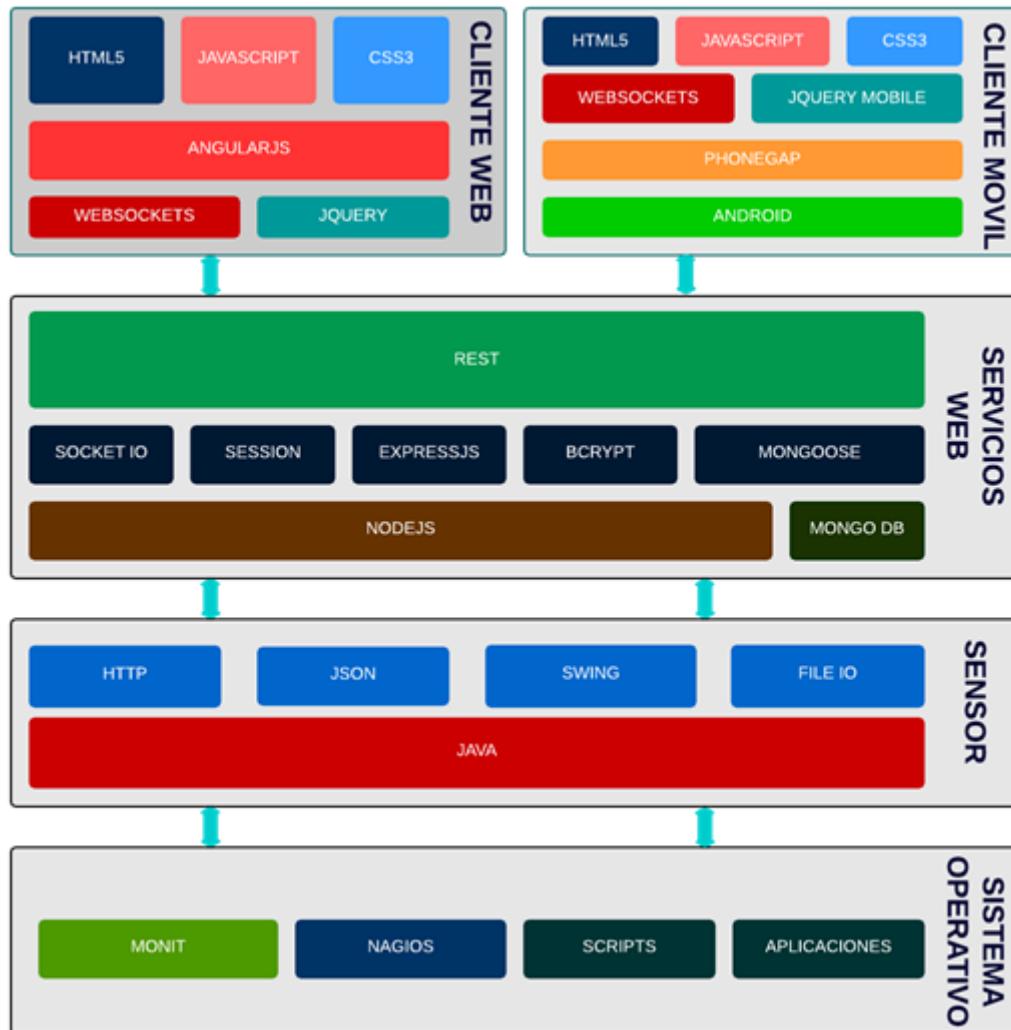


Figura 12. Diagrama del diseño de la arquitectura del sistema.

Fuente: Elaboración propia.

El sistema de notificaciones en tiempo real está enfocado en recibir alertas de servidores GNU/Linux los cuales pueden tener instaladas aplicaciones para la monitorización de sus recursos, cada una de estas aplicaciones de monitorización utilizan señales y eventos para notificar al sistema operativo sobre alguna incidencia, es por esta razón que es necesario instalar en cada uno de estos servidores una aplicación Sensor la cual será utilizada por cada una de las aplicaciones de monitorización, para emitir sus alertas.

La aplicación sensor puede ser utilizada desde cualquier otra aplicación simplemente se deben pasar como atributos parámetros como el mensaje y el tipo de alerta. La aplicación Sensor será desarrollada utilizando el lenguaje de programación JAVA el cual tiene cierta independencia del sistema operativo en el que funciona, esta aplicación utilizará librerías para el manejo de paquetes a través del protocolo HTTP así como también librerías para el manejo de información en archivos y mensajes en formato JSON.

La aplicación de servicios Web representa el enlace entre los dispositivos cliente y los servidores monitorizados debido a que es a través de esta aplicación que se enviarán las notificaciones. Esta aplicación Web consta de un conjunto de librerías que funcionan bajo NodeJS que tienen como finalidad proveer seguridad, conexión a base de datos, manejo de sesiones y comunicación.

Haciendo uso de los principios de REST se realizarán llamadas a los servicios provistos en la aplicación, de esta forma se obtendrá una mejor implementación y mantenibilidad de la aplicación.

Finalmente las aplicaciones Web cliente están enfocadas en ofrecer a los usuarios la mejor experiencia visual posible, haciendo uso de librerías y Frameworks de última generación con un enfoque adaptativo a diferentes dispositivos.

Las aplicaciones Web Cliente reciben notificaciones del Proveedor de servicios, el cual utiliza WebSockets para hacerlo. Toda la información mostrada en estas aplicaciones es obtenida de los servicios REST del Proveedor de Servicios.

4.1.4. Cuadro de Avance

Para esta iteración se trabajó en seis historias de usuarios en un periodo de 15 días. A continuación la lista de historias trabajadas con el esfuerzo y duración de cada historia de usuario.

Código	Estimación	Duración	Estado
H1 (Realizar el flujo de trabajo general del sistema de notificaciones)	13	5	Hecho
H2 (Realizar el flujo de trabajo de la aplicación Web que ofrecerá los servicios Web)	8	3	Hecho
H3 (Realizar el flujo de trabajo de la aplicación Web cliente)	5	2	Hecho
H4 (Realizar el flujo de trabajo de la aplicación Móvil)	5	1	Hecho
H5 (Realizar el análisis de la aplicación Sensor)	5	2	Hecho
H6 (Realizar el diseño de la arquitectura del sistema)	3	1	Hecho
H7 (Implementación del prototipo de la aplicación Sensor)	8	1	En Progreso

Tabla 2. Cuadro de avance iteración 1.

Fuente: elaboración propia.

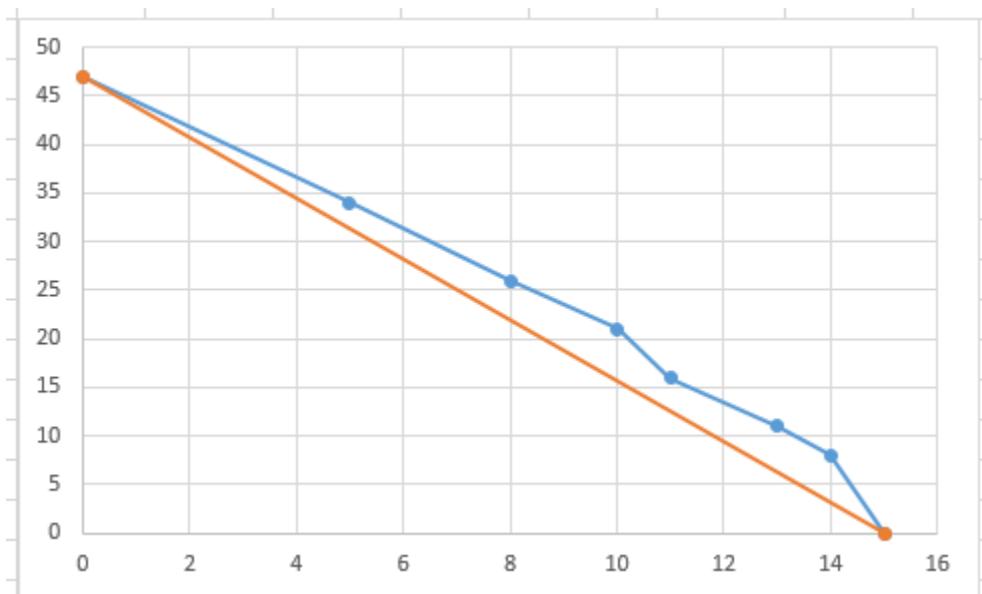


Figura 13. Diagrama de Avance de historias de usuario, Iteración 1.

Fuente: Elaboración Propia

4.1.5. Retrospectiva

Bueno	Malo	Mejorar
<ul style="list-style-type: none"> - La mayor parte de las historias de usuario fueron completadas. - Se obtuvo un entendimiento claro de cuál es el rol de cada aplicación dentro del sistema. 	<ul style="list-style-type: none"> - Por problemas de herramientas no se pudo generar archivo PNG del Diseño de la arquitectura. - En general no fue buena la estimación debido a que no se completó en su totalidad las historias planificadas. 	<ul style="list-style-type: none"> - En siguientes iteraciones se considerará una mejor estimación de las historias de usuario para evitar dejar tareas pendientes.

Tabla 3. Retrospectiva iteración 1.

Fuente: Elaboración propia.

4.2. Iteración 2 Cetus

A partir de esta iteración se definieron los diseños de los prototipos funcionales del sistema los cuales nos permitieron verificar el correcto funcionamiento de las tecnologías escogidas para el desarrollo de cada una de las aplicaciones que conforman el sistema de notificaciones.

4.2.1. Cuadro de Historias de usuario

Código	Título	Descripción	Tipo	Estimación	Prioridad
H7	Prototipo inicial de la aplicación Sensor	<p>Para iniciar el desarrollo de la aplicación sensor es necesario contar el diseño de la aplicación Sensor se debe tomar en cuenta las tecnologías que se utilizarán en su desarrollo así como también como interactuarán entre ellas.</p> <p>CA: Se debe contar con una gráfica del diseño de la aplicación donde se pueda diferenciar las distintas tecnologías a utilizar en su implementación.</p>	Diseño	5	1
H8	Módulo de configuración	<p>El módulo de configuración es el encargado de manejar las distintas opciones que ofrece la aplicación sensor, para poder conectarse a diferentes servidores.</p> <p>CA: Este módulo debe permitir a los usuarios, cambiar la ruta o URL a la cual se conecta el sensor para realizar el envío respectivo de las alertas, así como también el puerto de comunicación, el nombre de usuario y la contraseña de autenticación.</p> <p>La configuración de la aplicación sensor debe ser almacenada en un archivo de texto plano donde la contraseña debe ser cifrada en este archivo.</p>	Desarrollo	13	3

H9	Módulo de comunicación	<p>El módulo de comunicación será el encargado de enviar los paquetes de información utilizando el protocolo HTTP.</p> <p>CA: Este módulo debe proporcionar al desarrollador la capacidad de enviar información a través del protocolo HTTP en un formato JSON para facilitar la interpretación de esta información en el lado del servidor.</p>	Desarrollo	8	1
H10	Pruebas de integridad de la aplicación sensor	<p>La aplicación debe ser ejecutada desde línea de comandos permitiendo solo los parámetros definidos.</p> <p>CA: Desde línea de comandos la aplicación permitirá los parámetros:</p> <ul style="list-style-type: none"> • Tipo de alerta: info, warning, danger • Título de alerta • Descripción alerta 	Pruebas	5	3

Tabla 4. Cuadro de historias de usuario, iteración 2.

Fuente: Elaboración propia.

4.2.2. Diseño

La aplicación Sensor será la encargada de enviar las alertas correspondientes desde los servidores hacia el servicio Web por lo tanto es necesario tomar en cuenta los siguientes aspectos para la implementación de esta aplicación.

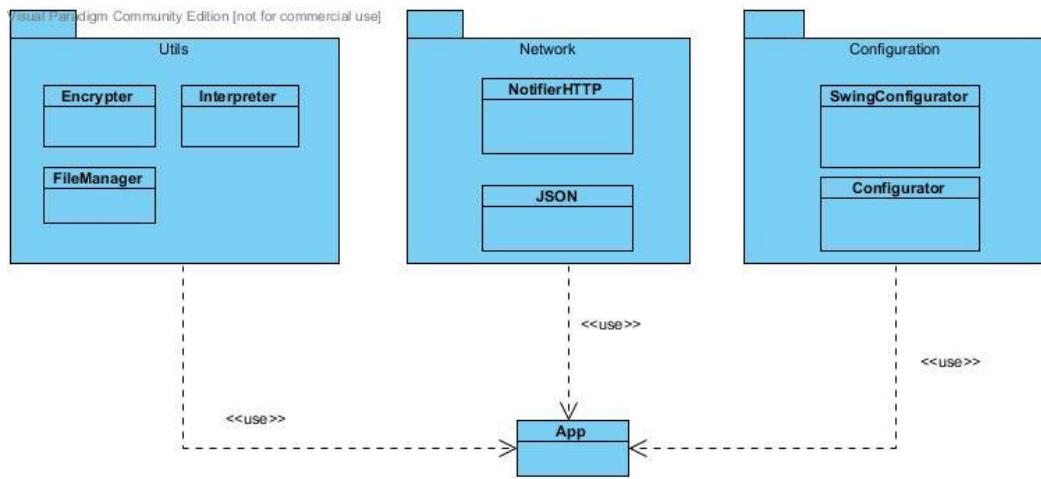


Figura 14. Diagrama Diseño Sensor.

Fuente: Elaboración propia

En la gráfica anterior podemos observar que la clase principal “App” hace uso de los paquetes “utils”, “network” y “configuration”, los cuales a su vez contienen clases para realizar la encriptación, el manejo de archivos, la comunicación con el servicio Web así como también la interfaz gráfica para lanzar las alertas.

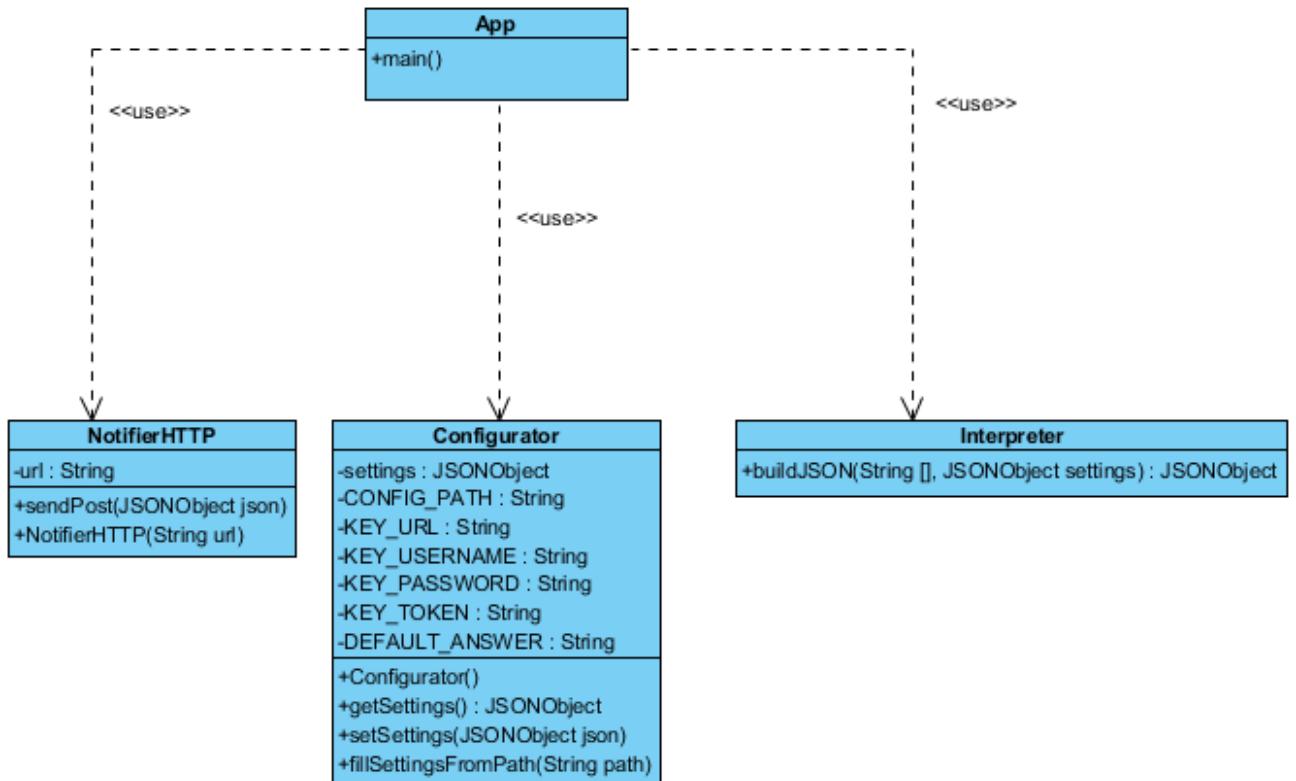


Figura 15. Diagrama de Diseño Notifier.

Fuente: Elaboración propia.

En el diagrama anterior podemos observar a detalle como las distintas clases interactúan entre sí para enviar alertas, basando su configuración en información obtenida desde archivos. La clase “App” hace uso de las clases “NotifierHTTP”, “Configurador” y “Interpretador” estas clases permiten abstraer las tareas de lectura de configuraciones, envío de paquetes HTTP y construcción de los objetos JSON. Es necesario tomar en cuenta que cuando la aplicación “Notifier” envíe alertas están deben contener la información adecuada por lo que es necesario definir la estructura del objeto JSON, en la siguiente imagen podemos apreciar el estado actual de la estructura “Alert”.

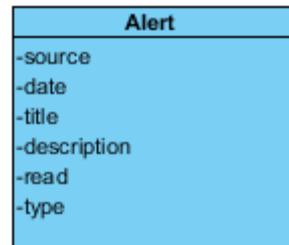


Figura 16. Estructura Alerta.

Fuente: Elaboración propia

El módulo de conexión a servicios Web, está encargado de obtener un interpretación de los parámetros ingresados y una vez que estos fueron interpretados el modulo debe generar una respuesta utilizando un objeto JSON.

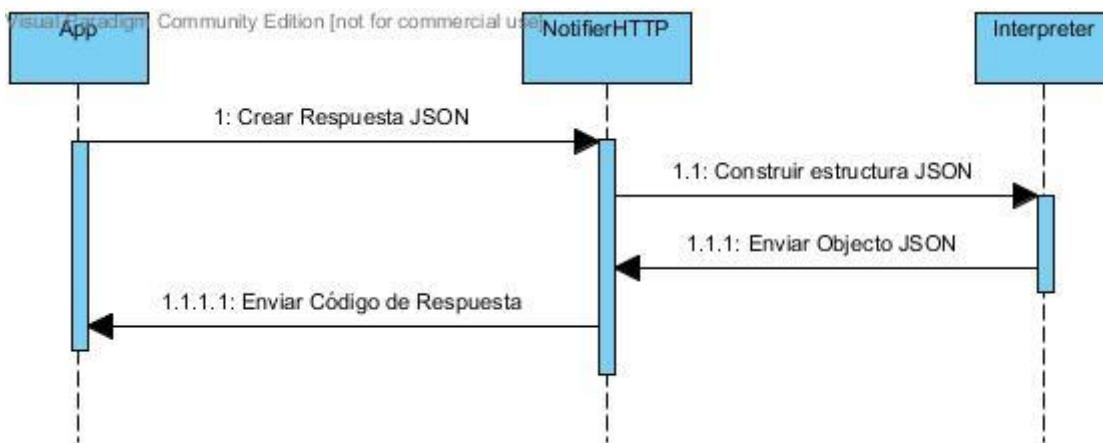


Figura 17. Diagrama de Secuencia Generar Alerta.

Fuente: Elaboración Propia

4.2.3. Implementación

En la etapa de implementación se desarrollaron los módulos de previstos en la etapa de diseño, de los cuales el código más relevante fueron *NotifierHTTP* y *Configurador* estas dos clases permiten a la aplicación Sensor configurar la conexión y también enviar información hacia un servicio Web.

- **NotifierHTTP (Modulo de conexión a servicios Web)**

El código utilizado para realizar la estructura del objeto JSON es mostrado a continuación donde podemos denotar que un objeto Alerta mínimamente requiere tres atributos los cuales son: la fecha en la que la Alerta es generada, el estado de la alerta, que inicialmente es no leída y la dirección IP del equipo emisor de la alerta.

```
public JSONObject buildJSON(String[] arguments, JSONObject settings) {  
  
    try {  
        DateFormat dateFormat = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");  
        Date currentDate = new Date();  
        InetAddress address = InetAddress.getLocalHost();  
  
        settings.put("date", dateFormat.format(currentDate));  
        settings.put("read", "false");  
        settings.put("source", address.getHostAddress());  
    }  
}
```

Definimos información básica de la alerta

Figura 18. Diagrama del código para construir objetos JSON.

Fuente: Elaboración Propia

Una vez que los atributos iniciales fueron configurados, procedemos a completar la estructura del objeto Alerta, para proceder con la creación del objeto necesitamos obtener el valor de los parámetros utilizados en la ejecución del programa. Cada alerta debe contar con un título, una descripción y un tipo de alerta, el título representa el sujeto del mensaje, el cual será visualizado por los clientes directamente. La descripción representa un mensaje detallado del evento generado en el servidor. Finalmente el tipo de alerta representa la severidad de la alerta la cual se utilizará para clasificar las alertas en los dispositivos clientes.

El código utilizado a continuación nos permite fácilmente obtener y establecer el valor de cada uno de los parámetros utilizados, con estos parámetros el objeto JSON es fácilmente creado y los valores de sus atributos son establecidos.

```

switch (arguments[i]) {
    case "-p":
        settings.put("type", arguments[i+1]);
        break;
    case "-d":
        settings.put("description", arguments[i+1]);
        break;
    case "-t":
        settings.put("title", arguments[i+1]);
        break;
    case "-h":
        System.out.println("- Notifier v.0.2 -");
        System.out.println("-h\t\t Display this message");
        System.out.println("-p\t\t Alert Type, just choose one of this [ alert, info, warning ]");
        System.out.println("-d\t\t Alert Description, if the description has more than one word please to use \"\" ");
        System.out.println("-t\t\t Alert Title, if the description has more than one word please to use \"\" around");
        System.out.println("Please to use: java -jar notifier.jar -p < danger | info | warning > -t \"title Alert\"");
        System.out.println("Settings content:");
        System.out.println(settings.toString());
        break;
    default:
        settings.put("title", "Invalid Parameter");
        settings.put("description", "The Parameter: " +
            arguments[i]+" is not recognized");
        break;
}

```

Los parámetros principales. Cada alerta
debe contar con:
Tipo, descripción y título

Figura 19. Código para reconocer parámetros.

Fuente: Elaboración propia.

Cuando el objeto JSON está construido procedemos a establecer conexión con el servicio encargado de recibir alertas. La dirección del servidor inicialmente fue colocado como parte del código, ya con la implementación del módulo de configuración fue posible mover dichas configuraciones a los archivos de configuración correspondientes. El código que observamos a continuación nos permitió realizar la conexión, definir el tipo de contenido que se manejará así como también definir el método HTTP que se utilizará para enviar las alertas.

```

public void sendPOST(JSONObject json) throws IOException {
    URL object = new URL(this.url);
    HttpURLConnection connection = (HttpURLConnection) object.openConnection();
    connection.setDoOutput(true);
    connection.setDoInput(true);
    connection.setRequestProperty("Content-Type", "application/json; charset=utf8");
    connection.setRequestProperty("Accept", "application/json");
    connection.setRequestMethod("POST");

    // Send json
    OutputStreamWriter wr = new OutputStreamWriter(connection.getOutputStream());
    wr.write(json.toString());
    wr.flush();

    // Display what returns the POST request
    StringBuilder sb = new StringBuilder();
    int HttpResult = connection.getResponseCode();

    if (HttpResult == HttpURLConnection.HTTP_OK) {
        String line;
        try (BufferedReader br = new BufferedReader(new InputStreamReader(connection.getInputStream(), "utf-8"))) {
            while ((line = br.readLine()) != null) {
                sb.append(line);
                sb.append("\n");
            }
        }
    }
}

```

Definimos el tipo de respuesta y enviamos el mensaje

Verificamos si la respuesta fue correcta

Figura 20. Creación de la respuesta HTTP.

Fuente: Elaboración propia.

- **Configurator (Modulo de configuración de la aplicación)**

El módulo de configuración es el encargado de recolectar información de los archivos de configuración para poder utilizar dicha información de manera sencilla y eficiente. El módulo inicialmente está desarrollado para trabajar con archivos de texto. La estructura del archivo de configuración es simple y los valores insertados en este archivo deben estar separados por un salto de línea, los valores permitidos son: URL, nombre de usuario, contraseña, Token o llave de acceso.

1	http://localhost:3000/api/alert	URL
2	gabitosoft@gmail.com	Nombre Usuario
3	[C@1e77dda	Contraseña
4	1234TOKEN4321	Token
5		

Figura 21. Estructura archivo de configuración.

Fuente: Elaboración propia.

El módulo “Configurator” permite cargar las configuraciones desde un archivo de texto situado en el mismo directorio del ejecutable de la aplicación. Los atributos utilizados por la clase “Configurator” son llenados en el método constructor. Inicialmente el objeto JSON es vaciado para evitar utilizar información errónea, si el archivo de configuración existe entonces las llaves KEY_URL, KEY_USERNAME, KEY_PASSWORD y KEY_TOKEN serán obtenidas en caso de que el archivo de configuración no pueda ser accedido, las llevas obtendrán su valor de la constante DEFAULT_ANSWER.

```

public Configurator() {
    this.settings = new JSONObject();
    settings.clear();

    try {
        FileManager fileManager = new FileManager();
        if (fileManager.existFile(CONFIG_PATH)) {

            ArrayList<String> content = fileManager.readFile(CONFIG_PATH);
            settings.put(KEY_URL, content.get(0));
            settings.put(KEY_USERNAME, content.get(1));
            settings.put(KEY_PASSWORD, content.get(2));
            settings.put(KEY_TOKEN, content.get(3));
        } else {
            settings.put(KEY_URL, DEFAULT_ANSWER);
            settings.put(KEY_USERNAME, DEFAULT_ANSWER);
            settings.put(KEY_PASSWORD, DEFAULT_ANSWER);
            settings.put(KEY_TOKEN, DEFAULT_ANSWER);
        }
    } catch (Exception ex) {

        System.out.println(ex.getMessage());
        // TODO add to logger
    }
}

```

Figura 22. Método Constructor Configurator.

Fuente: Elaboración propia.

El método encargado de cargar las configuraciones desde el archivo de texto es “fileSettingsFromFile” este método obtendrá la información como texto plano desde el archivo de configuración y en base al carácter de separación procederá a llenar el objetoJSON para posteriormente construir el objeto JSON en base a esta información.

```

public void fillSettingsFromFile(String path) {

    settings.clear();
    FileManager fmanager = new FileManager();
    for (String text : fmanager.readFile("conf.txt")) {  
        Establecemos el nombre del  
        archivo de configuración que  
        utilizaremos
    }

}

```

Figura 23. Método para llenar configuraciones desde un archivo.

Fuente: Elaboración propia.

4.2.4. Pruebas de Funcionalidad

Las pruebas realizadas para verificar el correcto funcionamiento de la aplicación “Notifier” están sujetas a la creación de un servicio temporal que permita recibir paquetes HTTP, en los cuales podemos verificar el contenido del mismo. Así también se utilizó una terminal de comandos para simular la llamada a la aplicación con sus respectivos parámetros. A continuación la tabla de pruebas realizadas para verificar el funcionamiento de la aplicación sensor, cada una de las pruebas cuenta con sus respectivos resultados.

Prueba	Resultado Esperado	Resultado Obtenido	Estado
Utilizando los parámetros: -p danger -t “procesador” -d “Esto es una prueba” Ejecutar la aplicación desde línea de comandos, en un sistema operativo Windows con una versión 6 o superior de Java.	OK	OK	PASÓ
Utilizando los parámetros: -p danger -t “procesador” -d “Esto es una prueba” Ejecutar la aplicación desde línea de comandos, en un sistema operativo del tipo Linux con una versión 6 o superior de Java.	OK	OK	PASÓ
Utilizando el parámetro: -h Ejecutar la aplicación desde línea de comandos en cualquier sistema operativo con una versión 6 o superior de Java	- Notifier v.0.2 - -h Display this message -p Alert Type, just choose one of this [alert, info, warning] -d Alert Description, if the description has more than one word please to use "" around the word -t Alert Title, if the description has more than one word please to use "" around the word Please to use: java -jar notifier.jar -p < danger info warning > -t "title Alert" -d "Description Alert"	- Notifier v.0.2 - -h Display this message -p Alert Type, just choose one of this [alert, info, warning] -d Alert Description, if the description has more than one word please to use "" around the word -t Alert Title, if the description has more than one word please to use "" around the word Please to use: java -jar notifier.jar -p < danger info warning > -t "title Alert" -d "Description Alert"	PASÓ
Utilizando el parámetro: -i Ejecutar la aplicación desde línea de comandos en cualquier sistema operativo con entorno gráfico configurado y con una versión 6 o superior de Java	Interfaz gráfica para la configuración de la aplicación sensor	Interfaz gráfica para la configuración de la aplicación sensor	PASÓ

Tabla 5. Pruebas de funcionalidad, iteración 2.

Fuente: Elaboración propia.

4.2.5. Cuadro de Avance

Código	Estimación	Duración	Estado
H7 (Prototipo de la aplicación Sensor)	5	2	Hecho
H8(Módulo de configuración)	13	6	Hecho
H9(Módulo de comunicación)	8	5	Hecho
H10(Pruebas de integración utilizando una terminal de comandos)	5	1	Hecho

Tabla 6. Cuadro de avance, iteración 2.

Fuente: Elaboración propia.

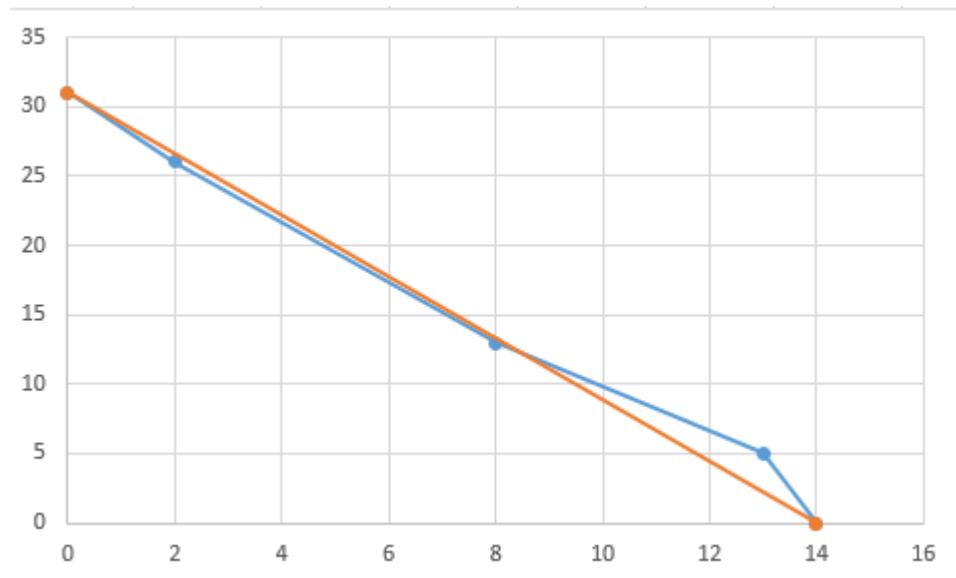


Figura 24. Diagrama de Avance de historias de usuario, Iteración 2.

Fuente: Elaboración Propia

4.2.6. Retrospectiva

Bueno	Malo	Mejorar
<ul style="list-style-type: none"> - Dado que no se tenían muchas historias de usuario, el desarrollo de las mismas fue rápido y con pocos errores. - Las herramientas de desarrollo para este tipo de aplicaciones facilitó la implementación y proporcionó mejores resultados respecto al tiempo de desarrollo. 	<ul style="list-style-type: none"> - Algunos requerimientos para el manejo de parámetros no fueron claramente detallados al inicio de la iteración por lo que tuvo que detallarse y realizar pruebas con varias opciones antes de completar el trabajo de la historia de usuario. - Dado que la aplicación de servicios no fue implementada aún, las pruebas fueron solo de funcionalidad por lo que más adelante será necesario realizar las pruebas de integración. 	<ul style="list-style-type: none"> - Las historias de usuario y los criterios de aceptación pueden ser más específicos respecto a la funcionalidad de la aplicación así como de los elementos relacionados a ella.

Tabla 7. Retrospectivas de iteración 2.

Fuente: Elaboración propia.

4.3. Iteración 3 Columba

El desarrollo de la aplicación Web de servicios tiene un énfasis en sostenibilidad y acoplamiento. Esta aplicación permitirá a los desarrolladores crear aplicaciones cliente basadas en su API. En esta iteración se trabajará en la parte del sistema que maneja los servicios los cuales se desarrollarán utilizando NodeJS como lenguaje de programación y Mongodb como manejador de base de datos.

4.3.1. Cuadro de Historias de Usuario

Código	Título	Descripción	Tipo	Estimación	Prioridad
H11	Prototipo de la aplicación Web de Servicios.	Realizar el diseño y la implementación de la ruta para recibir solicitudes a través del método HTTP POST y almacenarlas en un registro de Base de datos, esta base de datos puede tener cualquier nombre y a futuro será reemplazada.	Diseño	8	1
H12	Estructura “alert”, “user” y “sensor”.	Realizar el diseño de la estructura del documento “alerta”, “user” y “sensor” los cuales deben estar en formato JSON. El documento debe ser creado conjuntamente con los documentos de la base de datos “jarvis”	Diseño	5	1
H13	Registrar alertas.	Realizar la implementación de la funcionalidad para crear alertas, la creación debe ocurrir después de que el método HTTP POST es recibido, la ruta del método es api/alert/new . La alerta debe ser almacenada en base de datos.	Desarrollo	8	2

H13	Creación de usuarios.	<p>Realizar el diseño de la estructura del documento usuarios, la cual debe permitir registrar nuevos usuarios a través de la ruta api/user/new</p> <p>La información relevante de un usuario es:</p> <ul style="list-style-type: none"> • Nombres y apellido • Dirección Electrónica • Contraseña • Tipo de usuario (normal, administrador) • Estado usuario (conectado, desconectado) 	Desarrollo	13	3
H14	Creación sensores.	<p>Realizar el diseño de la estructura del documento sensores, la cual debe permitir registrar nuevos sensores a través de la ruta api/sensor/new</p> <p>La información relevante de un sensor es:</p> <ul style="list-style-type: none"> • Nombre • Dirección IP • Estado sensor (conectado, desconectado) 	Desarrollo	13	3

H15	Pruebas integración con la aplicación Sensor.	Utilizando un sensor de prueba y una terminal de comandos verificar que las solicitudes enviadas a través de la aplicación sensor son registradas en el documento alertas.	Pruebas	3	3
-----	---	--	---------	---	---

Tabla 8. Cuadro de historias de usuario, iteración 3.

Fuente: Elaboración propia.

4.3.2. Diseño

La aplicación de servicios web recibe las alertas de los distintos sensores instalados en cualquier ordenador, estas alertas deben ser autenticadas debido a que podrían ser generadas por equipos no asociados a ninguna cuenta de usuario, si la información de la alerta es auténtica entonces la aplicación debe registrar esta nueva alerta en la base de datos.

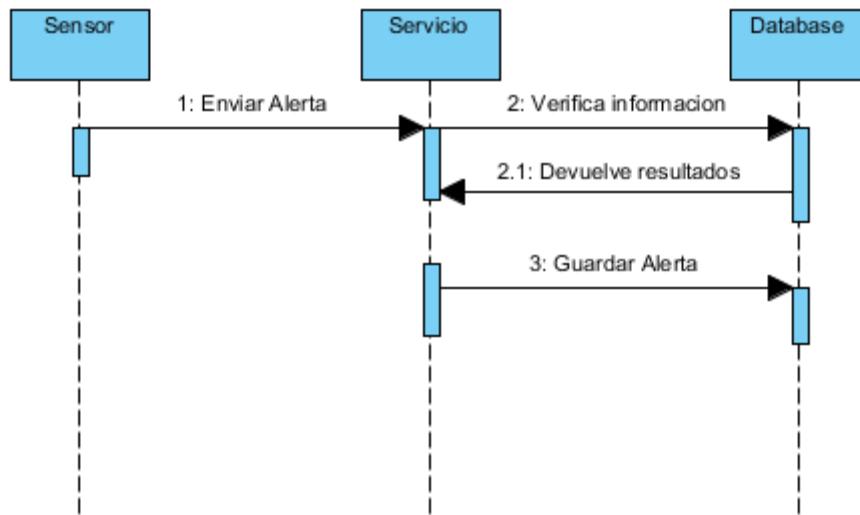


Figura 25. Diagrama de secuencia, Alerta en servicio web.

Fuente: Elaboración propia.

La aplicación de servicios web es la encargada de gestionar todas las alertas provenientes de los distintos sensores y encaminarlas a sus respectivos clientes, es por esta razón que es necesario modelar cada aspecto que se encuentre relacionado a la aplicación de servicios. Los modelos necesarios para trabajar con alertas, usuarios y sensores son los siguientes.

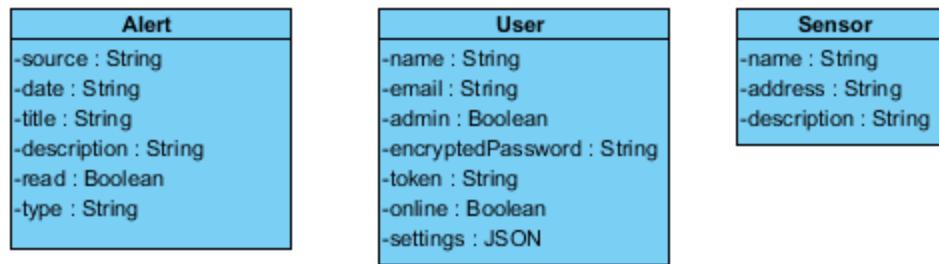


Figura 26. Modelo de objetos JSON.

Fuente: Elaboración propia.

Cada uno de los modelos representa un documento en la base de datos, por lo que el registro y obtención de información se realiza de manera sencilla y rápida, con lo que es posible manejar gran cantidad de información eficientemente.

El modelo “Alert” cuenta con 6 atributos los cuales son:

- **source**, representa la dirección IP origen de la alerta.
- **date**, representa la fecha y hora en la que fue registrada la alerta.
- **title**, representa el título de la alerta, este título nos permitirá fácilmente identificar el motivo de la alerta generada.
- **description**, representa una corta descripción del motivo de la generación de la alerta.
- **read**, representa el estado actual de una alerta, inicialmente toma el valor de falso.
- **type**, representa el tipo de una alerta, en la aplicación solo se manejan los tipos “info”, “warning”, “danger” y “unknow”.

Los tipos de alertas nos permiten clasificar y priorizar las alertas, con ayuda de interfaz de usuario podemos realizar diferentes presentaciones para cada una de las alertas. Los tipos de alerta implementados son:

- Alertas de información (info).
- Alertas de advertencia (warning).
- Alertas de peligro (danger).
- Alertas desconocidas (unknow).

El modelo “User” contendrá información específica de las preferencias de cada usuario permitiendo a las aplicaciones cliente ofrecer configuraciones personalizadas, este modelo está compuesto por los atributos:

- **name**, representa el nombre completo del usuario.
- **email**, representa el correo electrónico del usuario.
- **admin**, representa el tipo de usuario.
- **encryptedPassword**, representa la contraseña encriptada del usuario.
- **token**, representa el valor de una llave temporal que permitirá validar las alertas.
- **online**, representa el estado actual del usuario.
- **settings**, representa el conjunto de configuraciones definidas por el usuario.

En el modelo “User” podemos notar que el atributo “settings” almacena información relacionada a las configuraciones definidas por el usuario, estas configuraciones son mostradas en la siguiente gráfica.

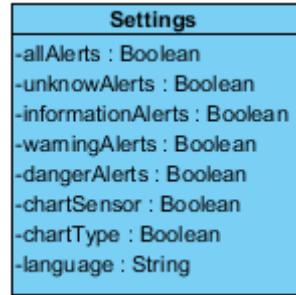


Figura 27. Modelo del objeto Settings.

Fuente: Elaboración propia.

El objetivo del modelo “Setting” es utilizar cada uno de sus atributos como una variable bandera con los que verificaremos si el usuario desea visualizar algunas alertas, por ejemplo el caso de que un usuario no desee visualizar las alertas de información debido a que no tienen un significado específico o simplemente porque solo desea enfocarse en alertas de peligro o advertencia.

Los atributos del modelo “Setting” son detallados a continuación:

- **allAlerts**, si su valor es verdad la aplicación debe mostrar todas las alertas al usuario.
- **unknowAlerts**, en caso de que sea verdad el valor de este atributo el usuario podrá ver las alertas de este tipo, caso contrario no serán mostradas.
- **informationAlerts**, si el valor de este atributo es verdad el usuario podrá ver las alertas de este tipo, caso contrario no serán mostradas.
- **warningAlerts**, si el valor de este atributo es verdad el usuario podrá ver las alertas de este tipo, caso contrario no serán mostradas.
- **dangerAlerts**, si el valor de estas atributo es verdad el usuario podrá ver las alertas de este tipo, caso contrario no serán mostradas.
- **chartSensor**, si el valor de este atributo es verdad el usuario podrá visualizar la gráfica de alertas recibidas por sensor.
- **chartType**, si el valor de este atributo es verdad el usuario podrá visualizar la gráfica de alertas por tipo.
- **language**, este atributo nos permite definir el idioma que el usuario desea usar en la aplicación por defecto es “english”.

Finalmente el modelo “Sensor” como su nombre indica nos permitirá almacenar información referida al sensor que está asociado a la aplicación este modelo solo será utilizado para tener registros de las alertas provenientes de cada sensor y así poder realizar distintas gráficas basadas en esta información.

Los atributos del modelo sensor son:

- **name**, este atributo representa el nombre del sensor, por el cual serán identificadas las alertas.
- **addresss**, este atributo representa la dirección IP del sensor.
- **description**, este atributo nos permitirá agregar una descripción simple del sensor.

En esta etapa de diseño nos enfocamos en cada uno de los modelos que serán utilizados por la aplicación de servicios, y dado que en esta iteración se realizará la implementación de las rutas para crear alertas, crear usuarios y crear sensores es necesario establecer la URL que utilizará cada una de estas implementaciones.

URL	Objetivo
http:// <dirección server>/api/alert/create	Para la creación de alertas.
http:// <dirección server>/api/user/create	Para la creación de usuarios.
http://<dirección server>/api/sensor/create	Para la creación de sensores.

Tabla 9. Definición de rutas para alertas, usuarios y sensores.

Fuente: Elaboración propia.

Basándonos en las rutas definidas se realizará la implementación de cada uno de los métodos encargados de la creación de cada uno de estos modelos.

4.3.3. Implementación

4.3.3.1. Estructura del proyecto

El proyecto fue estructurado basado en la estructura de proyectos que utilizan el Framework ExpressJS, esta estructura nos permite separar claramente servicios de modelos y de rutas de acceso a la aplicación, a continuación podemos observar la estructura del proyecto.

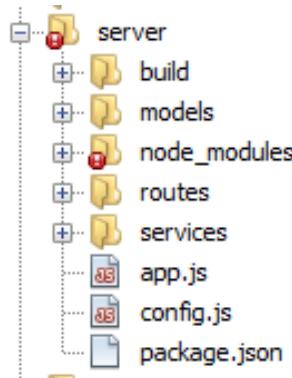


Figura 28. Estructura proyecto aplicación web servicios.

Fuente: Elaboración propia.

La estructura contiene una carpeta “build” en la cual se almacena configuraciones que permitirán compilar y construir algunas dependencias de Nodejs.

La carpeta “models” contendrá cada uno de los archivos de los modelos de la aplicación en este caso los modelos “Alert”, “User”, “Sensor”.

En la carpeta “node_modules” encontraremos todas las dependencias del proyecto respecto a librerías de encriptación, Websockets y base de datos.

En la carpeta “routes” se almacenarán los archivos de cada una de las rutas de acceso a la aplicación estos archivos a su vez contendrán la lógica necesaria para crear, modificar o eliminar cada uno de los modelos.

La carpeta “services” contendrá los archivos para realizar conexiones a base de datos o si fuese necesario hacia otro servicio útil para la aplicación.

En el archivo “app.js” se colocarán las llamadas principales a la aplicación en este archivo se inicializará la aplicación y crearán las rutas y conexiones necesarias.

El archivo config.js contiene información relacionada a la configuración de la aplicación así como el puerto de conexión o el nombre de la base de datos.

Finalmente en el archivo “package.json” toda la información referente a las dependencias, versión de la aplicación y descripción de la misma serán establecidas.

4.3.3.2. Implementación alertas nuevas

Para la creación de alertas se implementó la funcionalidad de esta característica en la ruta /api/alert la cual hace uso del método POST para recibir la información.



```
130 // POST alert
131 app.post('/api/alert/create', function(req, res) {
132
```

A screenshot of a code editor showing a snippet of Node.js code. The code defines a POST route for '/api/alert/create' using the 'app' object. The code is numbered from 130 to 132. A cursor is visible at the start of the route definition.

Figura 29. Cabecera ruta nueva alerta.

Fuente: Elaboración propia.

En la implementación de esta ruta fueron considerados los usuarios debido a que es necesario validar si el usuario que envía la alerta corresponde al grupo de usuarios registrados y en el caso de que lo este es necesario verificar el “Token” o llave que tiene para evitar que cualquier usuario genere alertas para cualquier sensor.

```

User.findOne({ email: req.body.email }, function(error, user) {
  if (user.token != req.body.token) {
    console.log('Invalid Token');
    res.send(500, 'Invalid Token');
    return;
  }
});

// POST create
Alert.create({
  source: req.body.source,
  date: req.body.date,
  type: req.body.type,
  title: req.body.title,
  description: req.body.description,
  read: false
}, function(err, alert) {
  if (err) {
    res.send(500, 'error-post-alert' + err);
    return;
  }

  // Send the alert to the client
  for (var username in app.connections) {
    app.connections[username].emit('alert', alert);
  }
});

res.send(200);

```

Figura 30. Función crear alerta.

Fuente: Elaboración propia.

La función crear alerta después de verificar la llave o “Token” proporcionados procede a almacenar la información de la alerta en base de datos haciendo uso del ORM Mongoose la nueva alerta es almacenada en base de datos a través del método “create”.

Ahora con la información de la alerta almacenada en base de datos se procede a enviar las notificaciones a los usuarios conectados a la aplicación. En este caso se utilizó una estructura de control para iterar en el arreglo de usuarios conectados, en este punto la aplicación no tiene registro de ningún usuario, porque la implementación de esta funcionalidad será llevada a cabo en posteriores iteraciones.

4.3.4. Pruebas de Funcionalidad

A continuación la tabla de las pruebas realizadas para verificar el funcionamiento de la aplicación web de servicios, cada una de las pruebas cuenta con sus respectivos resultados.

Las pruebas demostradas a continuación fueron llevadas a cabo en un sistema operativo del tipo GNU/Linux en el mismo entorno de red de la máquina que alberga al servicio web, con estas precondiciones fueron llevadas a cabo las pruebas mencionadas en la tabla.

Prueba	Resultado Esperado	Resultado Obtenido	Estado
Ejecutar el comando: <pre>curl -d '{"server":"127.0.0.1", "date":"04-03-2014", "message": "Alert from console", "read": false, "type": "danger"}' -H "Content-Type: application/json" http://localhost:3000/api/alert/create</pre>	Registro de una nueva alerta, en el modelo de base de datos “Alert”, donde el tipo de alerta es danger	Registro de una nueva alerta, en el modelo de base de datos “Alert”, donde el tipo de alerta es danger	PASÓ
Ejecutar el comando: <pre>curl -d '{"server":"127.0.0.1", "date":"04-03-2014", "message": "Alert from console", "read": false, "type": "warning"}' -H "Content- Type: application/json" http://localhost:3000/api/alert/create</pre>	Registro de una nueva alerta, en el modelo de base de datos “Alert”, donde el tipo de alerta es warning	Registro de una nueva alerta, en el modelo de base de datos “Alert”, donde el tipo de alerta es warning	PASÓ
Ejecutar el comando: <pre>curl -d '{"server":"127.0.0.1", "date":"04-03-2014", "message": "Alert from console", "read": false, "type": "info"}' -H "Content-Type: application/json" http://localhost:3000/api/alert/create</pre>	Registro de una nueva alerta, en el modelo de base de datos “Alert”, donde el tipo de alerta es info	Registro de una nueva alerta, en el modelo de base de datos “Alert”, donde el tipo de alerta es info	PASÓ
Ejecutar el comando: <pre>curl -d '{"server":"127.0.0.1", "date":"04-03-2014", "message": "Alert from console", "read": false, "type": ""}' -H "Content-Type: application/json" http://localhost:3000/api/alert/create</pre>	Registro de una nueva alerta, en el modelo de base de datos “Alert”, donde no existe un tipo de alerta	Registro de una nueva alerta, en el modelo de base de datos “Alert”, donde no existe un tipo de alerta	PASÓ

Tabla 10. Pruebas de funcionalidad realizadas en iteración 3.

Fuente: Elaboración propia.

4.3.5. Pruebas de Integración

Las pruebas de integración llevadas a cabo en esta iteración tuvieron como enfoque la configuración de la aplicación sensor que nos permite enviar alertas desde cualquier dispositivo con Java instalado y la aplicación de servicios de Web. Ambas aplicaciones fueron puestas a prueba de manera que las alertas generadas desde la aplicación sensor puedan ser registradas en la base de datos de la aplicación servicio.

Prueba	Resultado Esperado	Resultado Obtenido	Estado
Ejecutar el comando: <code>java -jar Sensor.jar -p info -t "Alert Title" -d "Alert description"</code>	En consola: OK En base de datos: alerta registrada del tipo “info”	En consola: OK En base de datos: alerta registrada del tipo “info”	PASÓ
Ejecutar el comando: <code>java -jar Sensor.jar -p warning -t "Alert Warning" -d "Alert description"</code>	En consola: OK En base de datos: alerta registrada del tipo “warning”	En consola: OK En base de datos: alerta registrada del tipo “warning”	PASÓ
Ejecutar el comando: <code>java -jar Sensor.jar -p danger -t "Alert Danger" -d "Alert description"</code>	En consola: OK En base de datos: alerta registrada del tipo “danger”	En consola: OK En base de datos: alerta registrada del tipo “danger”	PASÓ
Detener la aplicación de servicios y ejecutar el siguiente programa Ejecutar el comando: <code>java -jar Sensor.jar -p info -t "Alert Title" -d "Alert description"</code>	En consola: Destino inalcanzable En base de datos: Ningún registros nuevo debería ser insertado	En consola: Destino inalcanzable En base de datos: Ninguna alerta nueva fue insertada	PASÓ

Tabla 11. Pruebas de integración realizadas en iteración 3.

Fuente: Elaboración propia.

4.3.6. Cuadro de Avance

Código	Estimación	Duración	Estado
H11 (Prototipo de la aplicación Web de Servicios.)	8	5	Hecho
H12 (Estructura “alerta”.)	5	1	Hecho
H13 (Registrar alertas.)	8	3	Hecho
H14 (Creación usuarios.)	13	2	Hecho
H14 (Creación sensores.)	13	1	Hecho
H15 (Pruebas integración con la aplicación Sensor.)	3	2	Hecho

Tabla 12. Cuadro de avance, iteración 3.

Fuente: Elaboración propia.

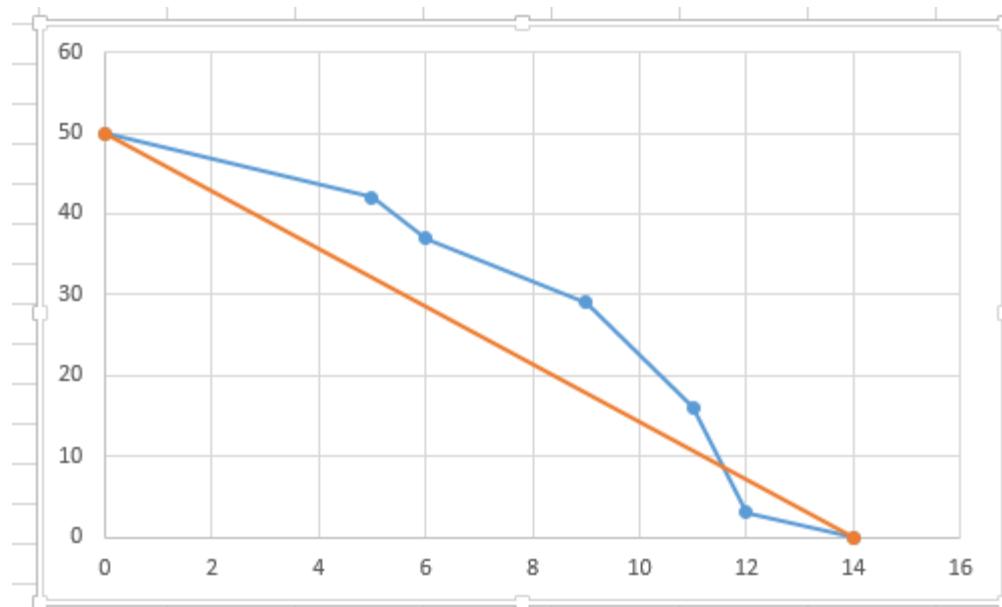


Figura 31. Diagrama de avance Iteración 3.

Fuente: Elaboración propia.

4.3.7. Retrospectiva

Bueno	Malo	Mejorar
<ul style="list-style-type: none"> - El nivel de detalle en las historias de usuario ayudo a que la implementación sea clara. 	<ul style="list-style-type: none"> - Las pruebas deben ser más claras y en lo posible se deben mejorar las mismas. - En general la estimación de las historias de usuario no fueron coherentes, dado que las mismas no fueron completadas con el esfuerzo estimado. 	<ul style="list-style-type: none"> - Es necesario trabajar más en historias de usuario relacionadas a servicios web para que las aplicaciones cliente puedan fácilmente integrarse. - Se deben considerar factores como pruebas y documentación al momento de realizar la estimación.

Tabla 13. Retrospectiva iteración 3.

Fuente: Elaboración propia.

4.4. Iteración 4 Cygnus

El diseño de la aplicación Web cliente tiene mayor énfasis en ofrecer una interfaz de usuario útil, que permita a los usuarios tener una grata experiencia. En esta iteración se realizarán los respectivos diseños para cada una de las páginas de la aplicación Web cliente, las cuales posteriormente obtendrán toda la información desde los servicios Web.

4.4.1. Cuadro de Historias de Usuario

Código	Título	Descripción	Tipo	Estimación	Prioridad
H16	Página de inicio.	<p>Como página de inicio se debe contar con tres diferentes secciones, las cuales son la cabecera, el contenido principal y el pie de página.</p> <p>CA:</p> <p>La cabecera debe permitir a los usuarios acceder a la página principal, a través de campos de acceso.</p> <p>El contenido principal de la aplicación debe ofrecer una sección de información de la aplicación la cual tendrá énfasis en las funcionalidades que la aplicación ofrece, es recomendable dividir la sección de contenido en 2 partes: la parte de información y la parte de registro de nuevos usuarios donde se utilizará un botón para acceder a la página de creación de usuarios.</p> <p>El pie de página debe utilizar los mismos colores que fueron utilizados en la cabecera.</p>	Diseño	8	1

H17	Página de configuraciones.	<p>La página de configuraciones debe permitir a los usuarios realizar cambios de la aplicación fácilmente para esto es necesario que esta página resalte claramente cada grupo de configuraciones que el usuario puede realizar.</p> <p>CA:</p> <ul style="list-style-type: none"> La página de configuración debe contar con los siguientes campos de configuración. Todas las alertas activadas. Desactivar alertas de peligro. Desactivar alertas de advertencia. Desactivar alertas de información. Desactivar alertas desconocidas. Desactivar la gráficas de porcentaje de alertas recibidas. Desactivar la gráfica de porcentaje de alertas por sensor. Cambiar el idioma de la aplicación. 		Diseño	8	2
H18	Página de creación de usuarios.	<p>La página de creación de usuarios debe contener los siguientes campos:</p> <p>CA:</p> <ul style="list-style-type: none"> Nombre y Apellido. Dirección de correo electrónico. Contraseña. Repetir contraseña. Botón para completar la operación. 		Diseño	5	2

H19	Página tablero.	<p>La página tablero es la primera página que el usuario podrá visualizar después de ingresar y validar sus credenciales, esta página debe contar con los siguientes elementos:</p> <p>CA:</p> <p>Una sección donde puedan mostrarse las gráficas de alertas por tipo y alertas por sensor.</p> <p>Una sección lateral que permita a los usuarios navegar en las diferentes páginas de la aplicación.</p> <p>Una sección que permita a los usuarios visualizar las alertas nuevas registradas, en este caso cada alerta puede ser representada por un botón.</p>	Diseño	8	1
H20	Página de creación Sensores.	<p>La página de creación de sensores en estructura es similar a la página de creación de usuarios diferenciándose en los campos necesarios para el registro de sensores.</p> <p>CA:</p> <p>La página de creación de sensores debe contener los siguientes campos:</p> <ul style="list-style-type: none"> Nombre. Dirección IP. Descripción. 	Diseño	5	2

H21	Página de listado de usuarios.	<p>Para el diseño de la página de listado de usuarios se deben tomar en cuenta las siguientes funcionalidades.</p> <p>CA:</p> <p>Dado que los datos de usuarios pueden a llegar a ser altos es necesario que la página utilice paginación.</p> <p>Los usuarios deben estar ordenados de forma alfabética en una tabla, la cabecera de dicha tabla debe mostrar el nombre de cada campo.</p> <p>Cada fila debe utilizar un control del tipo “checkbox” para seleccionar al usuario de la fila.</p> <p>La tabla debe contar con un control del tipo “checkbox” que permita seleccionar a todos los usuarios.</p> <p>La tabla de usuarios debe permitir a los mismos seleccionar uno de ellos para realizar la edición de su información.</p> <p>La página debe contar con los botones de eliminar y crear.</p>	Diseño	8	2
H22	Página de listado de alertas.	La página de listado de alertas utilizará la misma estructura que la página de listado de usuarios con la excepción que en esta página no se contará con el botón Crear.	Diseño	5	2
H23	Página de listado de sensores.	La página de listado de sensores utilizará la misma estructura que la página de listado de usuarios.	Diseño	5	2

Tabla 14. Cuadro de historias de usuario, iteración 4.

Fuente: Elaboración propia.

4.4.2. Diseño

El diseño de la aplicación Web cliente nos permite enfocarnos en la experiencia del usuario a través de distintos diseños de páginas podemos ofrecer una vista previa de cada una de ellas y fácilmente el usuario determinará qué tan útil o accesible será cada una de las páginas.

4.4.2.1. Página de Inicio

La página de inicio será la encargada de ofrecer una breve explicación de la aplicación así como también encaminará a los usuarios a acceder al sistema o a las distintas secciones de la aplicación web cliente.

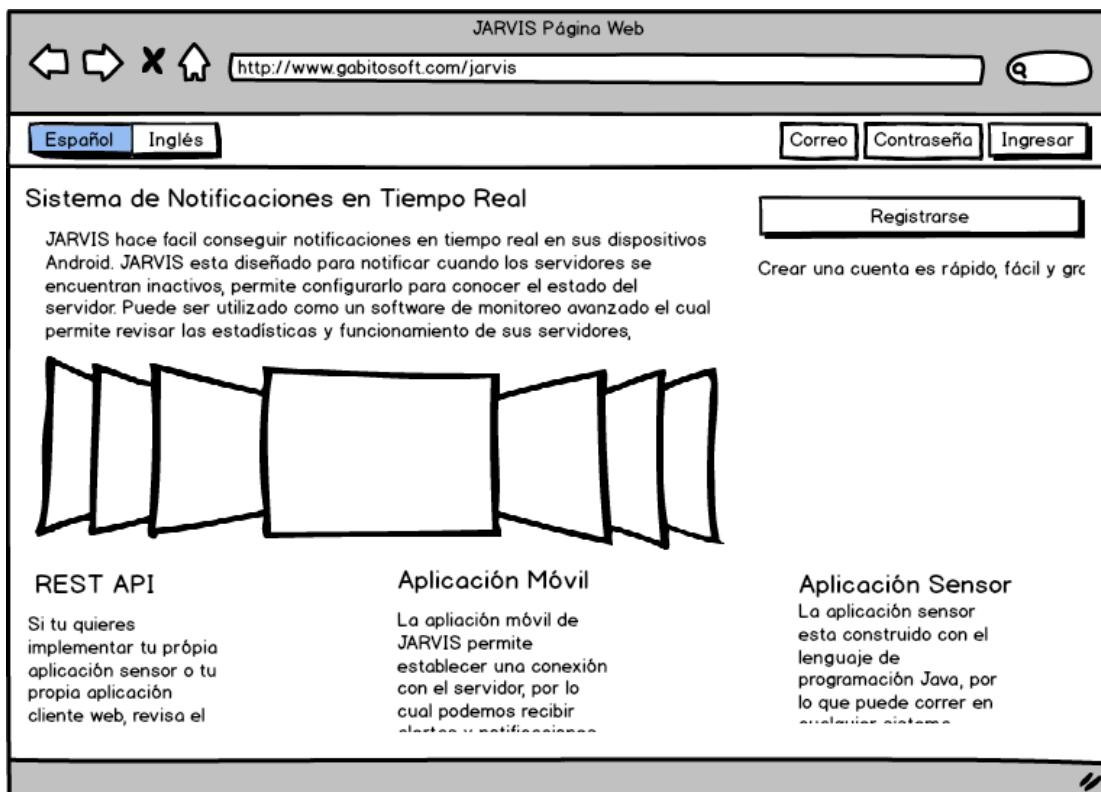


Figura 32. Diseño página inicio, aplicación web cliente.

Fuente: Elaboración propia.

La página de inicio representa el punto de partida de la aplicación web cliente por lo cual esta página contiene información multimedia que demostrará los aspectos primordiales de la aplicación.

4.4.2.2. Página Tablero

La página tablero representa la primera página que el usuario visualizará después de acceder con sus credenciales, en esta página el usuario podrá obtener un rápido resumen de las nuevas alertas generadas, el porcentaje de alertas generadas por cada uno de los sensores y el porcentaje de alertas generadas por tipos de alertas.

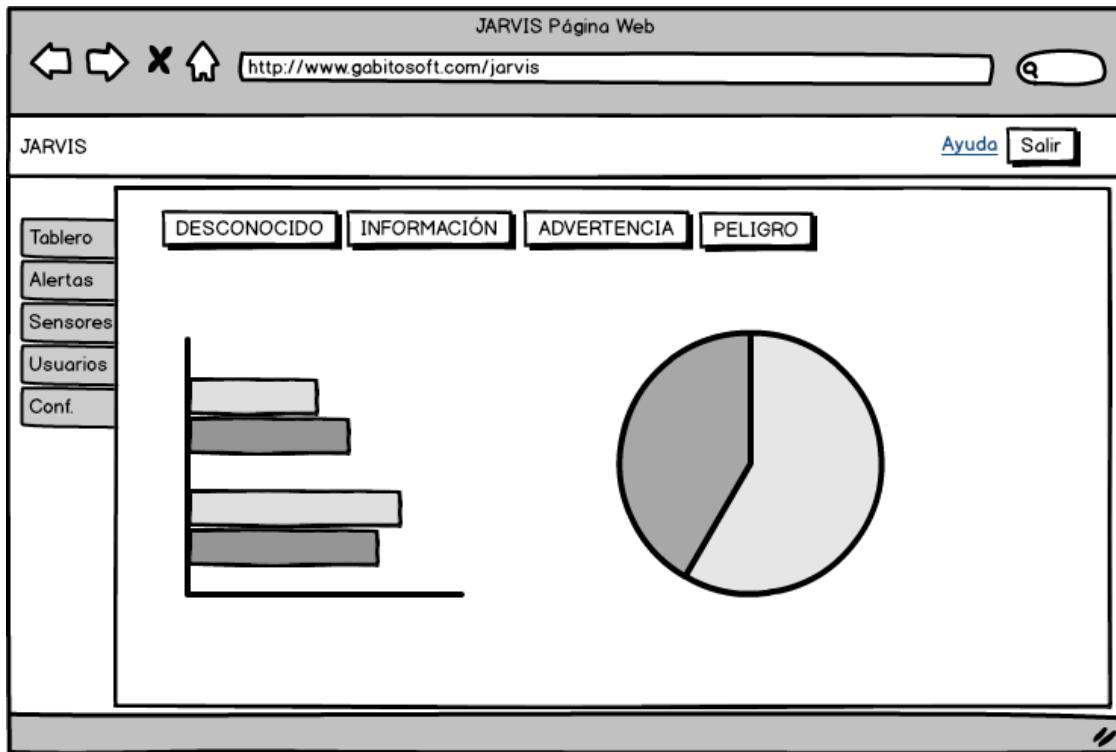


Figura 33. Diseño página tablero, aplicación web cliente.

Fuente: Elaboración propia.

Haciendo uso del menú que se encuentra en la parte lateral izquierda de la página el usuario podrá navegar entre las distintas páginas de la aplicación.

4.4.2.3. Página Listado Alertas

En la página alertas el usuario podrá listar todas las alertas generadas por los distintos sensores, también en esta página el usuario eliminará o modificará las alertas mostradas. Un filtro en la página ayudará a los usuarios a visualizar las alertas del tipo seleccionado, de esta forma solo las alertas relevantes para el usuario serán mostradas.

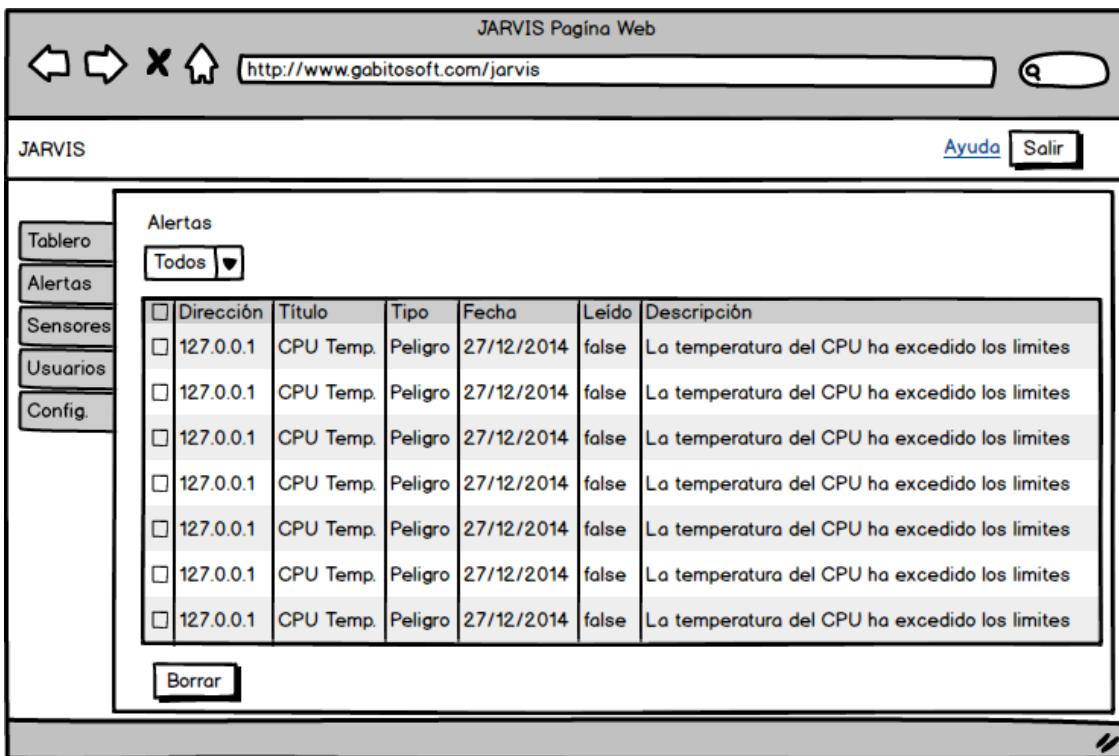


Figura 34. Diseño página alertas, aplicación web cliente.

Fuente: Elaboración propia.

Si el usuario desearía eliminar varias alertas simplemente debe seleccionarlas y a continuación presionar el botón eliminar, en el caso de que los usuarios desearían eliminar todas las alertas de la página deberían seleccionar el control “checkbox” ubicado en la cabecera de la tabla de alertas al seleccionar este control todas las alertas mostradas en el página serán seleccionadas y nuevamente presionando el botón eliminar todas estas alertas serían eliminadas.

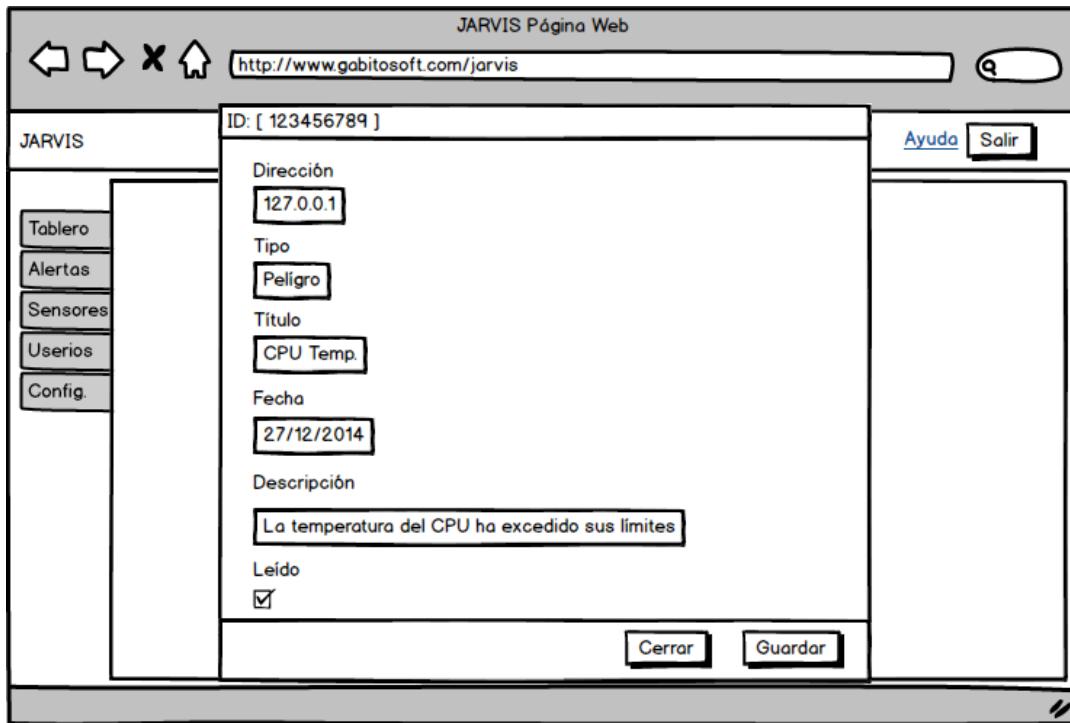


Figura 35. Diseño cuadro dialogo editar alerta, aplicación web cliente.

Fuente: Elaboración propia.

Dado que esta página permite a los usuarios editar la información de cada una de las alertas mostradas, es necesario realizar el diseño de un cuadro de dialogo que permita realizar esta tarea. El cuadro de dialogo debe ser mostrado cuando se realiza clic sobre cualquier ítem de una alerta y una vez que es mostrado dicho cuadro el usuario podrá cambiar la información relevante de la alerta seleccionada.

4.4.2.4. Página Listado Sensores

El diseño de la página sensores es similar al de la página alertas dado que se mantendrá el mismo esquema respecto al diseño general de la aplicación web cliente donde solo se cambiarán los campos relacionados a cada respectivo documento.

La página de listado de sensores utilizará una tabla para mostrar la información de cada uno de los sensores los cuales a su vez cuentan con los controles para modificar o seleccionarlos. En esta página también será posible eliminar los sensores seleccionados.

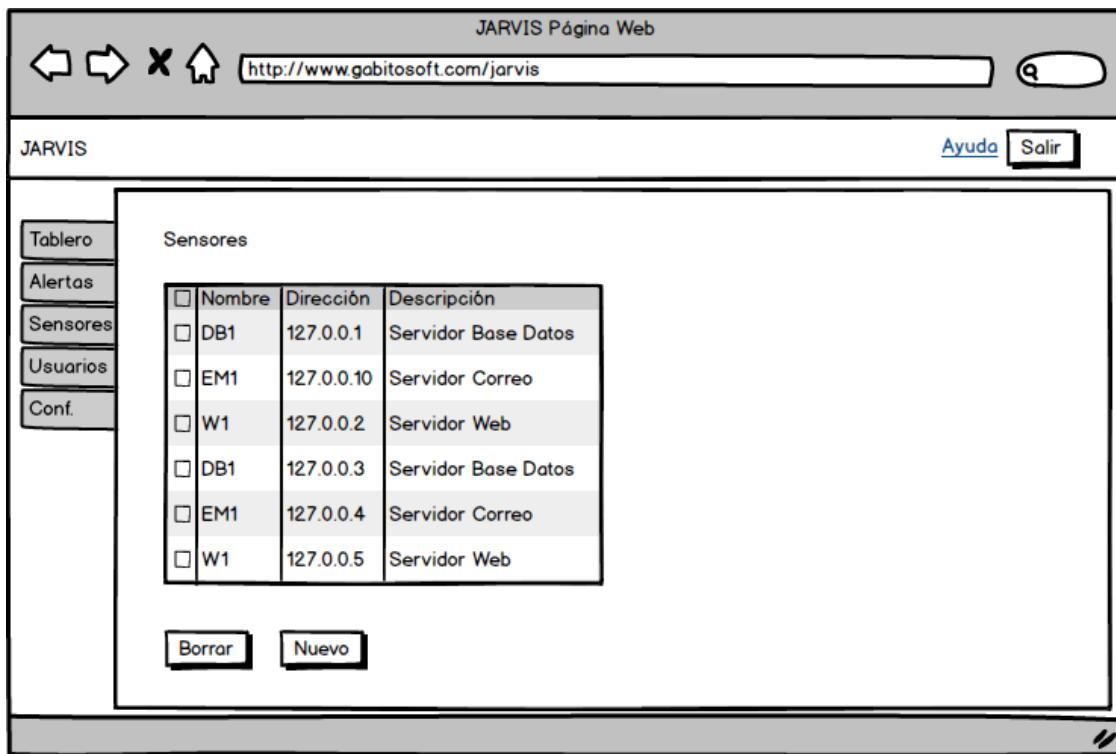


Figura 36. Diseño listado de sensores, aplicación web cliente.

Fuente: Elaboración propia.

El usuario podrá modificar la información de cada uno de los sensores seleccionándolos, el cuadro de dialogo para la edición de los sensores es similar al cuadro de dialogo de la página alertas, pero con diferencia en los campos nombre, dirección y descripción. Este cuadro de dialogo al igual que otros cuenta con los botones de cerrar y guardar los cuales se encuentran ubicados en el pie del cuadro de dialogo.

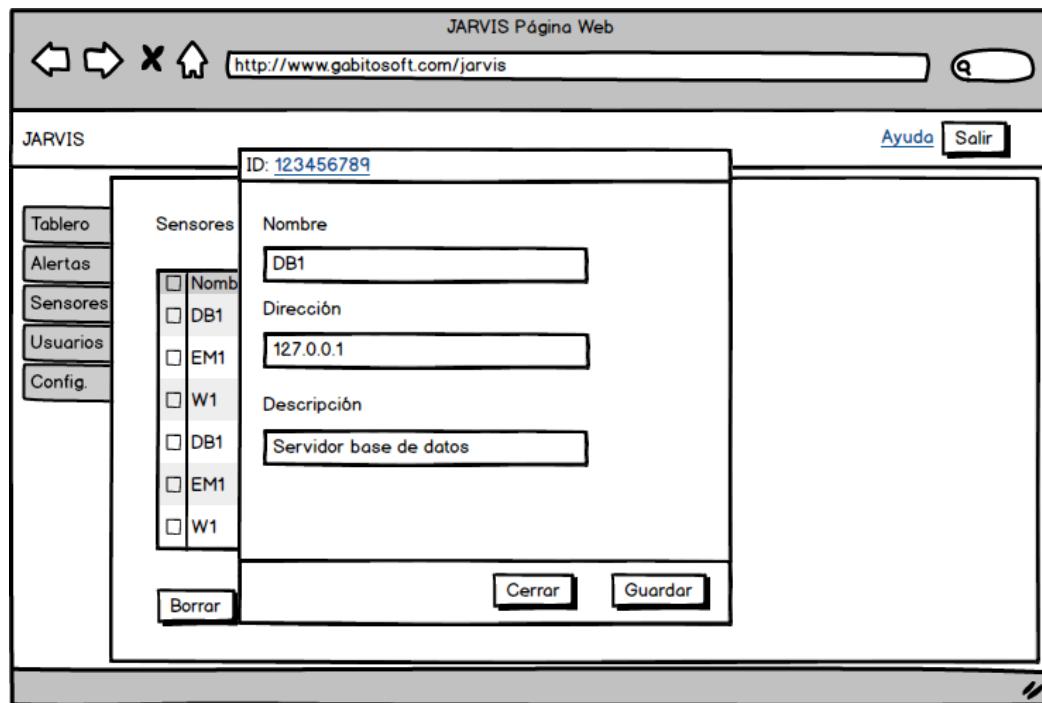


Figura 37. Diseño cuadro dialogo edición sensor, aplicación web cliente.

Fuente: Elaboración propia.

4.4.2.5. Página Creación Sensores

La página de creación de sensores permitirá a los usuarios crear un sensor utilizando la información proporcionada por el usuario.

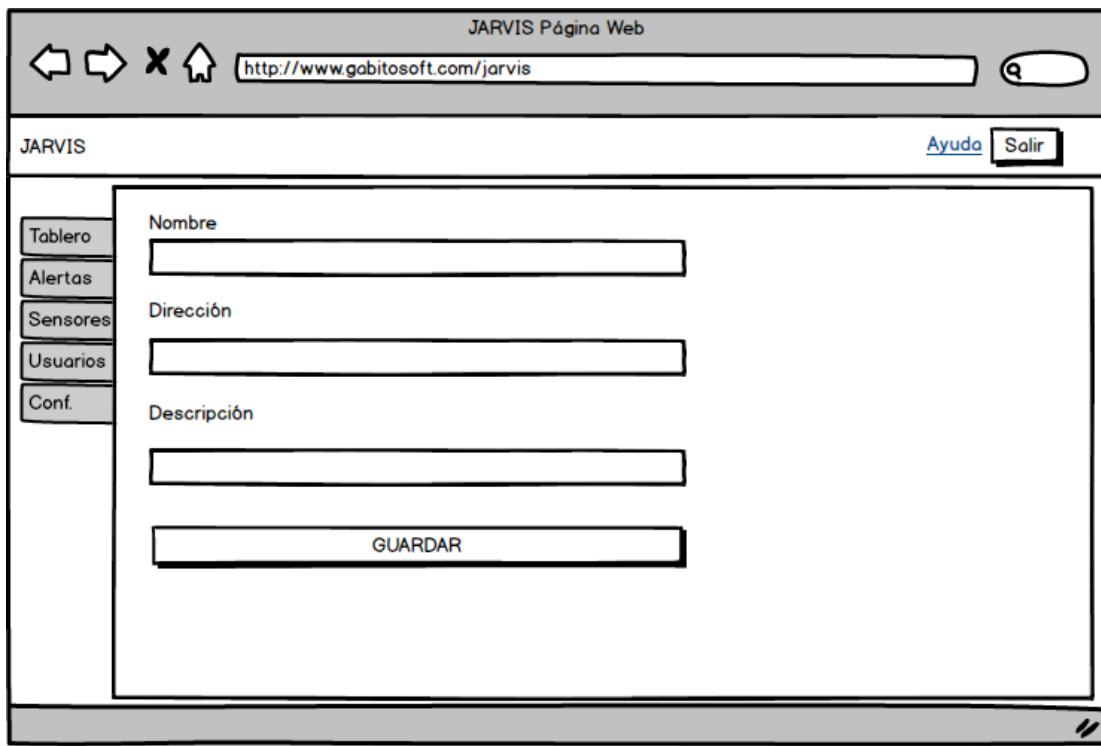


Figura 38. Diseño creación sensores, aplicación web cliente.

Fuente: Elaboración propia.

4.4.2.6. Página Listado Usuarios

La página de listado de usuarios, mantiene el mismo esquema que la página de listado de alertas y listado de sensores, en esta página los usuarios obtendrán información relacionada a usuarios registrados y fácilmente podrá identificar a los usuarios administradores de los que no lo son.



Figura 39. Diseño página listado usuarios, aplicación cliente web.

Fuente: Elaboración propia.

La página de listado de usuarios solo podrá ser visible para los usuarios administradores del sistema, en esta página los usuarios administradores podrán cambiar el tipo de usuario de cada uno de los listados así como también la información personal registrada.

4.4.2.7. Página Creación Usuarios

La página de creación de usuarios permite a los cualquier usuario del sistema crear nuevos usuarios, los usuarios creados desde esta página inicialmente no son usuarios administradores por lo que si se desearía crear un usuario del tipo administrador sería necesario cambiar el tipo de usuario desde el cuadro de dialogo para editar usuarios.

Esta página al igual que la página crear sensores mantiene el mismo esquema y cuenta con controles similares, tomando en cuenta los campos necesarios para registrar la información de un usuario.

The screenshot shows a web-based application interface for JARVIS. At the top, there is a header bar with icons for back, forward, stop, and home, followed by the URL <http://www.gabitosoft.com/jarvis>. Below the header, the title "JARVIS" is displayed, along with links for "Ayuda" and "Salir". On the left side, a vertical menu bar contains the following items: Tablero, Alertas, Sensores, Usuarios, and Config., with "Config." currently selected. The main content area contains a form for creating a new user. The form fields are labeled "Nombre", "Correo electrónico", "Contraseña", and "Confirmación", each with an associated input field. At the bottom of the form is a large button labeled "GUARDAR".

Figura 40. Diseño creación de usuarios, aplicación cliente web.

Fuente: Elaboración propia.

La página de creación de usuarios puede ser accedida desde 2 diferentes enlaces, el primero es desde la página de inicio a través del botón “Registrarse”. Otra forma para acceder a esta página es desde la página listar usuarios a través del botón “Nuevo”.

4.4.2.8. Página Configuraciones

Para el diseño de la página de configuraciones fueron tomados en cuenta cada uno de los requerimientos de usuario respecto a las configuraciones que realizaría, dado que esta página es de mucha importancia para los usuarios, debe ser fácilmente utilizada ayudando al usuario a entender claramente cada una de las opciones de configuración.

La página se encuentra dividida en tres secciones utilizando controles del tipo “checkbox” para poder fácilmente habilitar o deshabilitar cada opción de configuración.

La primera sección representa las configuraciones que se pueden realizar sobre las alertas, cada uno de los controles permitirá a los usuarios habilitar todas las alertas en general, alertas desconocidas, alertas de información, alertas de advertencia y alertas de peligro, cuando el usuario deshabilite cualquiera de estas configuraciones simplemente no serán enviadas las notificaciones de alertas, sin embargo cada alerta será registrada en la base de datos. La segunda sección representa la sección de gráficos, si deshabilitamos las opciones de esta sección los gráficos mostrados en la página tablero no serán mostrados. Finalmente la tercera sección corresponde a la sección de configuración de lenguaje, la aplicación contará con la opción de configurar el lenguaje de la aplicación, el lenguaje inicial de la aplicación será el inglés, también se contará con la opción de seleccionar el lenguaje español si fuese requerido.

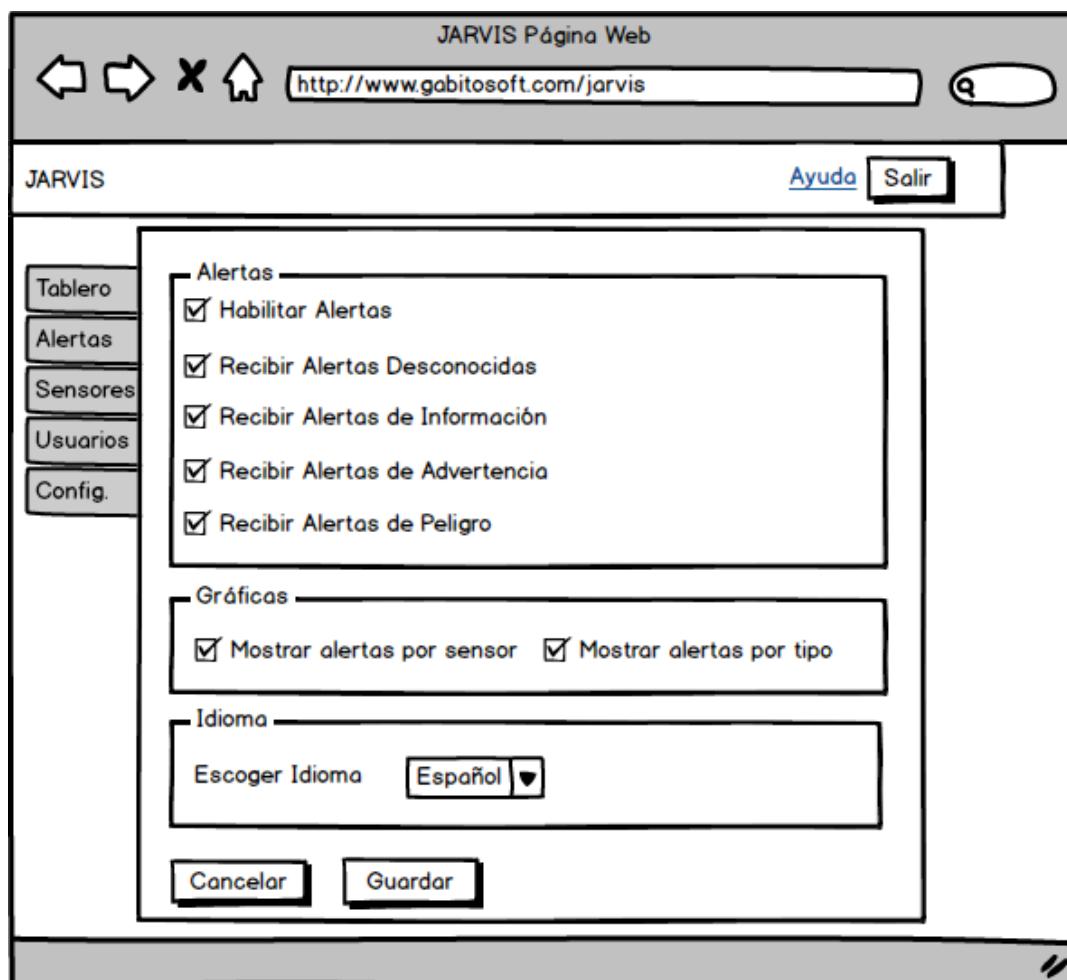


Figura 41. Diseño página configuraciones, aplicación web cliente.

Fuente: Elaboración propia.

Al igual que anteriores páginas la página de configuraciones contará con los controles para cancelar los cambios, así como también para registrar los cambios en base de datos.

4.4.3. Cuadro de Avance

Código	Estimación	Duración	Estado
H16 (Página de inicio.)	13	2	Hecho
H17 (Página de configuraciones.)	8	2	Hecho
H18 (Página de creación de usuarios.)	5	1	Hecho
H19 (Página tablero.)	8	2	Hecho
H20 (Página de creación Sensores.)	5	1	Hecho
H21 (Página de listado de usuarios.)	8	2	Hecho
H22 (Página de listado de alertas.)	5	1	Hecho
H23 (Página de listado de sensores.)	5	1	Hecho

Tabla 15. Cuadro de avance, iteración 4.

Fuente: Elaboración propia.

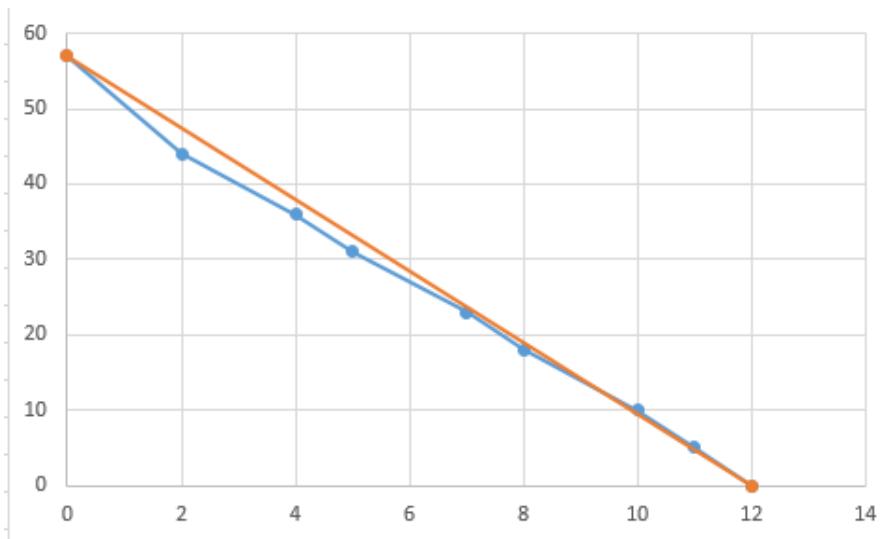


Figura 42. Diagrama de avance de historias de usuario, Iteración 4.

Fuente: Elaboración propia.

4.4.4. Retrospectiva

Bueno	Malo	Mejorar
<ul style="list-style-type: none"> - Las historias de usuarios planificadas para esta iteración fueron completadas. - Se utilizaron nuevas herramientas para el diseño de las diferentes páginas. - Separando el diseño de las diferentes páginas se pudo completar rápidamente las historias asignadas. 	<ul style="list-style-type: none"> - A pesar que se completaron todas las historias de usuario, aún existe una sobre estimación. 	<ul style="list-style-type: none"> - En lo posible se debe intentar realizar una mejor valoración de las historias de usuarios. - Se debe continuar trabajando en los detalles de cada historia de usuario para simplificar el trabajo en ella.

Tabla 16. Retrospectiva iteración 4.

Fuente: Elaboración propia.

4.5. Iteración 5 Crater

En la anterior iteración se desarrollaron los diseños de las páginas de la aplicación web cliente en los cuales se definió la estructura de las páginas y los controles que estarán contenidos. Cada control mostrará información obtenida desde los servicios web, que estarán configurados para enviar información en formato JSON.

Las historias de usuario de esta iteración estarán enfocadas en el desarrollo de la aplicación web cliente basando su implementación en los diseños presentados, también se realizarán las implementaciones necesarias del lado del servidor para proveer de información y mecanismos de comunicación a la aplicación web cliente.

4.5.1. Cuadro de Historias de Usuario

Código	Título	Descripción	Tipo	Estimación	Prioridad
H24	Página de inicio.	<p>La página de inicio representará la primera página que el usuario visualizará después de ingresar la respectiva URL de la aplicación web cliente.</p> <p>CA:</p> <p>Como usuario de la aplicación web es necesario que la página sea mostrada tanto en dispositivos móviles como en ordenadores, los colores para la página serán los mismos que utiliza el tema “Jumbo” de la librería Bootstrap. La implementación debe basarse en el diseño presentado previamente, el texto e imágenes serán trabajadas en posteriores historias de usuario, por lo que no es necesario especificar este contenido en esta iteración.</p>	Desarrollo	5	1

H25	Servicio de autenticación, integrado a la aplicación web cliente.	<p>La implementación del servicio de autenticación debe permitir a los usuarios registrados en el sistema acceder a la página tablero.</p> <p>CA:</p> <p>Basados en las estructuras de rutas y servicios definidos en el lado del servidor, se debe crear un nuevo servicio que permita a los usuarios validar sus credenciales, las cuales serán pasadas como parámetros en la solicitud enviada al servicio, la ruta definida para esta implementación será: /api/user/login</p> <p>En caso de que los parámetros sean inválidos o se produzca alguna excepción en el lado del servidor se procederá a mostrar un mensaje de error en la página de inicio. Si la autenticación fue satisfactoria la aplicación será direccionada hacia la página tablera.</p>		Desarrollo	8	1
-----	---	---	--	------------	---	---

H26	Rutas y estructura de la aplicación web cliente.	<p>Todas las páginas de la aplicación web cliente utilizarán la misma estructura, excepto la página de inicio, por lo cual es necesario definir una sola estructura.</p> <p>CA:</p> <p>Haciendo uso de los beneficios del framework AngularJS definir e implementar la estructura de la aplicación web cliente, la cual utilizará rutas para facilitar la navegación dentro de la aplicación, las cuales serán:</p> <ul style="list-style-type: none"> /dashboard /alert /sensor /newSensor /setting /user /newUser /profile /help 		Desarrollo	13	1
-----	--	---	--	------------	----	---

H27	Página tablero.	<p>La página tablero debe permitir a los usuarios visualizar información relevante relacionada al estado de sus servidores. Por lo que el desarrollo de la página será basada en el diseño previamente realizado.</p> <p>CA:</p> <p>Se debe utilizar librerías gráficas que permitan mostrar relación entre alertas y sensores así como también la relación entre las alertas y sus diferentes tipos, inicialmente se utilizarán datos fijos.</p> <p>Los botones situados en la parte superior de la página tablero deben permitir a los usuarios navegar hacia la página alertas. La implementación de la página tablero debe basarse en el diseño realizado previamente.</p>	Desarrollo	13	1
H28	Página listar alertas.	<p>El desarrollo de la página para listar alertas está orientado a mostrar grandes cantidades de información.</p> <p>CA:</p> <p>El desarrollo de la página debe basarse en el diseño realizado previamente, en el cual se utilizará las características del tema “Jumbo” de bootstrap.</p> <p>Dado que la página manejará grandes cantidades de información se debe considerar utilizar algún tipo de paginación que permita navegar al usuario entre las distintas alertas generadas.</p>	Desarrollo	8	2

H29	Servicio Eliminar alertas.	<p>La implementación del servicio para eliminar alertas debe ser realizada utilizando la estructura de las rutas desarrolladas previamente.</p> <p>CA:</p> <p>La ruta para eliminar alertas utilizará un parámetro para identificar la ruta especificada.</p> <p>La ruta que se definirá para utilizar el servicio de eliminar alertas será la siguiente:</p> <p>/api/alert/:id</p> <p>La cual utilizará el método REST “delete”</p>	Desarrollo	8	2
H30	Servicio Modificar alertas.	<p>La implementación del servicio para modificar alertas debe ser realizada utilizando la estructura de las rutas desarrolladas previamente.</p> <p>CA:</p> <p>La ruta para modificar alertas requiere que se utilice un parámetro para identificar la alerta seleccionada.</p> <p>La ruta que se utilizará para acceder al servicio de modificación de alertas será la siguiente:</p> <p>/api/alert/:id</p> <p>La cual utilizará el método REST “put”</p>	Desarrollo	8	2

H31	Página Configuraciones	<p>La página de configuraciones se basará en el diseño realizado previamente, el cual utilizará los colores del tema “Jumbo” de Bootstrap.</p> <p>CA:</p> <p>Inicialmente los controles contendrán información estática la cual en posteriores iteraciones recibirán información desde los servicios web.</p>	Desarrollo	8	2
-----	------------------------	--	------------	---	---

Tabla 17. Cuadro de historias de usuario, iteración 5.

Fuente: Elaboración propia.

4.5.2. Diseño

En esta iteración se trabajó en el diseño de 3 nuevos servicios web los cuales fueron pensados para ser implementados en los archivos de usuarios así como también en los archivos de alertas, estos diseños están representados de la siguiente manera.

4.5.2.1. Autenticación de Usuarios

La autenticación de usuarios utilizará los parámetros email y contraseña, los cuales a su vez serán verificados en base de datos, el resultado de esta consulta será enviado y procesado por el servicio web el cual a su vez se encargará de retornar el código y estado de la autenticación.

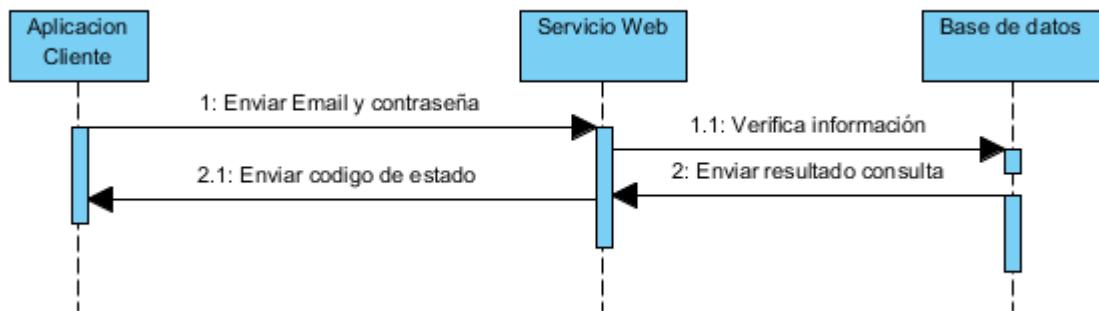


Figura 43. Diagrama de secuencia autenticación de usuarios.

Fuente: Elaboración propia.

4.5.2.2. Modificar Alertas

Para llevar a cabo la modificación de alertas es necesario contar con el identificador de la alerta que será modificada, una vez que la alerta es encontrada en base de datos, se procede a actualizar la información de dicha alerta, finalmente el servicio web se encargará de enviar el estado de la modificación de la alerta a la aplicación cliente.

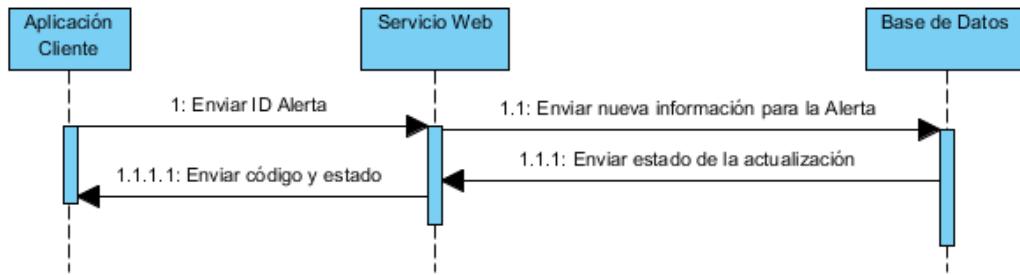


Figura 44. Diagrama de secuencia modificar alerta.

Fuente: Elaboración propia.

4.5.2.3. Eliminar Alertas

El proceso de eliminar alertas es muy parecido al de modificar alertas, por lo que únicamente se diferencian de la operación que se realiza sobre base de datos. En este caso el ID de la alerta a ser eliminada es enviado al servicio web el cual se encarga de proceder con esta acción en base de datos y posteriormente enviar el código y resultado a la aplicación cliente.

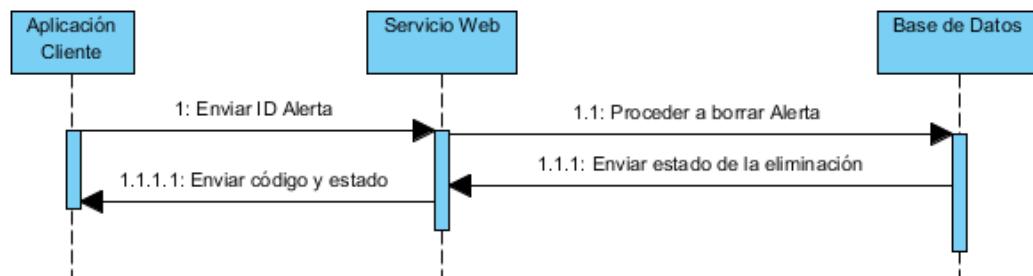


Figura 45. Diagrama de secuencia eliminar alerta.

Fuente: Elaboración propia.

4.5.3. Implementación

4.5.3.1. Página de inicio

El proyecto aplicación web cliente fue estructurado en base a los requerimientos previamente realizados, para los cuales fueron creadas un conjunto de carpetas que contienen tanto las librerías a utilizar, así como fuentes, estilos, imágenes y documentos para manejar internacionalización, la aplicación cuenta con una clara estructuración de los diferentes archivos que se manejarán.

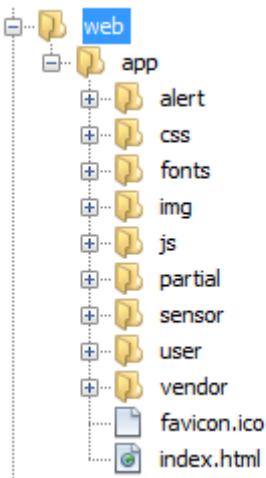


Figura 46. Diagrama estructura proyecto aplicación web cliente.

Fuente: Elaboración propia.

Para la implementación de la página inicial se creó el archivo “index.html” el cual se sitúa en la raíz del proyecto, este archivo utiliza un conjunto de librerías que permiten el desarrollo de aplicaciones web de manera rápida y modular.

```

<html lang="en" ng-app="webApp">
    <head>
        <meta charset="utf-8">
        <link rel="stylesheet" href="css/app.css">
        <link rel="stylesheet" href="css/bootstrap.min.css">
        <link rel="stylesheet" href="css/ng-grid.css">
        <link rel="shortcut icon" href="favicon.ico" type="image/x-icon" />
        <script src="vendor/jquery-2.1.0.min.js"></script>
        <script src="vendor/angular.min.js"></script>
        <script src="vendor/angular-route.min.js"></script>
        <script src="vendor/ng-grid-2.0.11.min.js"></script>
        <script src="vendor/angular-translate.min.js"></script>
        <script src="vendor/bootstrap.min.js"></script>
        <script src="vendor/ng-i18n-0.2.1.js"></script>
        <script type="text/javascript" src="js/main.js"></script>
        <title>Notification System on Real Time</title>
    </head>

```

Figura 47. Librerías utilizadas por la página de inicio.

Fuente: Elaboración propia.

La implementación de la página inicial fue dividida en diferentes archivos en los cuales se realizó la respectiva implementación de cada sección. El archivo “index.html” contiene al archivo encargado de mostrar la información de la cabecera, así como también la sección de contenido.

```

<body>
    <div class="navbar navbar-inverse navbar-fixed-top" role="navigation">
        <div class="container-fluid">
            <div class="navbar-header">
                <button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-collapse">
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                </button>
                <a class="navbar-brand" href="#">J.A.R.V.I.S.</a>
            </div>
            <!-- HEADER -->
            <div ng-include src="'partial/common/header.html'" class="navbar-collapse collapse"></div>
            <!-- /HEADER -->
        </div>
        <div class="container-fluid">
            <div class="row">
                <div ng-include src="'partial/common/signin.html'"></div>
            </div>
        </div>
    </div>
</body>

```

Sección cabecera

Sección contenido

Figura 48. Implementación página de inicio, aplicación web cliente.

Fuente: Elaboración propia.

En la sección cabecera se utiliza el controlador “UserController” el cual es el encargado de manejar toda la lógica concerniente a la información de usuarios. La implementación de la cabecera es considerablemente reducida debido a que los estilos forman parte de las librerías utilizadas.

```
[<form class="navbar-form navbar-right" ng-controller="UserController" ng-submit="loginUser()" action="">
]<div class="form-group">
| <input type="text" ng-model="username" id="username" name="username" placeholder="Email" class="form-control">
|</div>
|<div class="form-group">
| | <input type="password" ng-model="password" id="password" name="password" placeholder="Password" class="form-control">
|</div>
| <button type="submit" class="btn btn-success">Login</button>
| <input type="hidden" name="_csrf" value="{{ _csrf }}>
|</form>
```

Figura 49. Implementación cabecera, página de inicio.

Fuente: Elaboración propia.

En la sección contenido se utiliza para mostrar toda la información referente a la aplicación, como también los controles necesarios para direccionar a los usuarios a la página de creación de cuentas.

En esta iteración el contenido de la página es estático, las imágenes utilizadas son ejemplos de la utilidad que podría tener esta página. La primera versión de la página de inicio es mostrada a continuación.



Figura 50. Diagrama página de inicio, aplicación web cliente.

Fuente: Elaboración propia.

4.5.3.2. Autenticación de usuario

La funcionalidad de autenticación de usuarios se realizará en el archivo “user.js” el cual se encuentra dentro de la carpeta “routes” en el proyecto servicios web, todas las implementaciones relacionadas a los servicios web deben ser realizadas en sus respectivos archivos dentro de la ruta mencionada.

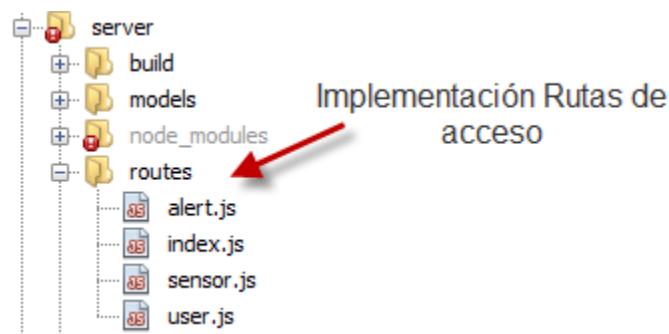


Figura 51. Diagrama de la estructura de la aplicación servicios web.

Fuente: Elaboración propia.

Para la implementación de la ruta de acceso “login” se utilizó el método REST “post” con el cual fue posible completar la implementación, en la cual se verificaron los parámetros enviados hacia esta ruta, así como también se procedió a encriptar la contraseña para realizar la respectiva comparación con la base de datos.

```

// POST login
app.post('/api/user/login', function(req, res) {
  if (!req.param('username') || !req.param('password')) {
    res.send(500, 'invalid-parameters');
    return;
  }

  User.findOne({ email: req.param('username') }, function(err, user) {

    if (err) {
      res.send(500, 'problems-findOne');
      return ;
    }

    if (!user) {
      res.send(500, 'user-not-found');
      return ;
    }

    // Compare password from the form params to the encrypted password of the user found.
    bcrypt.compare(req.param('password'), user.encryptedPassword, function(err, valid) {
      if (err) {
        res.send(500, 'encryptedPassword-failed');
        return ;
      }
    })
  })
})

```

Figura 52. Implementación autenticación usuario, servicios web.

Fuente: Elaboración propia.

Una vez que los datos son validados, el estado del usuario es cambiado a conectado, el servicio web envía una notificación a todos los usuarios, esta notificación permitirá mostrar el estado actual de cada usuario en las aplicaciones cliente, después de enviar las notificaciones una sesión es creada y el resultado de la autenticación es enviado a las aplicaciones cliente.

```

// Change status to online
var token = req.session.id;
User.update({ email: user.email }, { online : true, token: token }, function(err) {
  if (err) {
    res.send(500, 'login-fail-update');
  }

  // Inform other sockets (e.g. connected sockets that are subscribed) that this user is now logged in
  for (var username in app.connections) {
    app.connections[username].emit('connected', user.email);
    app.connections[username].emit('alert', 'User: ' + user.email + 'now is connected');
  }

  // Log user in
  Session.create({
    id: token,
    username: req.param('username')
  }, function(err) {
    if (err) {
      res.send(err);
    }
  });
}

res.send(200, { message: 'access-granted', token: user.token });
}

```

Figura 53. Implementación cambio de estado usuario, aplicación servicios web.

Fuente: Elaboración propia.

Después de realizar la implementación de los servicios necesarios para la autenticación de usuarios, se llevó a cabo la integración con la aplicación web cliente en donde haciendo uso de controladores temporales se realizó la solicitud de autenticación y la respuesta obtenida fue la siguiente.

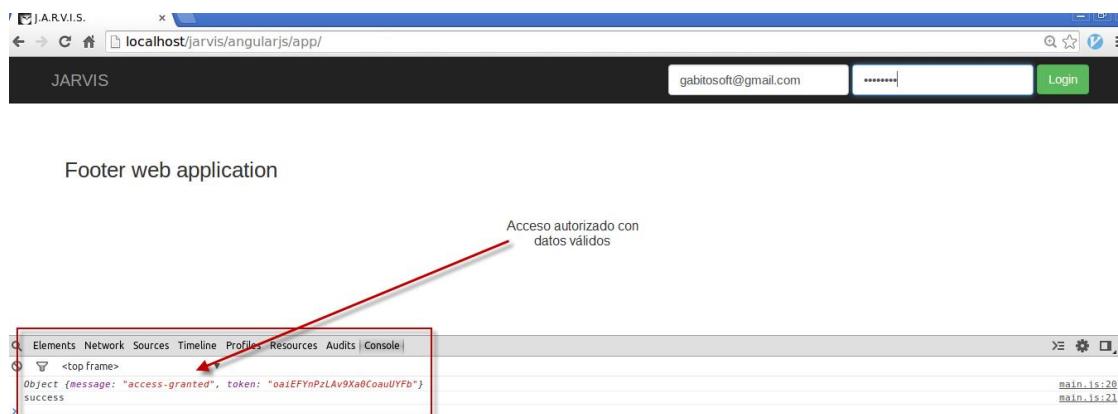


Figura 54. Página web de prueba, autenticación de usuarios.

Fuente: Elaboración propia.

Verificando el correcto funcionamiento de los servicios desarrollados se ingresaron datos inválidos con los cuales se obtuvo un mensaje de error en la aplicación web cliente, de esta forma se pudo verificar que el servicio funciona correctamente.



Figura 55. Pantalla de inicio, con error en la autenticación de usuarios.

Fuente: Elaboración propia.

4.5.3.3. Rutas y estructura aplicación Web Cliente

La navegación de la aplicación web se realizó utilizando rutas definidas en el archivo “main.js” el cual contiene toda la implementación relacionada a la navegación y llenado de información en los controles desde los servicios web.

Cada una de las rutas definidas utiliza una plantilla HTML que a través de los controladores definidas estas plantillas son cargadas con la información solicitada.

```
app.config(function($routeProvider) {
    $routeProvider
        .when('/dashboard', { templateUrl:'../alert/summary.html' })
        .when('/alert', { templateUrl: '../alert/list.html' })
        .when('/sensor', { templateUrl: '../sensor/list.html' })
        .when('/newSensor', { templateUrl: '../sensor/new.html' })
        .when('/setting', { templateUrl: '../setting/index.html' })
        .when('/user', { templateUrl: '../user/list.html' })
        .when('/newUser', { templateUrl: '../user/new.html' })
        .when('/profile', { templateUrl: '../user/profile.html' })
        .when('/help', { templateUrl: '../user/help.html' })
        .otherwise({ redirectTo: '/', templateUrl: '../alert/summary.html' });
});
```

Figura 56. Implementación rutas, aplicación web cliente.

Fuente: Elaboración propia.

La aplicación hace uso de diferentes controladores para enviar y recibir información de los servicios web, por lo que se vio la necesidad de crear los controladores:

- **MainController** representa el controlador principal, este controlador cuenta con los métodos para establecer las rutas definidas, el manejo de los archivos de internacionalización y la representación de datos en las gráficas de la página tablero.
- **UserController** Este controlador manejará la información de las páginas relacionadas a los usuarios ya sea la creación de usuarios, la autenticación, el cargado de los usuarios en el listado, la eliminación y modificación.
- **AlertController** El controlador de alertas será el encargado de trabajar con la información de las alertas, en este controlador se utilizaron métodos para el cargado de alertas, la filtración de alertas por tipo, la modificación y eliminación de las mismas.
- **SensorController** Al igual que anteriores controles este nos permitirá cargar la información de los sensores, modificarlos y eliminarlos.
- **SettingsController** El controlador de las configuraciones permite a los usuarios almacenar los cambios realizados en las configuraciones y cargarlos en la página de configuraciones.
- **PaginationController** Este controlador fue implementado con la finalidad de navegar haciendo uso de un control de paginación entre los distintos elementos listados en las páginas: lista de alertas, lista de usuarios y lista de sensores.

4.5.3.4. Página tablero

El desarrollo de la página tablero está enfocado en la información que el usuario necesita visualizar, por lo que se realizó gran parte del trabajo sobre las librerías gráficas que se usaron para mostrar los datos de las alertas.

La página tablero fue estructurada basándose en el diseño realizado para esta, tomando en cuenta que se utilizó la misma estructura para todas las páginas de la aplicación cliente web. La página cuenta con un menú que permite navegar entre las diferentes páginas así como también los botones que permiten al usuario visualizar el número de alertas nuevas que fueron generados y que no fueron revisadas.

Como mencionamos previamente la implementación para mostrar los datos en las gráficas de esta página fueron realizados en el controlador “**MainController**” en el cual se desarrolló la función “**loadSummaryData**”. En esta función se define el tipo de gráfica que se utilizará en este caso se usó la gráfica de tipo barras, se definieron las categorías en el eje “X” y los datos fueron cargados.

A continuación una porción del código que se utilizó para la implementación de las gráficas, se tomó en cuenta que la gráfica “**alertsByType**” tiene una implementación similar donde fueron cambiados datos como el tipo de gráfica y algunas configuraciones para mostrar la información lo más claramente posible.

```

var alertsbySensor = [];
$http.get('http://localhost:3000/api/alert/querysensor')
.then(function(result) {
  alertsbySensor = result.data;
})

$('#chart-container-left').highcharts({
  chart: {
    type: 'bar'
  },
  title: {
    text: 'Alerts by Sensor'
  },
  xAxis: {
    categories: ['Unknown', 'Information', 'Warning', 'Danger']
  },
  yAxis: {
    title: {
      text: 'Alerts'
    }
  },
  series: alertsbySensor
});
});

```

Se solicitan los datos desde el servicio web, y estos son cargados a la gráfica de alertas por sensor.

Figura 57. Implementación de la función para cargar datos a las gráficas.

Fuente: Elaboración propia.

Finalmente la página tablero fue completada con los estilos definidos en el tema “Jumbo” de la librería Bootstrap la cual facilitó el manejo de estilos, colores y posicionamiento de los diferentes elementos de esta página.



Figura 58. Página tablero, aplicación web cliente.

Fuente: Elaboración propia.

4.5.3.5. Página listar alertas

El desarrollo de la página alertas fue realizado rápidamente debido a que previamente se desarrolló la estructura y se definieron los estilos. La implementación de la función para obtener los datos desde los servicios es mostrada a continuación.

```
$scope.loadAlerts = function() {
    $http.get('http://localhost:3000/api/alert')
        .then(function(result) {
            $scope.alerts = result.data;
        });
};
```

Figura 59. Implementación de la función para cargar alertas.

Fuente: Elaboración propia.

Haciendo uso de los controles provistos por la librería “Bootstrap” y basándose en el diseño de la página para listar alertas esta fue completada, se tomó en cuenta que esta página puede contener grandes cantidades de información por lo que se determinó agregar controles que permitan manejar la paginación de todas las alertas.

Título	Tipo	Fecha	Leído	Descripción
Database	warning	15/06/2014 09:59:56		The capacity of processor has exceeded the limit
127.0.1.1	Memory	28/07/2014 01:41:50		The Memory is almost full
127.0.1.1	Memory	28/07/2014 01:42:24		Server is already update
127.0.1.1	Memory	28/07/2014 01:42:41		Proxy has been updated
127.0.1.1	Memory	28/07/2014 01:43:44		Process on memory has been cleaned

Figura 60. Diagrama página listar alertas, aplicación cliente web.

Fuente: Elaboración propia.

En la implementación de la página se realizó un cambio respecto al diseño, la columna “Read” en lugar de mostrar un texto con su respectivo valor, se utilizó un par de imágenes representando si las alertas ya fueron revisadas.

4.5.3.6. Modificar alertas

Para llevar a cabo la implementación del cuadro de diálogo modificar alertas, fue necesario realizar la respectiva implementación en los servicios web, esta implementación fue realizada de la misma forma que la funcionalidad para modificar la información de usuarios, pero en este caso utilizamos la estructura del documento alerta.

```
//UPDATE Alert
app.put('/api/alert/:id', function(req, res) {
  if (!req.body._id) {
    res.send(500, 'error-update-alert: key not found');
  }

  Alert.update({ _id: req.body._id }, {
    source: req.body.source,
    date: req.body.date,
    type: req.body.type,
    title: req.body.title,
    description: req.body.description,
    read: req.body.read
  },function (error) {
    if (error) {
      res.send(500, 'error-update-alert' + error);
    }

    res.send(200, 'alert-updated');
  });
});
```

Figura 61. Implementación servicio modificar alertas.

Fuente: Elaboración propia.

Una vez realizada la implementación del servicio de modificación de alertas, la integración con la aplicación web cliente fue rápida y basándose en el diseño del cuadro de dialogo previamente realizado se completó el desarrollo de esta funcionalidad, cabe mencionar que las alertas únicamente pueden recibir cambios del campo “Read” el cual determina si una alerta fue visualizada por el usuario, es debido a este comportamiento que los campos en cuadro de dialogo fueron deshabilitados para realizar ediciones exceptuando el campo “Read”, el resultado obtenido fue el siguiente.

Id: [53e0f7400ab143bd0b20d0e7] ×

Origen	<input type="text" value="127.0.1.1"/>
Tipo	<input type="text" value="warning"/>
Título	<input type="text" value="Test10"/>
Fecha	<input type="text" value="05/08/2014 11:24:48"/>
Descripción	<input type="text" value="Testing data 10"/>
Leído	<input checked="" type="checkbox"/>

Cerrar
Guardar

Figura 62. Diagrama cuadro de dialogo editar Alerta, aplicación web.

Fuente: Elaboración propia.

4.5.3.7. Eliminar alertas

La implementación del servicio para eliminar alertas en el lado del servidor fue realizada haciendo uso del método DELETE el cual forma parte de la definición REST.

De forma similar a previas implementaciones de servicios, la implementación del servicio eliminar hace uso de un parámetro enviado, en este caso el parámetro representa el identificador de la alerta el cual es único, en base a este parámetro se realiza la búsqueda de la alerta indicada y en el caso de que no exista, un mensaje con el código y estado es enviado a las aplicaciones cliente, caso contrario se hace uso del método “remove” en el objeto Alerta el cual se encargará de eliminar la alerta de la base datos y finalmente enviar un mensaje a las aplicaciones web cliente indicando que la alerta fue eliminada.

```

//DELETE Alert
app.delete('/api/alert/:id', function(req, res) {
  Alert.remove({
    _id: req.params.id
  }, function(err, alert){
    if (err){
      res.send(500, 'error-delete-alert' + err);
    }
    res.send(200, 'alert-deleted');
  });
});

```

Figura 63. Implementación eliminar alertas, servicio web.

Fuente: Elaboración propia.

La implementación del lado de la aplicación cliente se simplificó a realizar la llamada al servicio de eliminación de alertas, enfocando la mayor parte del trabajo en ofrecer una grata experiencia de usuario cuando una alerta es eliminada.

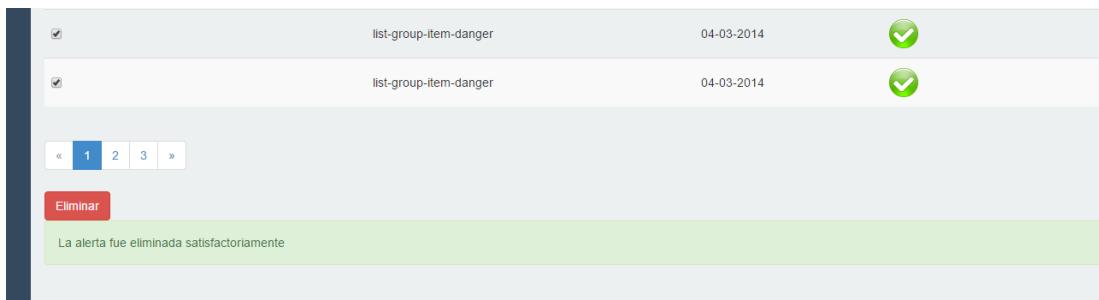


Figura 64. Diagrama eliminar alertas, aplicación web cliente.

Fuente: Elaboración propia.

4.5.3.8. Página configuración

En esta iteración el desarrollo de la página configuración fue enfocado en el lado de la aplicación cliente, utilizando información estática para que sea utilizada por los controles, las diferentes secciones de la página fueron claramente diferenciadas utilizando colores característicos de cada tipo de alertas, y manteniendo el mismo esquema de páginas que fueron desarrolladas en anteriores iteraciones se logró completar rápidamente el desarrollo de esta página.

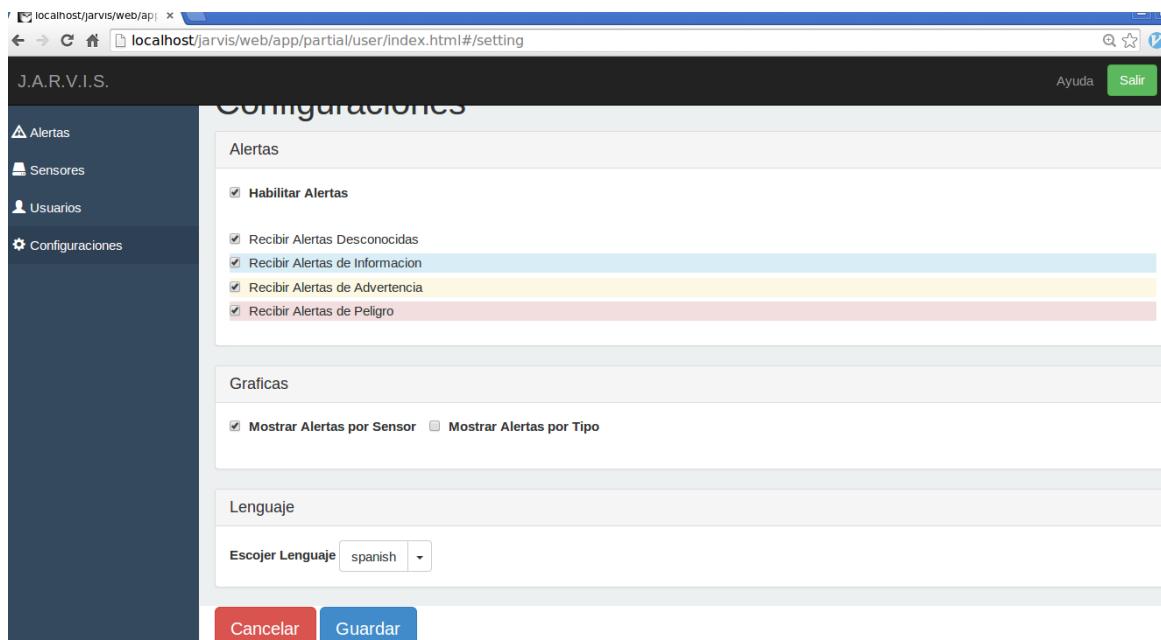


Figura 65. Diagrama página configuración, aplicación cliente web.

Fuente: Elaboración propia.

En posteriores iteraciones la página configuraciones permitirá a los usuarios filtrar el tipo de alertas que desean recibir, cambiar el idioma de la aplicación y mostrar las gráficas de alertas.

4.5.4. Pruebas de Funcionalidad

A continuación la tabla de las pruebas realizadas para verificar el funcionamiento de la aplicación web cliente, cada una de las pruebas cuenta con sus respectivos resultados.

Prueba	Resultado Esperado	Resultado Obtenido	Estado
<p>Verificar el servicio web de autenticación.</p> <p>Haciendo uso de una herramienta “REST client”, con la URL http://localhost:3000/api/user/login con los parámetros: username:gabitosoft@gmail.com password:mirmidon</p> <p>Y usando el método POST.</p> <p>Verificar que el código del resultado.</p>	El resultado de la invocación del servicio web de autenticación es 200 OK .	El resultado de la invocación del servicio web de autenticación es 200 OK .	PASÓ
<p>Verificar el servicio web de listado de alertas.</p> <p>Haciendo uso de una herramienta “REST client” , con la URL http://localhost:3000/api/alert</p> <p>Y usando el método POST</p> <p>Verificar el código del resultado y el contenido de la respuesta.</p>	El resultado de la invocación del servicio web para listar alertas es 200 OK , un conjunto de objetos JSON son mostrados dependiendo del número de alertas.	El resultado de la invocación del servicio web para listar alertas es 200 OK , un conjunto de objetos JSON son mostrados dependiendo del número de alertas.	PASÓ
<p>Verificar el servicio web de modificación de alertas.</p> <p>Haciendo uso de una herramienta “REST client” , con la URL http://localhost:3000/api/alert</p> <p>con los parámetros</p> <p><code>_id: '539da6dc7e3573110a280594'</code> <code>source: '127.0.0.1'</code>, <code>date: '09/11/2014 19:00:00'</code>, <code>type: 'danger'</code>, <code>title: 'Test'</code>, <code>description: 'Test description'</code>, <code>read: 'true'</code></p> <p>y usando el método PUT</p> <p>Verificar el código del resultado.</p>	El resultado de la invocación del servicio web de modificación de alertas es 200 OK .	El resultado de la invocación del servicio web de modificación de alertas es 200 OK .	PASÓ
<p>Verificar el servicio web de eliminación de alertas.</p> <p>Haciendo uso de una herramienta</p>	El resultado de la invocación del servicio web de eliminación de	El resultado de la invocación del servicio web de eliminación de	PASÓ

“REST client” , con la URL <code>http://localhost:3000/api/alert</code> con el parámetro <code>_id: ‘539da6dc7e3573110a280594’</code> y usando el método DELETE Verificar el código del resultado.	alertas es 200 OK.	alertas es 200 OK.	
--	---------------------------	---------------------------	--

Tabla 18. Pruebas de funcionalidad, iteración 5.

Fuente: Elaboración propia.

4.5.5. Pruebas de Integración

Las pruebas de integración llevadas a cabo en esta iteración tuvieron como enfoque realizar llamadas a los servicios implementados desde la aplicación web cliente, en donde se pudo constatar que la aplicación web cliente funciona correctamente haciendo uso de los servicios web.

Prueba	Resultado Esperado	Resultado Obtenido	Estado
Autenticación. Utilizando el navegador Google Chrome dirigirse a la página: http://localhost/jarvis en la página ingresar los credenciales usuario: gabitossoft@gmail.com contraseña: mirmidon Haciendo uso de la consola web provista por el navegador verificar el resultado.	La página tablero es mostrada, el nombre de usuario es mostrado en la cabecera de la página.	La página tablero es mostrada, el nombre de usuario es mostrado en la cabecera de la página.	PASÓ
Navegación listar alertas. Haciendo uso del menú lateral navegar hacia la página listar alertas.	La página de lista de alertas es mostrada y cada fila cuenta con el control “checkbox” para seleccionar la alerta.	La página de lista de alertas es mostrada y cada fila cuenta con el control “checkbox” para seleccionar la alerta.	PASÓ
Navegación listar sensores. Haciendo uso del menú lateral navegar hacia la página listar sensores.	La página de lista de sensores es mostrada y cada fila cuenta con el control “checkbox” para seleccionar cada sensor.	Utilizando datos estáticos se pudo constatar que los sensores son mostrados en la lista y cada uno cuenta con su respectivo control de selección.	PASÓ
Navegación listar usuarios. Haciendo uso del menú lateral navegar hacia la página listar usuarios.	La página de lista de usuarios es mostrada y cada fila cuenta con el control “checkbox” para seleccionar cada usuario.	Utilizando datos estáticos se pudo constatar que los usuarios son mostrados en la lista y cada uno cuenta con su respectivo control de selección.	PASÓ

Tabla 19. Pruebas de integración, iteración 5.

Fuente: Elaboración propia.

4.5.6. Cuadro de Avance

Código	Estimación	Duración	Estado
H24 (Página de inicio.)	8	2	Hecho
H25 (Servicio de autenticación integrado a la aplicación web cliente.)	8	2	Hecho
H26 (Rutas y estructura de la aplicación web cliente.)	13	3	Hecho
H27 (Página tablero.)	13	3	Hecho
H28 (Página listar alertas.)	8	1	Hecho
H29 (Servicio Eliminar alertas.)	8	1	Hecho
H30 (Servicio Modificar alertas.)	8	1	Hecho
H31 (Página Configuraciones)	8	1	Hecho

Tabla 20. Cuadro de avance, iteración 5.

Fuente: Elaboración propia.

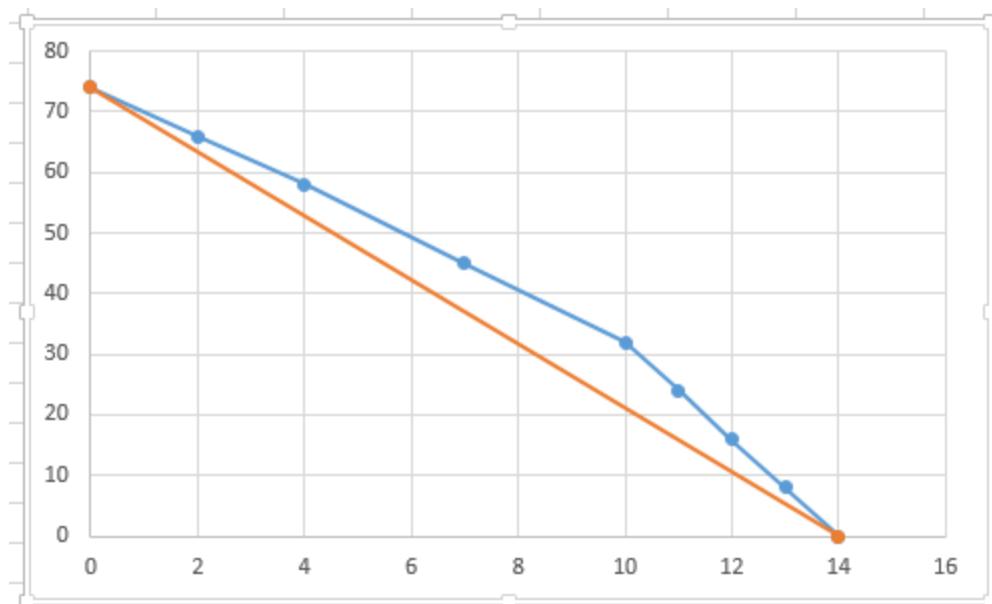


Figura 66. Diagrama de avance de historias de usuario iteración 5.

Fuente: Elaboración propia.

4.5.7. Retrospectiva

Bueno	Malo	Mejorar
<ul style="list-style-type: none"> - El avance respecto a la implementación de la aplicación web ya fue cubierto en gran parte. - El hecho de utilizar la librería Bootstrap para el manejo de estilos y colores facilitó la implementación del proyecto la elección de esta librería fue acertada. 	<ul style="list-style-type: none"> - No se tomaron en cuenta las pruebas al momento de realizar la estimación. - A pesar de haber completado el conjunto de historias de usuario para esta iteración, el tiempo que tomó completarlas no se acercó al estimado. 	<ul style="list-style-type: none"> - Se debe tomar en cuenta las pruebas para futuras estimaciones.

Tabla 21. Retrospectiva, iteración 5.

Fuente: Elaboración propia.

4.6. Iteración 6 Eridanus

Para esta iteración se trabajarán los requerimientos relacionados al desarrollo de una aplicación móvil que pueda conectarse a los servicios web implementados previamente.

A continuación las historias de usuario de esta iteración estarán enfocadas en el desarrollo de la aplicación móvil cliente enfocando su diseño e implementación para plataformas Android, también se trabajará en implementaciones para obtener estilos adecuados para este tipo de aplicaciones.

4.6.1. Cuadro de Historias de Usuario

Código	Título	Descripción	Tipo	Estimación	Prioridad
H32	Página de autenticación para dispositivos móviles (Diseño).	<p>La aplicación móvil necesita una primera página de autenticación donde los usuarios ingresen sus credenciales para acceder a la aplicación.</p> <p>CA:</p> <p>Como usuario de la aplicación es necesario contar con los controles adecuados para ingresar la dirección de correo electrónico, la contraseña, la URL del servidor proveedor del servicio web y el número de puerto.</p>	Diseño	5	1
H33	Página de autenticación para dispositivos móviles (Implementación).	<p>Con el diseño de la página de autenticación llevar a cabo la implementación.</p> <p>CA:</p> <p>La página de autenticación debe permitir a los usuarios acceder a la aplicación y visualizar la página de alertas, en caso de que las credenciales sean incorrectas se debe mostrar un mensaje, solicitando el ingreso de nuevas credenciales</p>	Desarrollo	8	1

H34	Página de listar alertas para dispositivos móviles (Diseño).	<p>La página de listado de alertas será la encargada de mostrar a los usuarios las alertas generadas.</p> <p>CA:</p> <p>La página de listado de alertas debe contener un conjunto de controles de tipo botón con los nombres de cada tipo de alerta.</p> <p>La página debe contar con un control de tipo lista que permita mostrar las alertas en una sola columna.</p> <p>La página debe contar con un conjunto de controles tipo botón los cuales serán utilizados para salir de la aplicación y navegar hacia la página de configuraciones.</p>	Diseño	5	2
H35	Página alertas para dispositivos móviles.	<p>Con el diseño de la página de listado de alertas, llevar a cabo la implementación de la misma.</p> <p>CA:</p> <p>La página listar alertas debe permitir a los usuarios filtrar las alertas por tipo.</p> <p>Las alertas generadas deben aparecer en orden ascendente, es decir las más antiguas deberán estar al final de la lista.</p> <p>Desde la página listar alertas debe ser posible navegar hacia otras páginas de la aplicación.</p>	Desarrollo	13	2
H36	Pruebas de integración y funcionalidad de la página de autenticación de usuarios, utilizando el servicio Web.	<p>Llevar a cabo la siguiente prueba de funcionalidad e integración de la página de autenticación de usuarios.</p> <p>CA:</p> <p>Verificar que la aplicación permita a usuarios registrados en la base de datos acceder a la aplicación.</p>	Prueba	2	3

H37	<p>Pruebas de integración y funcionalidad de la página alertas, utilizando el servicio Web.</p>	<p>Llevar a cabo la siguiente prueba de funcionalidad e integración de la página de listado de alertas.</p> <p>CA:</p> <p>Verificar que en la página listar alertas permita a los usuarios visualizar las nuevas alertas cuando estas son generadas por el servicio web.</p>	Prueba	2	3
-----	---	---	--------	---	---

Tabla 22. Cuadro de historias de usuario, iteración 6.

Fuente: Elaboración propia.

4.6.2. Diseño

En esta iteración se trabajó en el diseño de un par de páginas que formarán parte de la aplicación móvil cliente, en cada uno de los diseños realizados se utilizó la misma estructura, brindando una experiencia de usuario sencilla y agradable. A continuación se detalla el trabajo realizado en cada uno de los diseños.

4.6.2.1. Página de autenticación

El diseño de la página de autenticación de usuarios cuenta con los controles necesarios para que los usuarios puedan autenticarse, dado que esta página forma parte de una aplicación móvil se utilizaron elementos enfocados en este tipo de aplicaciones.



Figura 67. Diagrama página autenticación usuarios, aplicación móvil.

Fuente: Elaboración propia.

La página contendrá una imagen en la cabecera representando el logotipo de la aplicación, además de contar con los controles de tipo texto que servirán para introducir la URL de la aplicación de servicios web, el puerto por el cual se comunicarán, el nombre de usuario y la contraseña del mismo.

4.6.2.2. Página listar alertas

La página de listado de alertas fue diseñada bajo el mismo esquema que la página de autenticación, en este diseño podemos resaltar que los controles del tipo botón aparecen en la cabecera de la aplicación y en el pie de página, pero se debe tomar en cuenta que los botones de la parte superior de la página solo serán mostrados para la página de listado de alertas.

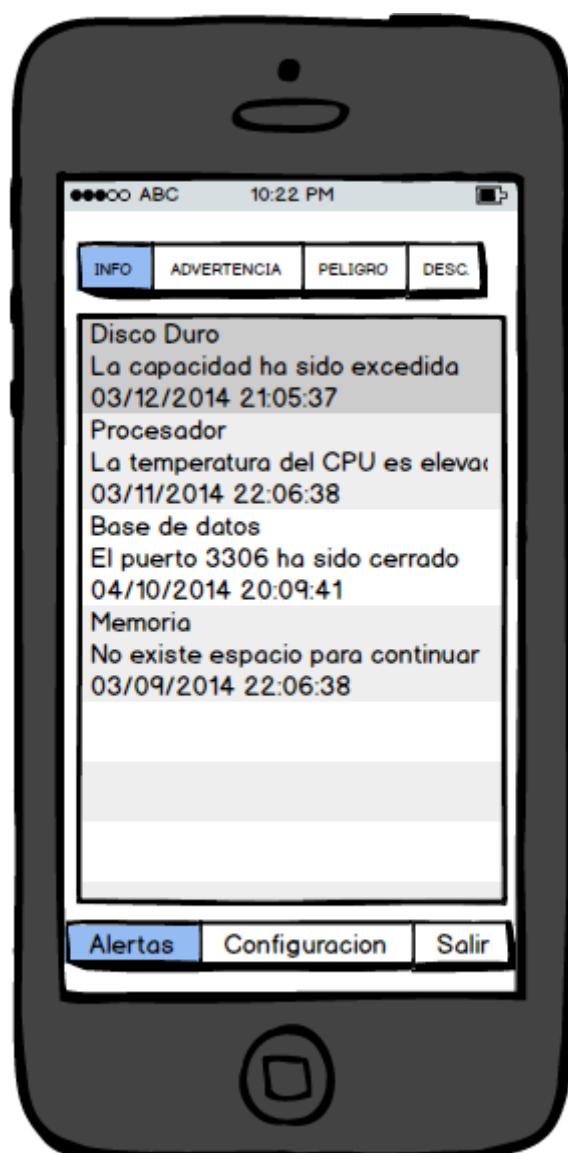


Figura 68. Diagrama página listado de alertas, aplicación móvil.

Fuente: Elaboración propia.

En el diseño de la lista de alertas podemos notar que cada elemento de la lista contendrá 3 líneas de texto, la primera de ellas representará el título de la alerta, seguida por una corta descripción y como última línea estará la fecha con el formato especificado.

4.6.3. Implementación

Para llevar a cabo la implementación de la aplicación móvil se comenzó por preparar el entorno en el cual la aplicación funcionaría, como fue mencionado anteriormente la aplicación fue pensada para dispositivo Android y dado que el desarrollo de la aplicación se basará en páginas web se optó por utilizar el Framework Phonegap el cual permite generar aplicaciones para sistemas operativos como Android en base a aplicaciones web. Después de crear el proyecto para la aplicación móvil se obtuvo la siguiente estructura de directorios.

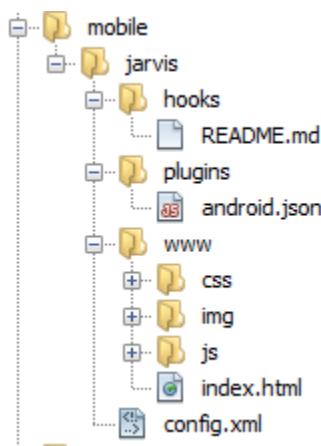


Figura 69. Gráfica estructura proyecto aplicación móvil.

Fuente: Elaboración propia.

El contenido de cada una de estas carpetas es utilizado por el Framework para llevar a cabo la conversión de nuestra aplicación web a aplicación para dispositivos Android, dado que solo necesitamos trabajar con la aplicación web no enfocaremos en el contenido de la carpeta “www” en la cual son almacenados archivos con páginas web, estilos y funciones en Java Script.

Dentro de la carpeta “www” encontraremos los directorios: “audio”, “css”, “img” y “js” los que utilizaremos para agregar los archivos necesarios en los respectivos directorios, también encontraremos el archivo “index.html” que al igual que en nuestra aplicación web cliente este archivo será el punto de acceso a la aplicación móvil.

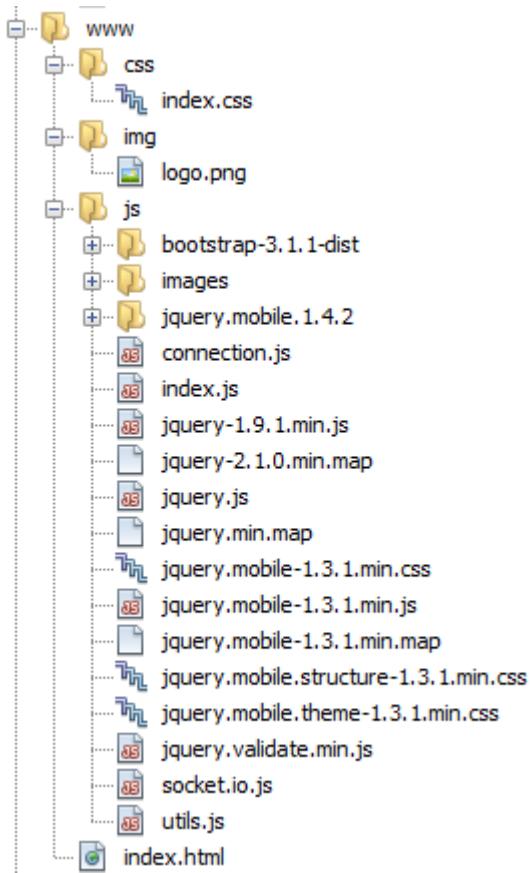


Figura 70. Diagrama estructura folder "www".

Fuente: Elaboración propia.

Dado que la aplicación web cliente no tiene la misma estructura de diseño que la aplicación móvil y las librerías utilizadas en la aplicación web cliente no fueron completamente adaptadas a dispositivos móviles se procedió a utilizar la librería “JQuery Mobile” la cual tiene un alto grado de adaptabilidad a dispositivos móviles, también ofrece una gran cantidad de controles y elementos que ayudan a la implementación de este tipo de aplicaciones. El archivo “index.html” contiene a todas las páginas que fueron creadas para la aplicación, este archivo hace uso de características de HTML5 y basándose en los diseños realizados se creó cada una de las páginas.

4.6.3.1. Página autenticación usuarios

Para la página de autenticación de usuarios se utilizaron estilos provistos por la librería JQuery Mobile, los cuales permiten que la aplicación tenga una interfaz de usuario uniforme tanto en navegadores como en aplicaciones hibridas. En esta porción de código dentro de la sección artículo se insertaron los controles del tipo texto los cuales son utilizados para la autenticación de usuarios.

```

<!-- Login -->
| <div id="page-login" data-role="page" data-fullscreen="true">
|   <header data-role="header">
|     <h1>J.A.R.V.I.S.</h1>
|   </header><!-- /header -->
|   <article data-role="content">
|     <form id="check-user" class="ui-body ui-body-b ui-corner-all container-form-login" data-ajax="false">
|       <fieldset class="fieldset-form-login">
|         <div class="ui-grid-a grid-form-login">
|           <div class="ui-block-a">
|             
|             <input type="text" id="server" class="text-field-form" placeholder="Server" value="192.168.1.117" data-theme="a" data-role="input" data-type="text" data-value="192.168.1.117"/>
|             <input type="text" id="port" class="text-field-form" placeholder="Port" value="3000" data-theme="a" data-role="input" data-type="text" data-value="3000"/>
|             <input type="email" id="username" class="text-field-form" placeholder="Username" value="gabriel" data-theme="a" data-role="input" data-type="text" data-value="gabriel"/>
|             <input type="password" id="password" class="text-field-form" placeholder="Password" value="gabriel" data-theme="a" data-role="input" data-type="password" data-value="gabriel"/>
|             <a href="#" id="login" data-role="button" class="buttons-forms" data-theme="a">Login</a>
|           </div>
|         </div>
|       </fieldset>
|     </form>
|   </article><!-- /content -->
|   <footer data-role="footer" data-fullscreen="true" data-position="fixed">
|     <a href="#help" data-role="button" data-icon="info" data-inline="true" class="buttons-forms">Help</a>
|     <a href="#about" data-role="button" data-icon="info" data-inline="true" class="buttons-forms">About</a>
|   </footer><!-- /footer -->
| </div><!-- /page -->

```

Campos para ingresar credenciales

Figura 71. Implementación página autenticación usuarios, aplicación móvil.

Fuente: Elaboración propia.

Después de la implementación de la página de autenticación se procedió a utilizar herramientas propias del Framework para generar el ejecutable para dispositivos móviles, estas herramientas son utilizadas desde línea de comandos y permiten generar un archivo que podrá ser ejecutado en nuestros dispositivos móviles virtuales.

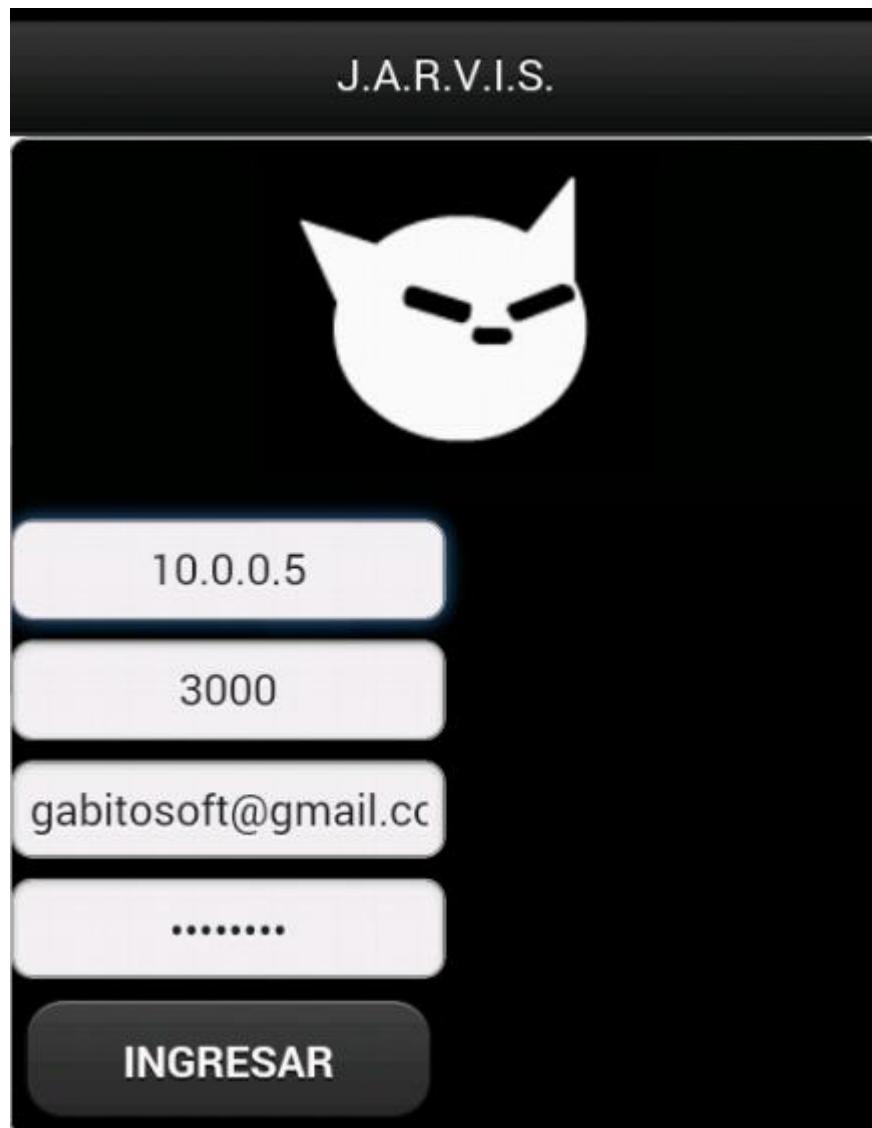


Figura 72. Diagrama página de autenticación, aplicación móvil.

Fuente: Elaboración propia.

En la página fueron implementados todos los controles mencionados en los requerimientos de usuario, dando forma a la página mostrada en la imagen anterior, también cabe mencionar que se manejó la estructura: contenedor, cabecera, artículo y pie de página para tener una clara diferenciación de cada sección de la página.

La funcionalidad para autenticar usuarios con información de los controles de texto fue desarrollada en el archivo “utils.js” en donde se realizó la conexión con el servicio web, se definieron las funciones para representar el estado de espera y las funciones donde se determina como proceder en caso de que los datos sean correctos o en el caso de la conexión no se lleve a cabo, a continuación una porción de la función encargada de realizar la autenticación.

```

$.ajax({
    url: 'http://' + server + ':' + port + '/api/user/login',
    data: { username : usr, password: pwd },
    type: 'post',
    dataType: 'json',

    beforeSend: function() {
        // This callback function will trigger before data is sent
        $.mobile.loading( 'show' );
    },
    complete: function() {
        // This callback function will trigger on data sent/received complete
        $.mobile.loading( 'hide' );
    },
    success: function (request, result) {
        window.location.href = '#alerts';
        init();
    },
    error: function (request, error) {
        // This callback function will trigger on unsuccessful action
        alert('Network error has occurred please try again');
    }
});

```


Si la autenticación fue realizada, la página listar alertas será mostrada

Figura 73. Implementación autenticación usuarios, aplicación móvil.

Fuente: Elaboración propia.

4.6.3.2. Página listar alertas

Al igual que la página de autenticación, la página de alertas fue implementada dentro del archivo “index.html” en una sección diferente, donde todos los controles y elementos necesarios para esta página fueron insertados.

Para esta página también se utilizó la estructura contenedor, cabecera, artículo y pie de página, pero en esta ocasión se insertó en el pie de página un control para navegar entre las diferentes páginas de la aplicación la cual a su vez contiene los botones: alertas, configuración y salir, estos botones no cuentan con toda la implementación necesaria, específicamente el botón configuración que no tiene ninguna acción asociada.

Haciendo uso del control “list-view” fue posible crear una lista que reciba elementos dinámicamente, este control inicialmente se encuentra vacía.

```

<!-- Alerts -->
<div data-role="page" id="alerts" >
  <div data-role="header">
    <h1>Alerts</h1>
  </div>
  <article data-role="content">
    <div data-role="controlgroup" data-type="horizontal" class="btn-group">
      <a data-role="button" data-mini="true" data-theme="a" class="ui-btn-inline buttons-forms">Info</a>
      <a data-role="button" data-mini="true" data-theme="a" class="ui-btn-inline buttons-forms">Warning</a>
      <a data-role="button" data-mini="true" data-theme="a" class="ui-btn-inline buttons-forms">Danger</a>
      <a data-role="button" data-mini="true" data-theme="a" class="ui-btn-inline buttons-forms">Unknown</a>
    </div>
    <div class="panel-alerts">
      <ul data-role="listview" class="list-group" id="alertList">
        <!-- Alerts will be added here -->
      </ul>
    </div>
  </article>
  Lista de alertas
<footer data-role="footer" data-fullscreen="true" data-position="fixed">
  <div data-role="navbar" data-iconpos="top">
    <ul>
      <li><a class="buttons-forms" href="#alerts" data-icon="alert" >Alerts</a></li>
      <li><a class="buttons-forms" href="#settings" data-icon="gear" >Settings</a></li>
      <li><a class="buttons-forms" data-role="button" data-icon="delete" >Exit</a></li>
    </ul>
  </div><!-- /navbar -->
</footer><!-- /footer -->
</div>
<!-- /Alerts -->

```

Figura 74. Implementación página autenticación usuarios, aplicación móvil.

Fuente: Elaboración propia.

La implementación para recibir alertas en tiempo real fue implementada en el archivo “connections.js” donde se hizo uso de la librería “SocketIO” para establecer conexión con la aplicación web de servicios.

```

loadScript('js/socket.io.js', function () {
  var socket = io.connect($('#server').val() + ':' + $('#port').val());
  socket.on('alert', function (data) {
    $('#alertList').append('<li class="alert-item list-group-item-' + data.type +
      '<h3 class="list-group-item-heading">' + data.title + '</h3>' +
      '<p class="list-group-item-text">' + data.description + '</p>' +
      '<p class="list-group-item-text">' + data.date + '</p>' +
      '</a></li>');
    socket.emit('message', { message: 'Client was notified' });

    window.plugin.notification.local.add({
      id: data._id,
      title: data.title,
      message: data.description,
      autoCancel: true,
      sound: 'android.resource://sounds-767-arpeggio'
    });
  });

  socket.on('connect', function () {
    $('#status').html('Connected');
    socket.emit('username', $('#username').val());
  });
});

```

Figura 75. Implementación recepción alertas, aplicación móvil.

Fuente: Elaboración propia.

Después de generar el archivo de ejecución nuestra aplicación ya es capaz de recibir alertas desde el servicio web, las cuales son mostradas con colores representativos dependiendo del tipo de alerta recibida.

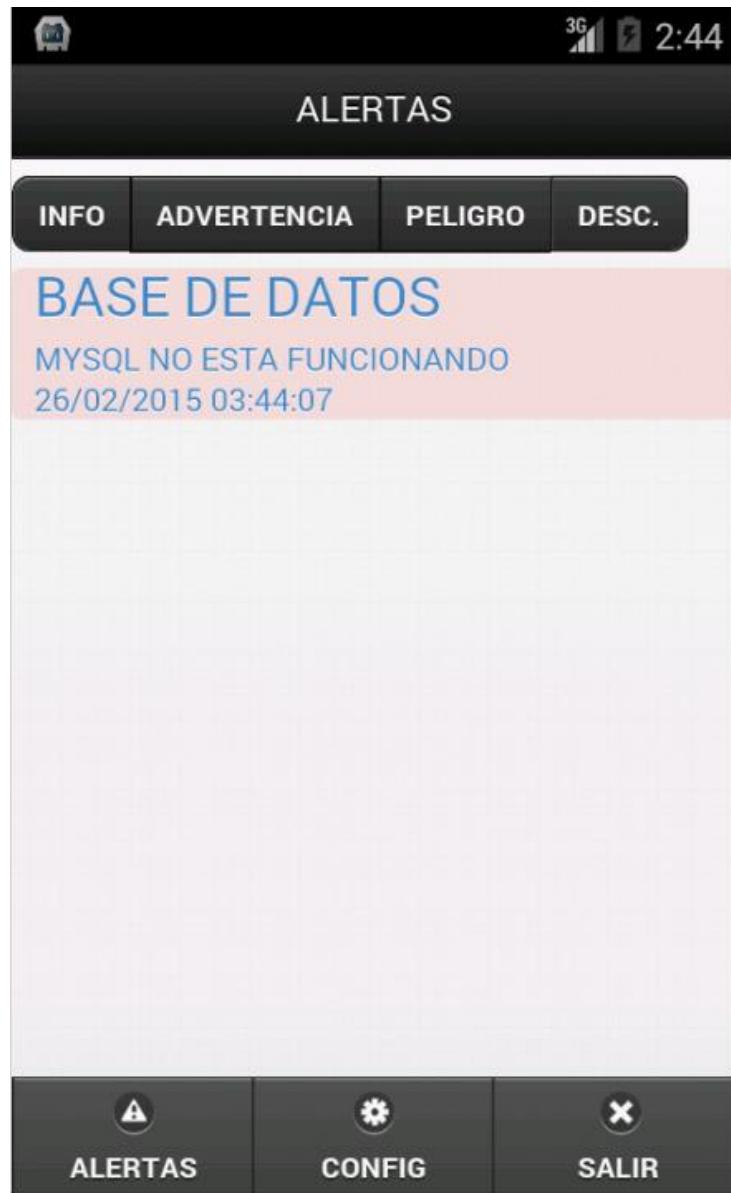


Figura 76. Diagrama página listado de alertas, aplicación móvil.

Fuente: Elaboración propia.

Los botones ubicados en el pie de página cuentan con la funcionalidad de navegar hacia otras páginas en este caso solo se cuenta con la página de listar alertas.

4.6.4. Pruebas de Funcionalidad

A continuación la tabla de pruebas realizadas para verificar el funcionamiento de la aplicación móvil cliente, cada una de las pruebas cuenta con sus respectivos resultados.

Prueba	Resultado Esperado	Resultado Obtenido	Estado
Ejecución en dispositivos Android. Una vez que la aplicación fue instalada en un dispositivo Android, ejecutar la aplicación móvil.	Después de ejecutar la aplicación la página de autenticación es mostrada.	Después de ejecutar la aplicación la página de autenticación es mostrada.	PASÓ
Mostrar notificaciones visuales en segundo plano. Después de ejecutar la aplicación móvil, llevar está a segundo plano y generando alertas, verificar que la aplicación genera algún tipo de notificación visual en el dispositivo.	Cuando se generan alertas y la aplicación se encuentra en segundo plano, en la barra de estado del dispositivo se muestra el título de la alerta.	Cuando se generan alertas y la aplicación se encuentra en segundo plano, en la barra de estado del dispositivo se muestra el título de la alerta.	PASÓ
Generar notificaciones sonoras en segundo plano. Después de ejecutar la aplicación móvil, llevar está a segundo plano y generando alertas, verificar que la aplicación genera algún tipo de notificación sonora en el dispositivo.	Cuando se generan alertas y la aplicación se encuentra en segundo plano, el tono “sounds-767-arpeggio.mp3” es reproducido.	No se reproduce ningún tipo de sonido.	NO PASÓ
Navegación. Haciendo uso de los botones en la parte inferior de la aplicación verificar que estos permiten a los usuarios navegar entre las distintas páginas de la aplicación.	La página de configuraciones es mostrada cuando seleccionamos el botón configuración, la página alertas es mostrada cuando seleccionamos el botón alertas y la aplicación es cerrada cuando presionamos el botón salir.	La página de configuraciones es mostrada cuando seleccionamos el botón configuración, la página alertas es mostrada cuando seleccionamos el botón alertas y la aplicación es cerrada cuando presionamos el botón salir.	PASÓ

Tabla 23. Pruebas de funcionalidad, iteración 6.

Fuente: Elaboración propia.

4.6.5. Pruebas de Integración

Las pruebas de integración llevadas a cabo en esta iteración tuvieron como enfoque realizar llamadas a los servicios implementados desde la aplicación cliente móvil, en donde se pudo constatar que la aplicación móvil cliente funciona correctamente haciendo uso de los servicios web.

Prueba	Resultado Esperado	Resultado Obtenido	Estado
Autenticación. En la página de autenticación ingresar los credenciales URL: 192.168.150.1 puerto: 3000 usuario: gabitosoft@gmail.com contraseña: mirmidon	La página de listado de alertas es mostrada.	La página de listado de alertas es mostrada.	PASÓ
Recibir alertas. Con la aplicación móvil ejecutándose generar alertas y verificar que estas son mostradas en la página de listado de alertas.	Las alertas generadas son mostradas en la página listar alertas, apareciendo en orden ascendente respecto a la fecha en la que fueron generadas.	Las alertas generadas son mostradas en la página listar alertas, apareciendo en orden ascendente respecto a la fecha en la que fueron generadas.	PASÓ
Cambio de estado usuario. Haciendo uso de una herramienta para visualizar base de datos no relaciones y con las credenciales del usuario de prueba, verificar que cuando la aplicación móvil es cerrada el usuario cambio su estado de conexión a false. Para llevar a cabo la prueba el usuario únicamente debe estar conectado desde el dispositivo móvil.	El campo “online” en el documento usuario es cambiado a “false”, cuando la aplicación es cerrada.	El campo “online” en el documento usuario es cambiado a “false”, cuando la aplicación es cerrada.	PASÓ

Tabla 24. Pruebas de integración, iteración 6.

Fuente: Elaboración propia.

4.6.6. Cuadro de Avance

Código	Estimación	Duración	Estado
H32 (Página de autenticación para dispositivos móviles. (Diseño))	5	1	Hecho
H33 (Página de autenticación para dispositivos móviles. (Implementación))	8	4	Hecho
H34 (Página de listar alertas para dispositivos móviles.)	5	1	Hecho
H35 (Página alertas para dispositivos móviles. (Implementación))	13	6	Hecho
H36 (Pruebas de integración y funcionalidad de la página de autenticación de usuarios, utilizando el servicio Web.)	2	1	Hecho
H37(Pruebas de integración y funcionalidad de la página alertas, utilizando el servicio Web)	2	1	Hecho

Tabla 25. Cuadro de avance, iteración 6.

Fuente: Elaboración propia.

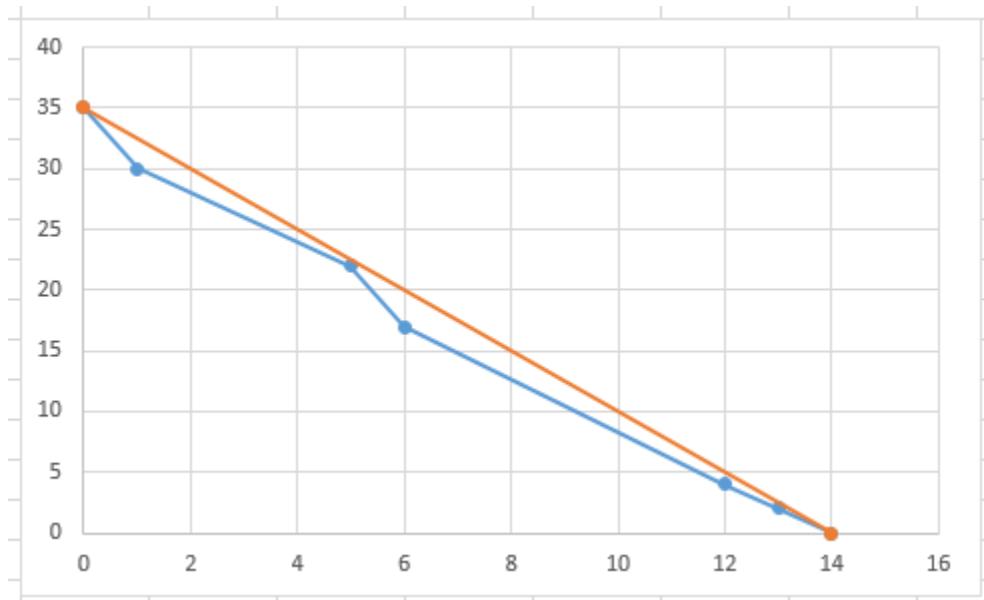


Figura 77. Diagrama de avance iteración 6.

Fuente: Elaboración propia.

4.6.7. Retrospectiva

Bueno	Malo	Mejorar
<ul style="list-style-type: none"> - Las historias de usuarios fueron completadas en el tiempo previsto. - En esta iteración fueron tomadas en cuenta las pruebas funcionalidad e integración. 	<ul style="list-style-type: none"> - Se invirtió bastante tiempo en la implementación de una historia de usuario, quizá una mayor fragmentación en la historia ayudaría a trabajar en más historias y llevarlas a cabo en menos tiempo. 	<ul style="list-style-type: none"> - Se debe intentar realizar historias de usuario más atómicas.

Tabla 26. Retrospectiva iteración 6.

Fuente: Elaboración propia.

4.7. Iteración 7 Centaurus

En esta iteración se trabajó en completar las funcionalidades adicionales de cada una de las respectivas aplicaciones, la implementación de cada una de las historias de usuario realizadas en anteriores iteraciones permitió que el trabajo realizado en esta iteración tienda a ser similar lo cual ayudo de gran manera a realizar un rápido desarrollo en esta iteración.

4.7.1. Cuadro de Historias de Usuario

Código	Título	Descripción	Tipo	Estimación	Prioridad
H38	Listar, crear, modificar y eliminar sensores.	<p>Haciendo uso de implementaciones realizadas para alertas, utilizar las mismas como base e implementar las funciones que permitan listar, crear, modificar y eliminar sensores.</p> <p>CA:</p> <p>Desde la aplicación web cliente un usuario autenticado puede listar sensores, crear sensores, modificar la información de los sensores y eliminarlos.</p>	Desarrollo	8	1
H39	Listar, crear, modificar y eliminar usuarios.	<p>Haciendo uso de implementaciones realizadas para alertas, utilizar las mismas como base e implementar las funciones que permitan listar, crear, modificar y eliminar usuarios.</p> <p>CA:</p> <p>Desde la aplicación web cliente un usuario autenticado puede listar usuarios, crear usuarios, modificar la información de los usuarios y eliminarlos.</p>	Desarrollo	5	1

H40	Página de configuraciones.	<p>La página configuraciones debe permitir a los usuarios almacenar los cambios realizados en esta página.</p> <p>CA:</p> <p>Como usuario los cambios realizados para cambiar el idioma, las alertas visibles y los gráficos de alertas, la aplicación web cliente debe permitir almacenar estos cambios, por lo tanto si un usuario cierra sesión y vuelve a autenticarse los cambios almacenados deben ser cargados.</p>	Desarrollo	8	1
H41	Filtrado de alertas según el tipo.	<p>En la página alertas el control de tipo “combobox” utilizado para filtrar alertas debe permitir a los usuarios mostrar las alertas basadas en su elección.</p> <p>CA:</p> <p>Seleccionando cada una de las alertas contenidas en el control “combobox” la página de alertas cambia su información dinámicamente sin necesidad de recargar completamente toda la página.</p>	Desarrollo	3	2

Tabla 27. Cuadro de historias de usuario, iteración 7.

Fuente: Elaboración propia.

4.7.2. Implementación

La implementación en esta iteración tuvo mayor énfasis en las tareas concernientes a la aplicación web cliente donde estas tareas fueron desarrolladas basadas en la implementación de funcionalidad para listar, crear, modificar y eliminar alertas.

4.7.2.1. Listar sensores

Para completar esta funcionalidad fue necesario crear una nueva ruta en la aplicación servicio web la cual se encargaría de obtener el listado de sensores desde la base de datos y enviarlos a las aplicaciones cliente. La implementación de esta funcionalidad es mostrada a continuación.

```

app.get('/api/sensor', function(req, res) {
  Sensor.find(function(err, sensors) {
    if (err) {
      res.send(err);
    }
    res.json(sensors);
  });
});

```

Figura 78. Implementación listar alertas, aplicación servicio web.

Fuente: Elaboración propia.

El desarrollo de la página listar sensores fue realizado rápidamente debido a que previamente se desarrolló la misma funcionalidad para listar alertas. La implementación de la función para obtener los datos desde los servicios es mostrada a continuación.

```

$scope.sensors = [];
$scope.loadSensors = function() {
  $http.get('http://localhost:3000/api/sensor')
    .then(function(result) {
      $scope.sensors = result.data;
    });
};

```

Figura 79. Implementación listar alertas, aplicación cliente web.

Fuente: Elaboración propia.

Haciendo uso de estilos y controles provistos por la librería “Bootstrap” y basándose en el diseño de la página para listar alertas la implementación fue completada.

4.7.2.2. Crear sensores

Para la creación de sensores se implementó la funcionalidad de esta característica en la ruta `/api/sensor/create` la cual hace uso del método REST “POST” para recibir información de las aplicaciones cliente y almacenarlas en la base de datos, esta ruta al igual que las siguientes baso su código en implementaciones realizadas para crear alertas.

```
// POST create an Sensor

app.post('/api/sensor/create', function(req, res) {

  Sensor.create({
    name: req.body.name,
    address: req.body.address,
    description: req.body.description
  }, function(err) {
    if (err) {
      res.send(err);
    }
  });

  res.send(200);
});
```

Figura 80. Implementación crear sensores, aplicación servicios web.

Fuente: Elaboración propia.

4.7.2.3. Modificar sensores

La implementación para realizar modificaciones en los sensores es similar a la implementación realizada para el servicio de alertas, por lo tanto basándonos en previas implementaciones se completó la implementación de esta funcionalidad, utilizando como ruta de acceso al servicio web **/api/sensor/:id** donde “id” representa el identificador del sensor a modificar, la implementación de la función fue realizada utilizando el método REST “PUT”.

```
Sensor.update({ _id: req.body._id }, {  
    name: req.body.source,  
    address: req.body.date,  
    description: req.body.type  
,function (error) {  
    if (error) {  
        res.send(500, 'error-update-sensor' + error);  
    }  
  
    res.send(200, 'sensor-updated');  
});  
});
```

Figura 81. Implementación modificar sensores, aplicación servicios web.

Fuente: Elaboración propia.

4.7.2.4. Eliminar sensores

El servicio web de eliminación de sensores hace uso del parámetro “id” al igual que la función para modificar sensores, los cuales buscan en la base de datos, en el caso de que no existiera el “id” especificado se envía un mensaje a las aplicaciones clientes, con el código y el título del error, en caso de que el “id” es encontrado se elimina su información de la base de datos, la ruta de acceso a este servicio es similar **/api/sensor/:id** pero en este caso haciendo uso del método REST “DELETE”.

```

// DELETE a Sensor
app.delete('/api/sensor/:id', function(req, res) {
  Sensor.remove({
    _id: req.params.id
  }, function(err, sensor){
    if (err){
      res.send(500, 'error-delete-sensor' + err);
    }
    res.send(200, 'sensor-deleted');
  });
});

```

Figura 82. Implementación eliminar sensores, aplicación servicios web.

Fuente: Elaboración propia.

La implementación del lado del cliente fue basada en la implementación de páginas creadas para modificar y eliminar alertas, por lo que simplemente se optó en realizar una copia de las mismas modificando la ruta de acceso y los textos que serían mostrados en las respectivas páginas.

4.7.2.5. Listar usuarios

Al igual que la implementación de listar sensores, fue necesario crear una nueva ruta en la aplicación servicio web la cual se encargaría de obtener el listado de usuarios desde la base de datos y enviarlos a las aplicaciones cliente, la ruta definida para esta funcionalidad es **/api/user** la cual hace uso del método REST “GET”. En la implementación podemos notar que se utilizaron los mismos métodos utilizados en listar alertas y listar sensores, para acceder a la base de datos.

```

// GET Users
app.get('/api/user', function(req, res) {
  User.find(function(err, users) {
    if (err) {
      res.send(500, err);
    }
    res.json(users);
  });
});

```

Figura 83. Implementación listar usuarios, aplicación servicios web.

Fuente: Elaboración propia.

4.7.2.6. Crear usuarios

Para la creación de usuarios se utilizó la ruta **/api/user/create** la cual es manejada por el método REST “POST” para enviar información desde las aplicaciones cliente y almacenarlas en la base de datos. La creación de usuarios representa un aspecto fundamental para la aplicación ya que es gracias a esta funcionalidad que podrá ser posible

crear nuevos usuarios, un aspecto que debemos considerar es que los usuarios que son creados utilizando esta función serán automáticamente configurados como usuarios no administradores, esto fue implementado de esta forma para evitar inconvenientes cuando se crean usuarios nuevos desde la aplicación web cliente. Con la creación del usuario un grupo de configuraciones también son creadas las cuales permiten al usuario visualizar todas las alertas, visualizar las gráficas y su lenguaje inicial será inglés. La información relacionada a la llave o “token” será generada basada en una función criptográfica de 16 bits.

```
//POST create
app.post('/api/user/create', function(req, res) {
  if (!req.body.password || req.body.password !== req.body.confirmation) {
    res.send(500, 'password-not-match');
    return;
  }

  var userToken = 'default';
  try {
    token = crypto.randomBytes(16);
    console.log('Have %d bytes of random data: %s', token.length, token);
  } catch (ex) {
    res.send(500, ex);
  }
})
```

Figura 84. Implementación crear usuario, aplicación servicio web.

Fuente: Elaboración propia.

La contraseña de los usuarios es encriptada, por lo tanto haciendo uso de la función “**passwordEncrypted**” se realiza esta acción.

```

bcrypt.hash(req.body.password, 10,
function passwordEncrypted(err, encryptedPwd) {
  if (err) return res.send(500, err);
  User.create({
    name: req.body.name,
    email: req.body.email,
    admin: false,
    online: false,
    encryptedPassword: encryptedPwd,
    token: userToken,
    settings: {
      allAlerts: true,
      unknowAlerts: true,
      informationAlerts: true,
      warningAlerts: true,
      dangerAlerts: true,
      chartSensor: true,
      chartType: true,
      language: 'English'
    }
  }, function(err) {
    if (err) {
      res.send(err);
    }
  });
});

res.send(200);
});

```

Figura 85. Implementación crear usuario, aplicación servicio web.

Fuente: Elaboración propia.

4.7.2.7. Modificar usuarios

La implementación para realizar modificaciones en la información de los usuarios es similar a la implementación realizada para el servicio de alertas, utilizando como ruta de acceso al servicio web **/api/user/:id** donde “id” representa el identificador del usuario, la implementación de la función fue realizada utilizando el método REST “PUT”.

```

//UPDATE User
app.put('/api/user/:id', function(req, res) {
  if (!req.body._id) {
    res.send(500, 'error-update-user: key not found');
  }

  bcrypt.hash(req.body.password, 10,
    function passwordEncrypted(err, encryptedPwd) {
      User.update({ _id: req.body._id }, {
        name: req.body.name,
        email: req.body.email,
        admin: req.body.type,
        online: req.body.online,
        encryptedPassword: encryptedPwd,
        settings: {
          allAlerts: req.body.allAlerts,
          unknowAlerts: req.body.unknowAlerts,
          informationAlerts: req.body.informationAlerts,
          warningAlerts: req.body.warningAlerts,
          dangerAlerts: req.body.dangerAlerts,
          chartSensor: req.body.chartSensor,
          chartType: req.body.chartType,
          language: req.body.language
        }
      },function (error) {
        if (error) {
          res.send(500, 'error-update-user' + error);
        }
      }

      res.send(200, 'user-updated');
    });
}

```

Figura 86. Implementación modificar usuario, aplicación servicio web.

Fuente: Elaboración propia.

4.7.2.8. Eliminar usuarios

El servicio web de eliminación de usuarios al igual que el método modificar hace uso del parámetro “id”, buscando en la base de datos este identificador para proceder a eliminar su información de la base de datos, la ruta de acceso a este servicio es **/api/user/:id** haciendo uso del método REST “DELETE”.

```

//DELETE User
app.delete('/api/user/:id', function(req, res){
  User.remove({
    id: req.params.id
  }, function(err, user){
    if (err) {
      res.send(500, 'user-error-delete' + err);
    }
    res.send(200, 'user-deleted');
  });
})

```

Figura 87. Implementación eliminar usuario, aplicación servicio web.

Fuente: Elaboración propia.

4.7.2.9. Página configuración

Para llevar a cabo la implementación de la página de configuración fue necesario adjuntar los campos de configuración al objeto usuario ya que cada usuario debe ser capaz de manejar sus propias configuraciones. En la aplicación servicios web fue agregado el punto de acceso **/api/user/settings** utilizando como parámetro de búsqueda el email del usuario, con este parámetro fue realizada la actualización de información. Al igual que otras funciones la función para actualizar la información de las configuraciones de usuario cuenta con estructuras de control que permiten verificar que la información enviada sea válida.

```
// POST User settings
app.post('/api/user/settings', function(req, res) {

    var data = req.body;
    var email = req.body.email;

    if (!data && !email) {
        res.send(500, 'user-settings-failed');
    }

    User.update({ email: email },
    {
        settings: {
            allAlerts: data.allAlerts,
            unknowAlerts: data.unknowAlerts,
            informationAlerts: data.informationAlerts,
            warningAlerts: data.warningAlerts,
            dangerAlerts: data.dangerAlerts,
            chartSensor: data.chartSensor,
            chartType: data.chartType,
            language: data.language
        }
    },
    function(err) {
        if (err) {
            res.send(500, 'user-settings-failed');
        }
        res.send(200);
    }
);
});
```

Figura 88. Implementación guardar configuraciones, aplicación servicio web.

Fuente: Elaboración propia.

El método REST utilizado en esta implementación fue “POST” dado que el origen de la solicitud es originada del lado de las aplicaciones cliente.

La implementación del lado del cliente fue realizada en 2 funciones las cuales permiten cargar las configuraciones almacenadas en base de datos y guardar las mismas. La primera función “loadUserSettings” realizará una llamada al servicio web para obtener la información del usuario, con esta información la página configuración permite a los usuarios visualizar

sus configuraciones.

La segunda función “saveSettings” es la encargada de recolectar la información de los controles de la página configuración y enviar estas al servicio web encargado de manejar las configuraciones, la implementación del lado del cliente nos permite preparar un conjunto de elementos visuales que ayudan a los usuarios a determinar si los cambios fueron correctamente enviados y almacenados o si ocurrió algún error.

```
app.controller('SettingsController', function($scope, $http) {  
  
    $scope.loadUserSettings = function() {  
        $http.get('http://localhost:3000/api/user/' + $scope.user.email)  
            .then(function(result) {  
                $scope.settings = result.data.settings;  
            });  
    };  
  
    $scope.saveSettings = function() {  
        settings = {  
            email: $scope.user.email,  
            allAlerts: $scope.settings.allAlerts,  
            unknowAlerts: $scope.settings.unknowAlerts,  
            informationAlerts: $scope.settings.informationAlerts,  
            warningAlerts: $scope.settings.warningAlerts,  
            dangerAlerts: $scope.settings.dangerAlerts,  
            chartSensor: $scope.settings.chartSensor,  
            chartType: $scope.settings.chartType,  
            language: $scope.settings.language  
        };  
  
        $http({  
            method: 'POST',  
            url: 'http://localhost:3000/api/user/settings',  
            data: settings  
        }).  
        success(function (data, status, headers, config) {  
            if (status === 200) {  
                $('#settings-saved').addClass('show');  
                $('#settings-saved').removeClass('hidden');  
            }  
        });  
    };  
});
```

función para cargar las configuraciones almacenadas en base de datos.

Función para guardar los cambios en base de datos.

Clases que serán mostradas si los cambios fueron guardados.

Figura 89. Implementación configuraciones, aplicación cliente web.

Fuente: Elaboración propia.

4.7.2.10. Filtrar Alertas

Una de las características que gran parte de los usuarios desea cuando cuenta con una lista que contiene gran cantidad de información, es el poder filtrar información basándose en algún tipo de criterio, en este caso el criterio de filtración es el tipo de alerta, en este caso el Framework AngularJS nos ayuda a filtrar la información simplemente configurando el elemento HTML que representará a las alertas. Esta característica hace uso de las alertas que fueron cargadas previamente en la lista, simplemente el control con el identificador “type-alert-hide” cambiará su valor y solo las alertas que sean del tipo seleccionado serán mostradas.

```

<input type="text" id="type-alert-hide" ng-show="false" ng-model="typeAlert" value="all"/>
<ul class="dropdown-menu" role="menu">
  <li class="text-capitalize" ng-value="unknow" ng-click="filterAlerts($event)">{{resource
  <li class="text-capitalize" ng-value="info" ng-click="filterAlerts($event)">{{resource
  <li class="text-capitalize" ng-value="warning" ng-click="filterAlerts($event)">{{resource
  <li class="text-capitalize" ng-value="danger" ng-click="filterAlerts($event)">{{resource
  <li class="text-capitalize" ng-value="" ng-click="filterAlerts($event)">{{resourceBund
</ul>

```

Función para filtrar alertas.

```

$scope.filterAlerts = function(event) {
  var type = event.target.innerText;
  var value = $(event.target).attr('ng-value'); Control HTML con el modelo asociado
  $('button-filter-alert').html(type);
  $('#type-alert-hide').val(value);
  $('#type-alert-hide').change();
};
});

```

Figura 90. Implementación filtrar alertas, aplicación web cliente.

Fuente: Elaboración propia.

4.7.3. Pruebas de funcionalidad

A continuación la tabla de pruebas realizadas para verificar el funcionamiento de la aplicación cliente y la aplicación de servicios web, cada una de las pruebas cuenta con sus respectivos resultados.

Prueba	Resultado Esperado	Resultado Obtenido	Estado
Creación Usuario. Haciendo uso de una herramienta “REST client” , con la URL http://localhost:3000/api/user/create Con los parámetros: password: “test” confirmation: “test” name: “username” email: user@email.com Y usando el método POST Verificar el código del resultado y el contenido de la respuesta.	El resultado de la invocación del servicio web de creación de usuarios es 200 OK .	El resultado de la invocación del servicio web de creación de usuarios es 200 OK .	PASÓ

Modificación Usuario. Haciendo uso de una herramienta “REST client” , con la URL <code>http://localhost:3000/api/user/:id</code> Con los parámetros: <code>password: “newTest”</code> <code>confirmation: “newTest”</code> <code>id: “434bc3cc4a7335011c204589”</code> Y usando el método PUT Verificar el código del resultado y el contenido de la respuesta.	El resultado de la invocación del servicio web de modificación de usuarios es 200 OK .	El resultado de la invocación del servicio web de modificación de usuarios es 200 OK .	PASÓ
Eliminar Usuario. Haciendo uso de una herramienta “REST client” , con la URL <code>http://localhost:3000/api/user/:id</code> Con el parámetro: <code>id: “434bc3cc4a7335011c204589”</code> Y usando el método DELETE Verificar el código del resultado y el contenido de la respuesta.	El resultado de la invocación del servicio web de eliminación de usuarios es 200 OK .	El resultado de la invocación del servicio web de eliminación de usuarios es 200 OK .	PASÓ
Creación Sensor. Haciendo uso de una herramienta “REST client” , con la URL <code>http://localhost:3000/api/sensor/create</code> Con los parámetros: <code>name: “SensorTest”</code> <code>address: “10.0.0.1”</code> <code>description: “Test”</code> Y usando el método POST Verificar el código del resultado y el contenido de la respuesta.	El resultado de la invocación del servicio web de creación de sensores es 200 OK .	El resultado de la invocación del servicio web de creación de sensores es 200 OK .	PASÓ

Modificación Sensor. Haciendo uso de una herramienta “REST client” , con la URL <code>http://localhost:3000/api/sensor/:id</code> Con los parámetros: <code>password: “newTest”</code> <code>confirmation: “newTest”</code> <code>id: “412aa2bc4c4512343a334234”</code> Y usando el método PUT Verificar el código del resultado y el contenido de la respuesta.	El resultado de la invocación del servicio web de modificación de sensores es 200 OK.	El resultado de la invocación del servicio web de modificación de sensores es 200 OK.	PASÓ
Eliminar Sensor. Haciendo uso de una herramienta “REST client” , con la URL <code>http://localhost:3000/api/sensor/:id</code> Con el parámetro: <code>id: “412aa2bc4c4512343a334234”</code> Y usando el método DELETE Verificar el código del resultado y el contenido de la respuesta.	El resultado de la invocación del servicio web de eliminación de usuarios es 200 OK.	El resultado de la invocación del servicio web de eliminación de usuarios es 200 OK.	PASÓ

Tabla 28. Pruebas de funcionalidad, iteración 7.

Fuente: Elaboración propia.

4.7.4. Pruebas de Integración

Las pruebas de integración llevadas a cabo en esta iteración se enfocaron en la aplicación cliente web, en la cual se pudo verificar el correcto funcionamiento de la página configuraciones así como la página alertas.

Prueba	Resultado Esperado	Resultado Obtenido	Estado
Guardar Configuraciones. Después de ingresar a la aplicación web cliente dirigirse a la página configuraciones situada en el menú izquierdo de la aplicación. Seleccionar “Spanish” en el control “combobox” de lenguaje. Guardar los cambios, navegar hacia otra página y verificar que la aplicación utiliza el idioma español.	La aplicación cliente muestra todas sus etiquetas en idioma español. Después de navegar entre diferentes páginas estas mantienen el idioma seleccionado.	Después de seleccionar el idioma español toda la aplicación es mostrada en este idioma.	PASÓ

Filtrar alertas. Después de ingresar a la aplicación web cliente dirigirse a la página de alertas haciendo uso del enlace situado en la parte lateral izquierda de la aplicación, haciendo uso del control “combobox” cambiar el filtro a “Danger” verificar que solo las alertas del tipo “Danger” son mostradas en la página.	Haciendo uso del control para la filtración de alertas fue posible verificar que solo las alertas del tipo seleccionado son mostradas en la página.	Las alertas del tipo seleccionado en el filtro de la página alertas son mostradas.	PASÓ
--	---	--	------

Tabla 29. Pruebas de integración, iteración 7.

Fuente: Elaboración propia.

4.7.5. Cuadro de Avance

Código	Estimación	Duración	Estado
H38 (Listar, crear, modificar y eliminar sensores.)	8	4	Hecho
H39 (Listar, crear, modificar y eliminar usuarios.)	5	2	Hecho
H40 (Página de configuraciones.)	8	4	Hecho
H41 (Filtrado de alertas según el tipo.)	3	2	Hecho

Tabla 30. Cuadro de avance, iteración 7.

Fuente: Elaboración propia.

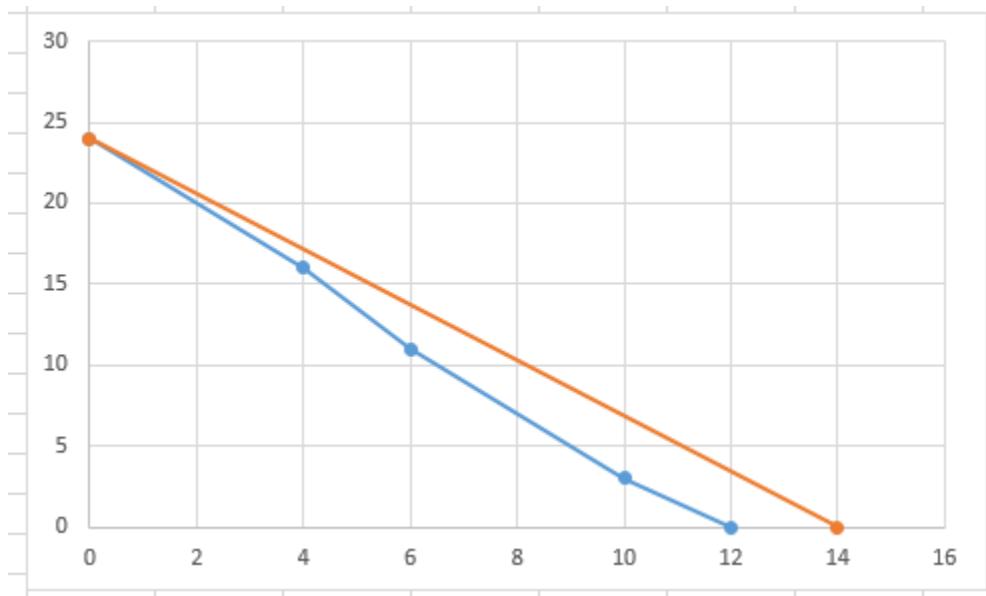


Figura 91. Cuadro de avance iteración 7.

Fuente: Elaboración propia.

4.7.6. Retrospectiva

Bueno	Malo	Mejorar
<ul style="list-style-type: none"> - Las historias de usuario fueron completadas antes de que finalice la iteración. - Las pruebas fueron tomadas en cuenta para la valoración de historias, tomándolas como parte de la historia de usuario y no así como una historia separada. 	<ul style="list-style-type: none"> - La iteración contó con un número reducido de historias de usuario. 	<ul style="list-style-type: none"> - A pesar de que las historias fueron completadas antes del final de la iteración se debe intentar evitar sobreestimar historias de usuario para evitar tiempo improductivo en el proyecto.

Tabla 31. Retrospectiva iteración 7.

Fuente: Elaboración propia.

CAPÍTULO V: CONCLUSIONES Y RECOMENDACIONES

Finalizado el presente proyecto, los resultados se llegaron a las siguientes conclusiones:

La arquitectura del servicio de notificaciones permite a los usuarios adaptar e integrar fácil y rápidamente aplicaciones de monitorización de sistemas a esta. Reduciendo los tiempos empleados en configuración y puesta en marcha de sus sistemas.

La transmisión de información relacionada al estado del hardware y servicios instalados en el servidor, son enviados en tiempo real al servicio web el cual se encarga de la generación instantánea de notificaciones.

Diferentes dispositivos cliente pueden ser enlazados con el servicio de notificaciones, haciendo uso de cuentas de usuario creadas en la aplicación Web mediante la interfaz de programación proporcionada por el servicio de notificaciones.

Manteniendo un bajo grado de acoplamiento y un alto grado de cohesión el proyecto fue separado en diferentes aplicaciones que permiten a los administradores de servidores generar sus propias aplicaciones de notificaciones, así como también utilizar aplicaciones existentes para recabar información de sus recursos.

Con la generación de notificaciones en tiempo real se logra tener mejores tiempos de respuesta, de parte de los administradores de servidores, dado que preparan y brindan soluciones a sus clientes en periodos cortos de tiempo.

Recomendación:

Configurar e implementar módulos de registro de bitácoras, que permitan guardar información de errores en tiempo de ejecución para agilizar las correcciones de los mismos y conocer los errores que se generan a lo largo de la utilización del sistema.

Extender el uso del servicio de notificaciones en tiempo real a distintas áreas, como por ejemplo: seguridad, mensajería, domótica y transporte. Dado que el sistema de notificaciones cuenta con un conjunto de instrucciones, que permiten enlazarlo a diferentes aplicaciones cliente, con un mínimo de conocimientos en programación de aplicaciones.

REFERENCIAS BIBLIOGRÁFICAS

- 1) Express Framework. (2014, Septiembre 9). Express API. <http://expressjs.com/4x/api.html>
- 2) NodeJS Community. (2014). About Documentation. <http://nodejs.org/documentation>
- 3) Abernethy Michael, 2011, Simplemente que es NodeJS. Obtenido de <http://www.ibm.com/developerworks/ssaopensource/library/os-nodejs/>
- 4) SocketIO Community. (s.f.) Using with Node http server. <http://socket.io/docs>
- 5) Marqués, A. (2013). Conceptos sobre APIs REST. Febrero 20, 2015, de asiermarques.com Sitio web: <http://asiermarques.com/2013/conceptos-sobre-apis-rest/>
- 6) León, C. (2013). Que es y para qué sirven WebSockets. Febrero 20, 2015, de Universidad de Laguna Sitio web: <http://nereida.deioc.ull.es/~stw/perlexamples/node26.html/~stw/perlexamples/node26.html>
- 7) Flex IT S.R.L.. (2015). MongoDB. 21 Febrero, 2015, de Flex IT S.R.L. Sitio web: <http://www.flexit.net/sp/developers/mongo-db>
- 8) Mongo Database Project. (2015). The MongoDB 2.6 Manual. <http://www.mongodb.org/manual>
- 9) Torres, M. (2010). CURSO DE jQuery. Febrero 21, 2015, de michelletorres.mx Sitio web: http://michelletorres.mx/cursos-cucei/curso-de-programacion-web/jquery/#.VPfK1_mG914
- 10) abcdiseño.com. (2012). Responsive Web Design. Febrero 22, 2015, de abcdiseño.com Sitio web: <http://www.abcdiseño.com/responsive-web-design/>
- 11) Phonegap Spain, s.f., Phonegap. Sitio Web: <http://www.phonegapspain.com/>
- 12) Fdez, I. (S.F.). Bootstrap, la gran herramienta de los CMS. Febrero 22, 2015, de 4web.es Sitio web: <http://www.4web.es/index.php/blog/105-bootstrap-la-gran-herramienta-de-los-cms>
- 13) Azaustre, C. (2013). ¿Qué es AngularJS? Primeros pasos para aprenderlo. Febrero 22, 2015, de carlosazaustre.es Sitio web: <https://carlosazaustre.es/blog/empezando-con-angular-js/>
- 14) Pressman, Roger S., 1997, p. 73
- 15) AngularJS Web Framework. (2015). Why AngularJS. <https://angularjs.org/>
- 16) Rafa Muñoz. (2011). Introducción a NodeJS. 9 de Septiembre de 2014, Sitio Web: <http://www.rmunoz.net/introduccion-a-node-js.html>
- 17) Jesus Ruiz. (2014). HTML5. 18 de Octubre de 2014, de Mozilla Developer Network Sitio Web: <https://developer.mozilla.org/es/docs/HTML/HTML5>
- 18) Francisco Martinez. (2010). La web en tiempo real, ¿Util o chuminada tecnológica?. 9 de Septiembre de 2014, de GenBeta Sitio web: <http://www.genbeta.com/web/la-web-en-tiempo-real-util-o-chuminada-tecnologica>
- 19) Argentina Gestión Inteligente. (2012). Curso Agiles SCRUM. www.argin.com.ar/gestion-agil-con-scrum

- 20) Tomislav Capan. (2014). Why the hell would I use nodeJS?. 9 Septiembre 2014, de Toptal Sitio web: www.toptal.com/nodejs/why-the-hell-would-i-use-node-js
- 21) Devi Kiran. (2014). REST API – Future of SOA. 9 Septiembre 2014, Kickstartpros Sitio web: www.kickstartpros.com

ANEXOS

Anexo 1: Manual de Instalación

Aplicación Sensor

La instalación de la aplicación sensor puede ser realizada utilizando el instalador ubicado en la carpeta Instaladores/sensor, dentro de esta carpeta se encuentra el instalador, con el nombre “jarvissensor_0.1.1_amd64.deb” este paquete automáticamente copiara los archivos “config.txt” y “sensor.jar” en sus respectivos directorios:

- /opt/sensor/config.txt
- /usr/bin/sensor.jar

Los siguientes pasos describen como debe llevarse a cabo la instalación de la aplicación sensor.

1. Desde línea de comandos diríjase al directorio “Instaladores/sensor”

```
cd Instaladores/sensor  
sudo dpkg -i jarvissensor_0.1.1_amd64.deb
```

2. Una vez que el proceso de instalación ha sido completado satisfactoriamente, proceda a configurar la aplicación para enviar las respectivas alertas al servicio web.

```
/usr/bin/sensor -i
```

3. Ejecute la siguiente instrucción para verificar el correcto funcionamiento de la aplicación sensor.

```
/usr/bin/sensor -d "alerta prueba" -t "titulo alerta" -p info
```

MONIT

La instalación de MONIT será llevada a cabo desde línea de comandos, la instalación de esta aplicación es sencilla dado que la misma se encuentra en los repositorios de casi todas las distribuciones GNU/Linux. A continuación el comando necesario para llevar a cabo la instalación desde un sistema operativo GNU/Linux Ubuntu.

```
sudo apt-get install monit
```

Una vez que concluida la instalación proceda a configurar sus alertas para que estas empiecen a generar información relacionada al estado del servidor. La configuración de MONIT no forma parte de este manual de instalación, mayor información relacionada a la configuración de esta herramienta podrá ser encontrada en el manual de usuario en la sección “Envío de notificaciones desde MONIT”.

Aplicación Web de servicios

Una vez que se ha completado la instalación de la aplicación sensor y MONIT procedemos a realizar la instalación de la aplicación web de servicios esta aplicación requiere tener instaladas algunas dependencias en el sistema operativo, la siguiente lista nos muestra que aplicaciones deben ser instaladas, para que la aplicación de servicios puede funcionar adecuadamente.

- NodeJS
- MongoDB
- G++
- Make
- Bin Utils
- GCC

De todas formas el paquete de instalación solicitará instalar previamente las dependencias mencionadas.

El proceso de instalación de la aplicación web de servicios debe ser iniciado ejecutando el paquete con el nombre “jarvis-service_1.0.1_all.deb”, la instalación de la aplicación web de servicios será llevada a cabo instantáneamente copiando archivos en la ruta /opt/jarvis/.

A continuación los pasos que deben ser realizados desde línea de comandos.

1. Desde línea de comandos diríjase al directorio “Instaladores/server”

```
cd Instaladores/server
```

2. Ejecute la siguiente instrucción para iniciar el proceso de instalación de la aplicación de servicios.

```
sudo dpkg -i jarvis-service_1.0.1_all.deb
```

3. Después de que finalizó el proceso de instalación, es necesario que inicie el servicio web para que las aplicaciones cliente puedan conectarse y empezar a recibir notificaciones. Desde el directorio “/opt/jarvis” ejecute el comando:

```
node app.js
```

Aplicación Web cliente

La aplicación web cliente cuenta con un script de instalación situado en la carpeta “Instaladores/web”, el cual es “install.sh” este script solo puede ser ejecutado en plataformas con sistema operativo GNU/Linux. El script realizará una copia de los archivos necesarios para ejecutar la aplicación web cliente en el servidor web local. La aplicación web cliente funcionará sobre un servidor web, como Apache2, Nginx, IIS, etc.

A continuación los pasos que deben ser realizados desde línea de comandos, en este caso se utilizó la ruta “/opt/lampp/htdocs” como directorio de almacenamiento de la aplicación web.

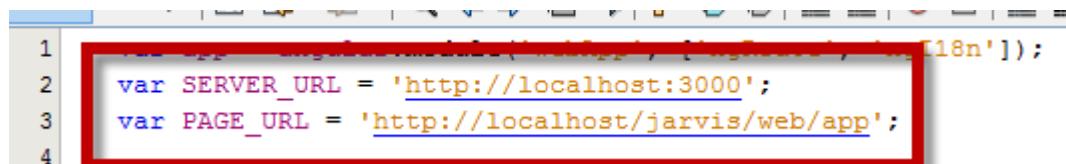
1. Desde línea de comandos diríjase al directorio “Instaladores/web”

```
cd Instaladores/web
```

2. Ejecute la siguiente instrucción para iniciar el proceso de instalación de la aplicación cliente web.

```
chmod +x install.sh  
sudo ./install.sh -p /opt/lampp/htdocs
```

3. Después de que finalizó el proceso de instalación, es necesario que configure la URL del servicio web para que la aplicación cliente pueda conectarse y empezar a trabajar con los servicios que ofrece nuestra aplicación de servicios. En la carpeta de la aplicación web cliente se encuentra el archivo “main.js” en el cual debemos cambiar la dirección URL del servicio web, así como también debe ser cambiada la dirección URL de nuestro servidor que hospeda la aplicación cliente web.



```
1 //  
2 var SERVER_URL = 'http://localhost:3000';  
3 var PAGE_URL = 'http://localhost/jarvis/web/app';  
4 //
```

Figura 92. Configurar URL servicio y URL de la aplicación cliente web.

Fuente: Elaboración propia.

4. Desde el navegador Google Chrome acceda a la aplicación cliente web, cambiando <Ruta Servidor> por la dirección de su servidor que aloja a la aplicación web cliente.

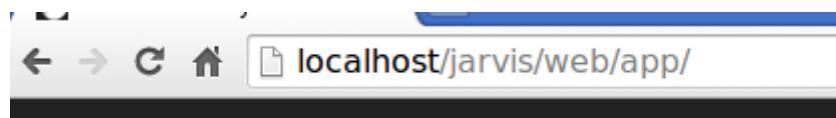


Figura 93. URL aplicación cliente web.

Fuente: Elaboración propia.

5. Para acceder por primera vez a la aplicación web cliente, utilice las siguientes credenciales en el formulario de acceso de la página de inicio. Las credenciales proporcionadas permitirán que cree usuarios, sensores y realice configuraciones. Es recomendable que cree otra cuenta para llevar a cabos las tareas de administración de la aplicación y elimine al usuario “admin”, dado que las credenciales de dicho usuario pueden llegar a ser utilizadas por varios clientes.

Aplicación móvil cliente

La aplicación web cliente cuenta con un archivo instalador el cual se encuentra en el directorio “Cliente Móvil/Instalador”, el nombre del archivo es “jarvis.apk” y debe ser copiado en una de las unidades de almacenamiento del dispositivo móvil. Dado que la aplicación se encuentra empaquetada, los pasos de instalación son fácilmente resumidos en los siguientes.

1. Copiar el archivo “jarvis.apk” en una unidad de almacenamiento del dispositivo móvil.



Figura 94. Diagrama copiar instalador, aplicación móvil.

Fuente: Elaboración propia.

2. Desde el navegador de archivos de su dispositivo móvil proceda a realizar la instalación seleccionando el archivo copiado.

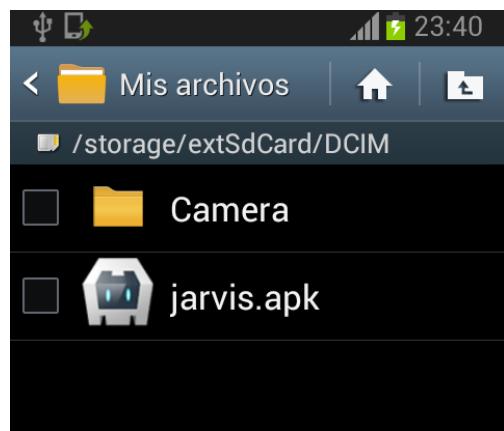


Figura 95 Diagrama instalación aplicación móvil.

Fuente: Elaboración propia.

3. Una ventana de confirmación será mostrada, en la cual indicaremos que se debe proseguir con la instalación.

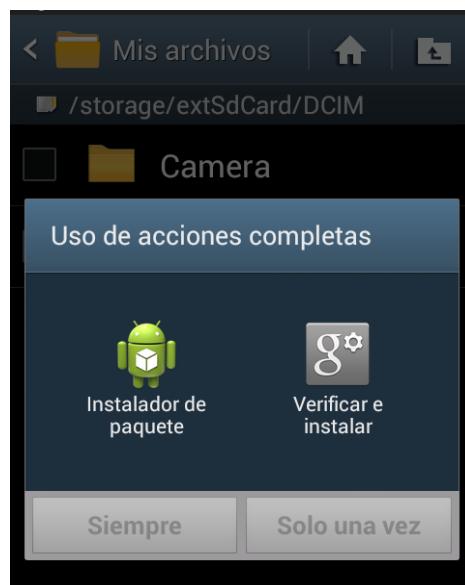


Figura 96. Diagrama instalación aplicación móvil.

Fuente: Elaboración propia.

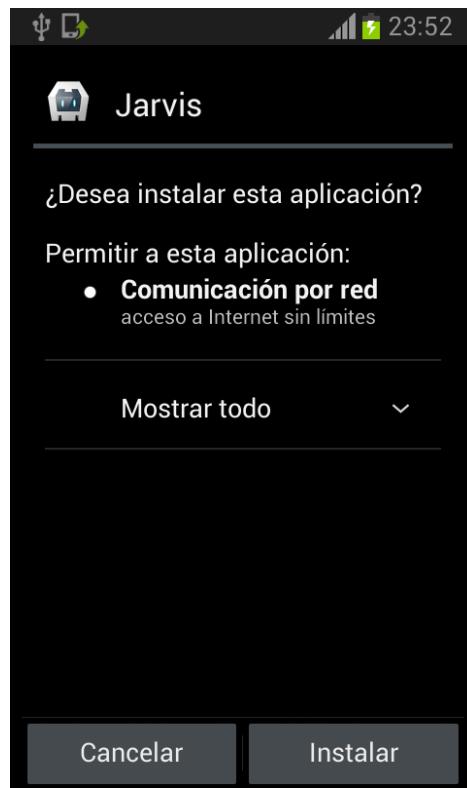


Figura 97. Diagrama instalación completa, aplicación móvil.

Fuente: Elaboración propia.

4. Una vez concluida la instalación, procedemos a iniciar la aplicación móvil.

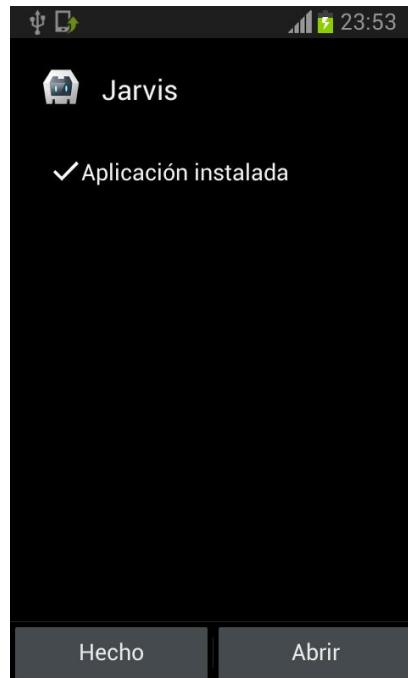


Figura 98. Diagrama abrir aplicación móvil.

Fuente: Elaboración propia.

5. Con la aplicación funcionando, procedemos a llenar los campos con información de nuestra aplicación de servicio recuerde que es necesario contar con credenciales válidas de un usuario para poder completar la conexión con el servicio web.



Figura 99. Diagrama pantalla de inicio, aplicación móvil.

Fuente: Elaboración propia.

Anexo 2: Manual de Usuario

JARVIS es un sistema de notificaciones en tiempo real, que puede ser ejecutado en cualquier sistema operativo que soporte las plataformas NodeJS y Java. Este sistema se encuentra integrado por 4 aplicaciones que se integran correctamente para ofrecer un servicio de notificaciones en tiempo real completo y eficiente.

Aplicación Sensor

La aplicación sensor permite a los usuarios enviar información desde los servidores hacia un servicio web. Una vez concluida la instalación de la aplicación sensor es recomendable llevar a cabo la revisión de la configuración de la aplicación sensor.

Configuración

Para llevar a cabo la tarea de configuración, debemos situarnos en el directorio donde se llevó a cabo la instalación de la aplicación y a continuación utilizaremos la siguiente instrucción desde el intérprete de comandos:

```
java -jar Sensor.jar -i
```

Una vez ejecutada esta instrucción, una ventana de configuración será mostrada, la cual nos permitirá ingresar la URL de la aplicación de servicio, así como también el nombre de usuario, la contraseña y el TOKEN asignado.

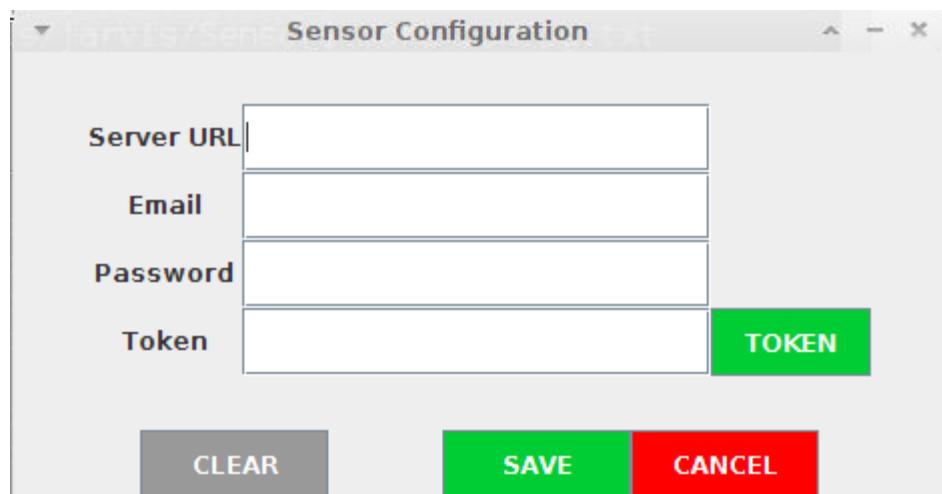


Figura 100. Diagrama ventana de configuración.

Fuente: Elaboración propia.

En caso de no contar con un TOKEN válido asignado haciendo uso de las credenciales ingresadas en los campos de nombre de usuario y contraseña presione el botón TOKEN el cual verificará la información y obtendrá un TOKEN válido que será almacenado en el archivo de configuración.

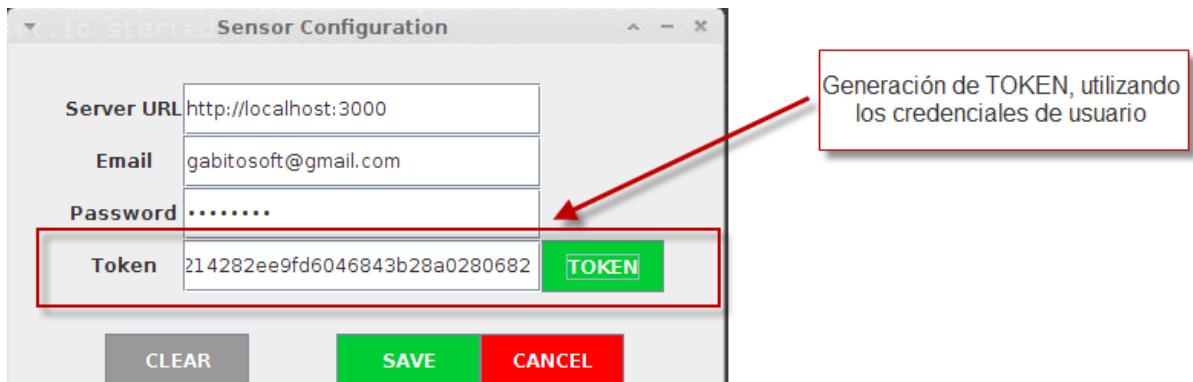


Figura 101. Diagrama obtener TOKEN.

Fuente: Elaboración propia.

Envío notificaciones desde línea de comandos

Una de las ventajas que provee la aplicación sensor es que fácilmente puede ser integrada a diferentes sistemas o aplicaciones dado que la aplicación sensor puede ser invocada desde línea de comandos. A continuación una tabla en la que se muestran los parámetros que pueden ser utilizados en el ejecución de la aplicación sensor.

Parámetro	Descripción
-p	Define el tipo de notificación que se utilizará, actualmente se cuenta con cuatro tipos de alertas: "info", "warning", "danger", "unknow".
-d	Permite definir una corta descripción del motivo por el cual se generó la alerta.
-t	Permite definir el título del motivo por el cual se generó la alerta.
-h	Un mensaje de ayuda es mostrada, en el cual se encuentran listados los diferentes usos de la aplicación con cada uno de los parámetros.
-i	Una ventana de configuración es mostrada, en la cual se pueden llevar a cabo las diferentes configuraciones.

Tabla 32. Descripción parámetros, aplicación sensor.

Fuente: Elaboración propia.

Envío de notificaciones desde MONIT

MONIT es una aplicación que nos permite obtener información relacionada al estado, capacidad y funcionamiento de un servidor, esta aplicación puede ser fácilmente instalada y configurada en un sistema operativo GNU/Linux. MONIT nos facilita la obtención de información por lo cual la aplicación sensor puede ser rápidamente integrada a MONIT, a continuación algunos ejemplos de cómo puede ser utilizada la aplicación sensor a través de MONIT.

Verificar que el servicio HTTP continúa funcionando.

```
check host localhost with address 127.0.0.1  
if failed port 80 protocol http then alert
```

Verificar si el servicio de correos continúa funcionando.

```
check host localhost with address 127.0.0.1  
if failed port 25 with protocol smtp then alert
```

Verificar si el servidor web está consumiendo demasiados recursos

```
check process apache with pidfile /var/run/httpd.pid  
if cpu > 95% for 2 cycles then alert  
if total cpu > 99% for 5 cycles then restart  
if memory > 50 MB then alert  
if total memory > 500 MB then restart
```

Verificar el espacio en disco duro

```
check filesystem disk2 with path /dev/disk2  
if space usage > 95% then alert  
check network eth0 with interface eth0  
if failed link then alert  
if changed link then alert
```

Verificar la conexión y tráfico de red

```
check network eth0 with address "fe80::aa20:66ff:fe50:4f6%eth0"  
if failed link then alert  
if changed link then alert  
if saturation > 80% then alert  
if total upload > 10 GB in last hour then exec  
"/usr/local/bin/script.sh"
```

Aplicación Web de servicios

La aplicación web de servicios permite a los usuarios establecer puntos de acceso para obtener información sobre los servidores que tengan registrados en el sistema, estos puntos de acceso son también conocidos como rutas de acceso.

Rutas de acceso

Las rutas de acceso nos permiten fácilmente identificar las funcionalidades que ofrece la aplicación de servicios por lo general este tipo de información es útil para llevar a cabo la implementación de nuevas aplicaciones cliente basando su información en la provista por la aplicación de servicios, en el siguiente cuadro puede encontrar el conjunto de rutas de acceso que ofrece la aplicación de servicios.

Ruta	Parámetros	Método	Descripción
/api/alert		GET	Devuelve todas las alertas registradas
/api/alert/status/:read	read:Boolean	GET	Devuelve todas las alertas que tengan el mismo valor que el parámetro “read”, lo cual es útil para obtener el listado de alertas leídas o no.
/api/alert/interval	startDate: Date endDate: Date	GET	Devuelve todas las alertas que hayan sido registradas en el intervalo de tiempo definido por “startDate” y “endDate” los cuales representan una fecha de inicio y una fecha fin.
/api/alert/types		GET	Devuelve un listado de las alertas clasificadas por tipo.
/api/alert/querysensor		GET	Devuelve información en formato JSON relacionada al número de alertas generadas por sensor

/api/alert/create	source: String date: Date type: String title: String description: String read: Boolean	POST	Una alerta es creada después de validar la información provista. Para esta ruta de acceso se utiliza, la dirección del servidor que genero la alerta(source), el título de la alerta(title), una corta descripción de la misma(description) y el parámetro para determinar si la alerta ya fue leída o no(read).
/api/alert/:id	id: String	DELETE	Elimina la alerta que tenga el id provisto en los parámetros.
/api/alert/:id	id: String source: String date: Date type: String title: String description: String read: Boolean	PUT	Modifica la información de la alerta que tenga el id provisto en los parámetros.
/api/user/create	name: String email: String password: String	POST	Crea un usuario, utilizando los parámetros nombre (name), correo electrónico (email), la contraseña (password).
/api/user/login	username: String password: String	POST	Permite autenticar a un usuario, utilizando su nombre de usuario (en este caso su correo electrónico) y su contraseña.

/api/user/logout	id: String username: String	POST	Cierra la sesión de usuario y cambio el estado del usuario en la base de datos, id representa el TOKEN o llave provisto en la ruta de acceso "/api/user/login"
/api/user/token	username: String	POST	Genera una llave o TOKEN utilizando el nombre de usuario.
/api/user/settings	data: JSON email: String	POST	Cambia la configuración establecida por el usuario.
/api/session	id: String	DELETE	Elimina la sesión que contenga el id enviado como parámetro.
/api/user	id: String	DELETE	Elimina el usuario que contenga el id enviado como parámetro.
/api/sensor		GET	Devuelve el listado completo de sensores registrados.
/api/sensor/creáte	name: String address: String description: String	POST	Crea un sensor en base a los parámetros: nombre (name), dirección o URL (address) y descripción (description).
/api/sensor	id: String name: String address: String description: String	UPDATE	Actualiza la información del sensor que tenga como identificador el id provisto en los parámetros.
/api/sensor	id: String	DELETE	Elimina el sensor que contenga el id provisto en los parámetros.

Tabla 33. Descripción rutas, aplicación servicio web.

Fuente: Elaboración propia.

La aplicación de servicios web puede ser inicializada, reiniciada y detenida. Esta funcionalidad básica permite tener un control simple de la aplicación así como también permite un control eficiente y rápido.

Iniciar Servicio

Para iniciar la aplicación de servicio desde línea de comandos ejecutar la instrucción:

```
node app.js
```

Esta instrucción verificará si el servicio se encuentra corriendo, en caso de que no lo este, el servicio será inicializado y puesto en marcha en el puerto 3000.

Detener Servicio

Para detener la aplicación de servicio desde línea de comandos, cancele el programa presionando las teclas CTRL+C desde la terminal que inicio el programa o proceda a buscar el proceso asociado:

```
ps -A | grep node
```

Esta instrucción le mostrara los procesos iniciados en nodejs, utilice el comando kill -9 xxx para terminal el proceso, donde xxx representa el número del proceso.

Aplicación Web cliente

La aplicación web cliente permite a los usuarios conocer el estado actual de los servidores que son monitorizados, la aplicación brinda información concerniente a las alertas generadas de cada servidor así como también permite a los usuarios especificar el tipo de alertas que serán mostradas en las aplicaciones cliente.

La aplicación web cliente puede ser accedida desde los navegadores Google Chrome y Firefox en los cuales los componentes de esta aplicación funcionan adecuadamente.

Autenticación

La página de autenticación permite a los usuarios, acceder a la aplicación web cliente, esta página presenta los campos: nombre de usuario y contraseña, en los cuales cada usuario debe ingresar sus credenciales.

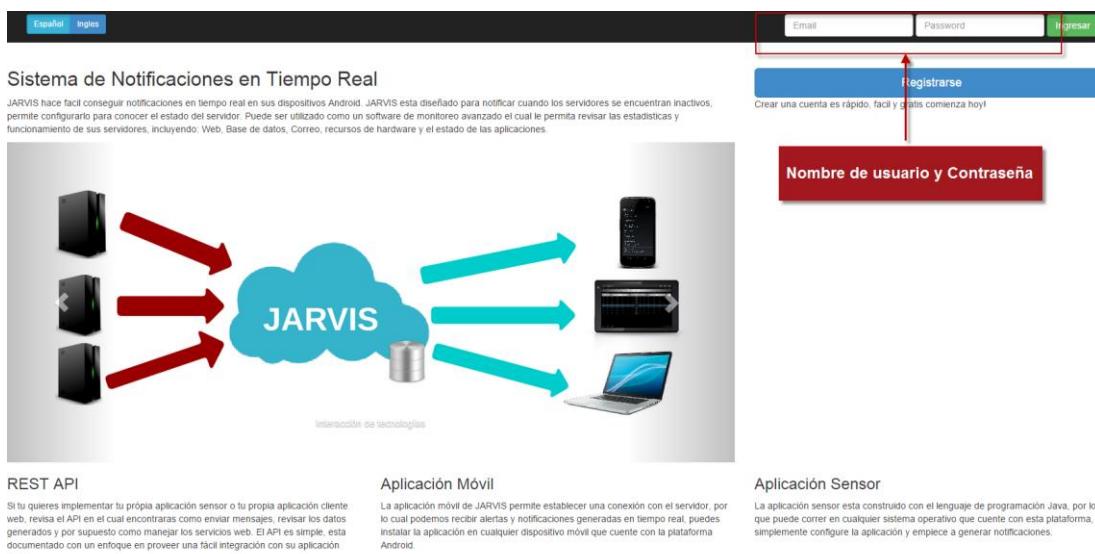


Figura 102. Página de autenticación, aplicación web.

Fuente: Elaboración propia.

La aplicación cliente cuenta con la funcionalidad para cambiar el idioma de la aplicación, en esta versión se utilizan los idiomas inglés y español los cuales pueden fácilmente ser cambiados haciendo uso de los botones situados en la parte superior izquierda de la página de autenticación.

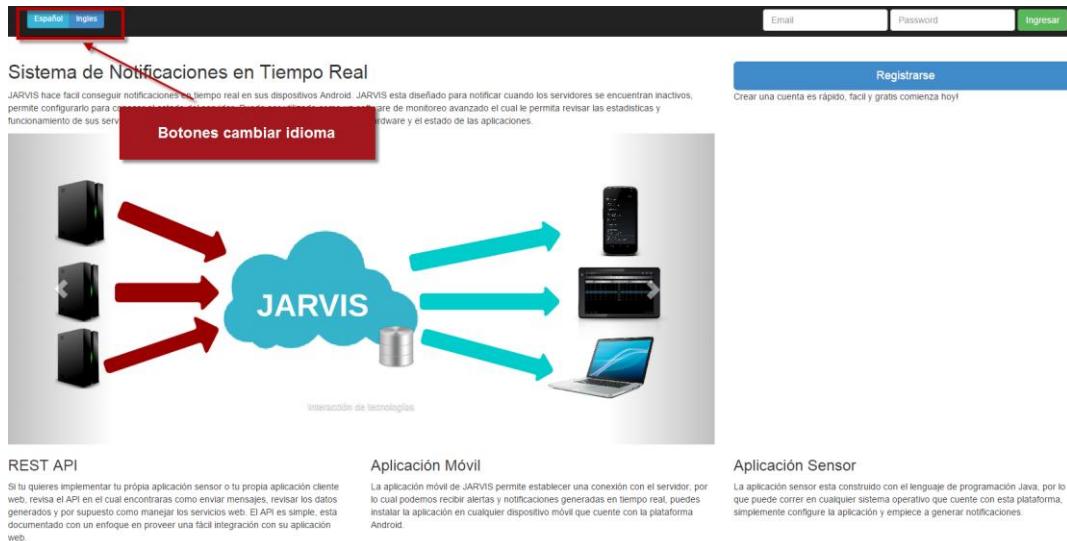


Figura 103. Página autenticación, cambiar idioma.

Fuente: Elaboración propia.

Listar usuarios

Para acceder al listado de usuarios registrados en el sistema, debe dirigirse al menú izquierdo de la aplicación y seleccionar el ítem usuarios, en la página de listado de usuarios podrá navegar a través de los controles de navegación situados en el pie de página. En esta página es mostrada la información concerniente a cada usuario.



Figura 104. Diagrama ir a página listar usuarios, aplicación web.

Fuente: Elaboración propia.

Usuarios				
	Nombre	Email	Administrador	Conectado
<input type="checkbox"/>	Gabriel Delgado	gabitosoft@gmail.com		
<input type="checkbox"/>	Jhon McLain	jhon@hd.com		
<input type="checkbox"/>	holly	holly@hotmail.com		

« 1 2 3 4 5 »
 [Página listar usuarios.](#)

[Eliminar](#)
[Nuevo](#)

Figura 105. Página listado de usuarios, aplicación web.

Fuente: Elaboración propia.

Crear usuarios

La creación de usuarios se realiza desde la página de listado de usuarios, el botón que permite llevar a cabo la navegación hacia la página de creación de usuarios se encuentra situado en el pie de página, una vez presionado el botón “Nuevo” la aplicación mostrará una página donde es necesarios introducir los campos de nombre, correo electrónico, contraseña y confirmación. Estos valores serán registrados en la base de datos una vez que se presione el botón “Guardar”.

Usuarios				
	Nombre	Email	Administrador	Conectado
<input type="checkbox"/>	Gabriel Delgado	gabitosoft@gmail.com		
<input type="checkbox"/>	Jhon McLain	jhon@hd.com		
<input type="checkbox"/>	holly	holly@hotmail.com		

« 1 2 3 4 5 »
 [Ir a página crear usuarios.](#)

[Eliminar](#)
[Nuevo](#)

Figura 106. Diagrama ir a página crear usuarios, aplicación web.

Fuente: Elaboración propia.

Página crear usuario.

Nombre: _____

Email: _____

Contraseña: _____

Confirmacion: _____

Guardar

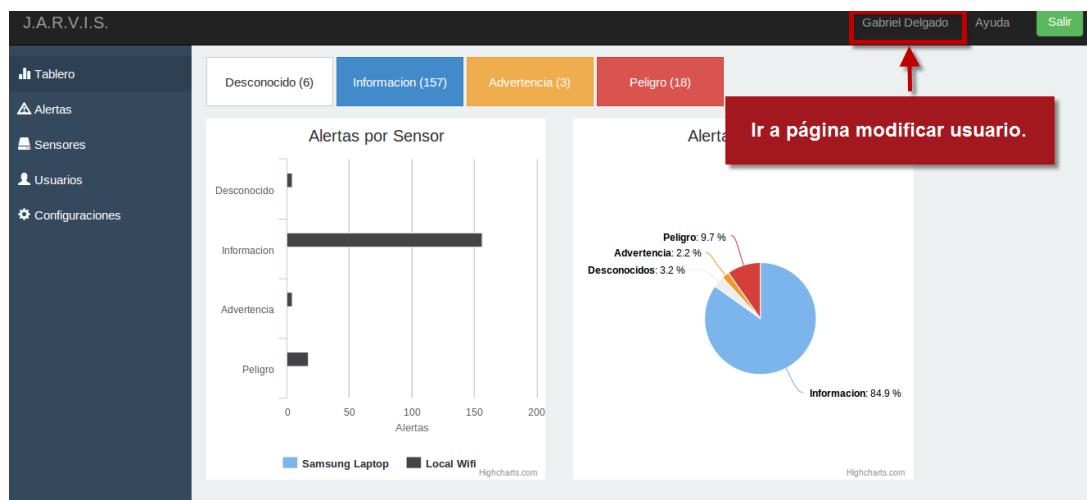
Figura 107. Página crear usuarios, aplicación web.

Fuente: Elaboración propia.

Modificar usuarios

Para llevar a cabo la modificación de información de un usuario distinto al usuario autenticado este debe contar con los permisos de usuario administrador, en caso de serlo, en la lista de usuarios debe seleccionar el usuario de su preferencia. Cuando el usuario es seleccionado un cuadro de dialogo es desplegado el cual contiene campos automáticamente completados con la información actual del usuario.

En caso de que no se cuenten con los permisos necesarios para llevar a cabo la modificación de usuarios, el cambio de información que podrá realizarse, únicamente será sobre el mismo usuario autenticado. Cada usuario puede cambiar su información personal accediendo a la página de información de usuario, para acceder a esta página se debe hacer clic sobre el nombre del usuario autenticado, el cual se encuentra localizado en la parte superior derecha de la aplicación.



La página mostrada después de seleccionar el nombre de usuario es similar a la página de creación de usuarios con la diferencia de que los campos ya se encuentran completados con la información del usuario autenticado.

Gabriel Delgado

Página modificar usuario.

Nombre: Gabriel Delgado

Email: gabitosoft@gmail.com

Contraseña:
Confirmacion:

Save

Figura 108. Página modificar usuario, aplicación web.

Fuente: Elaboración propia.

Eliminar usuarios

Para eliminar usuarios necesitamos dirigirnos a la página de listado de usuarios y en esta página procedemos a seleccionar los usuarios que serán eliminados. Debe tomar en cuenta que solo si cuenta con los permisos de administrador de la aplicación web podrá eliminar usuarios. Un problema conocido con la eliminación de usuarios, ocurre cuando un usuario administrador elimina su cuenta de la lista de usuarios esto ocasiona un funcionamiento erróneo de la aplicación.

	Nombre	Email	Administrador	Conectado
<input type="checkbox"/>	Gabriel Delgado	gabitosoft@gmail.com		
<input type="checkbox"/>	jhon McLain	jhon@hd.com		
<input type="checkbox"/>	holly	holly@hotmail.com		

← Eliminar usuarios.

← Eliminar Nuevo

Figura 109. Eliminar usuarios, aplicación web.

Fuente: Elaboración propia.

Listar sensores

Para acceder al listado de sensores registrados en el sistema, debe dirigirse al menú izquierdo de la aplicación y seleccionar el ítem sensores, en la página de listado de sensores podrá navegar a través de los controles de navegación situados en el pie de página. En esta página es mostrada la información concerniente a cada sensor.



Figura 110. Diagrama ir a página listar sensores, aplicación web.

Fuente: Elaboración propia.

Sensores			
	Nombre	Direccion	Descripcion
<input type="checkbox"/>	Samsung Laptop	127.0.0.1	
<input type="checkbox"/>	Local Wifi	127.0.1.1	Default IP assigned by the OS
Página listar Sensores.			
<input type="button" value="Eliminar"/>	<input type="button" value="Nuevo"/>		

Figura 111. Página listar sensores, aplicación web.

Fuente: Elaboración propia.

Crear sensores

La creación de sensores se realiza desde la página de listado de sensores, el botón que permite llevar a cabo la navegación hacia la página de creación de sensores se encuentra situado en el pie de página. Una vez presionado el botón “Nuevo” la aplicación mostrara una página donde es necesario introducir los campos de nombre, dirección y descripción. Estos valores serán registrados en la base de datos una vez que se presione el botón “Guardar”.

	Nombre	Direccion	Descripcion
<input type="checkbox"/>	Samsung Laptop	127.0.0.1	
<input type="checkbox"/>	Local Wifi	127.0.1.1	Default IP assigned by the OS

Ir a página crear sensores.



Figura 112. Diagrama crear sensores, aplicación web.

Fuente: Elaboración propia.

Modificar sensores

Para llevar a cabo la modificación de información de un sensor, se debe seleccionar uno de la lista de sensores. Cuando el sensor es seleccionado un cuadro de dialogo es desplegado el cual contiene campos que automáticamente son completados con la información actual del sensor.

: [536d5e8065bd45bd0ea4e97b]

Nombre
Samsung Laptop

Descripción

Guardar cambios en base de datos.



Figura 113. Página modificar sensores, aplicación web.

Fuente: Elaboración propia.

Eliminar sensores

Para eliminar sensores necesitamos dirigirnos a la página de listado de sensores y en esta página procedemos a seleccionar los sensores que serán eliminados. Con la selección de sensores establecida procedemos a presionar el botón “Eliminar” el cual enviará los identificadores de los sensores seleccionados al servicio web y los sensores serán eliminados de la base de datos.

	Nombre	Direccion	Descripcion
<input type="checkbox"/>	Samsung Laptop	127.0.0.1	
<input type="checkbox"/>	Local WiFi	127.0.1.1	Default IP assigned by the OS

Eliminar sensor.

« 1 2 3 »

Eliminar **Nuevo**

Figura 114. Diagrama eliminar sensores, aplicación web.

Fuente: Elaboración propia.

Listar alertas

Para acceder al listado de alertas registradas en el sistema, debe dirigirse al menú izquierdo de la aplicación y seleccionar el ítem alertas, en la página de listado de alertas podrá navegar a través de los controles de navegación situados en el pie de página. En esta página es mostrada la información concerniente a cada alerta generada.



Figura 115. Diagrama ir a página listar alertas, aplicación web.

Fuente: Elaboración propia.

Alertas

Origen	Título	Tipo	Fecha	Leido	Descripción
127.0.1.1	Test10	warning	05/08/2014 11:24:48		Testing data 10

« 1 2 3 »

Página listar alertas.

Eliminar

Figura 116. Página listar alertas, aplicación web.

Fuente: Elaboración propia.

Filtrar alertas

En la página de listado de alertas podemos realizar la filtración de alertas, el filtrado de alertas está enfocado en mostrar a los usuarios únicamente las alertas que considere importantes, el filtrado está basado en el tipo de alertas que se utilizan, las cuales son: peligro, advertencia, información y desconocidas. Para aplicar el filtrado de alertas utilice el campo de selección ubicado en la parte superior de la página listar alertas, este control cuenta con los tipos de alertas mencionados previamente.

Origen	Título	Tipo	Fecha	Descripción
127.0.1.1	Memory	info	28/07/2014 01:41:50	The Memory is almost full
127.0.1.1	Memory	info	28/07/2014 01:42:24	Server is already update
127.0.1.1	Memory	info	28/07/2014 01:42:41	Proxy has been updated
127.0.1.1	Memory	info	28/07/2014 01:43:44	Process on memory has been cleaned

« 1 2 3 »

Eliminar

Figura 117. Diagrama filtrar alertas, aplicación web.

Fuente: Elaboración propia.

Modificar alertas

Para llevar a cabo la modificación de información de una alerta, se debe seleccionar la alerta deseada de la lista de alertas. Cuando la alerta es seleccionado un cuadro de dialogo es desplegado el cual contiene campos que automáticamente son completados con la información de la alerta, únicamente el campo “Leído” puede ser modificado, dado que una alerta es generada con información predeterminada.

The screenshot shows a modal dialog box with the following fields:

- Id: [53e0f7400ab143bd0b20d0e7]
- Origen: 127.0.1.1
- Tipo: warning
- Título: Test10
- Fecha: 05/08/2014 11:24:48
- Descripción: Testing data 10
- Leido:

At the bottom of the dialog box, there is a red callout box containing the text "Guardar cambios en base de datos." with an arrow pointing to the "Guardar" button, which is also highlighted with a red border.

Figura 118. Cuadro de diálogo, modificar alertas, aplicación web.

Fuente: Elaboración propia.

Eliminar alertas

Para eliminar alertas necesitamos dirigirnos a la página de listado de alertas y en esta página procedemos a seleccionar las alertas que serán eliminadas. Con la selección de alertas establecida procedemos a presionar el botón “Eliminar” el cual enviara los identificadores de las alertas seleccionadas al servicio web y estas serán eliminadas de la base de datos.

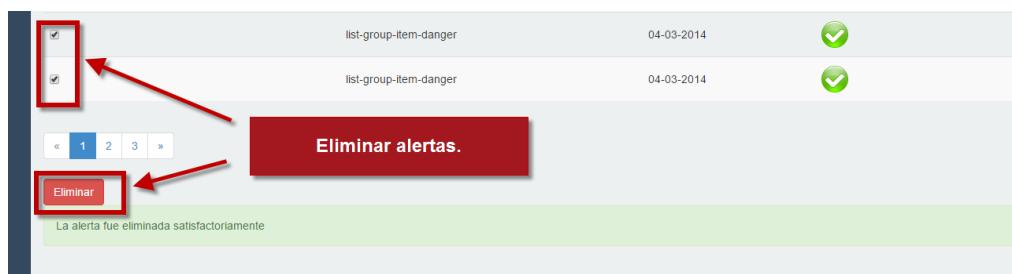


Figura 119. Diagrama eliminar alertas, aplicación web.

Fuente: Elaboración propia.

Configurar gráficos

La aplicación web cliente permite a los usuarios configurar la aplicación para que permita mostrar los gráficos de estado utilizados en la página de inicio, estos gráficos contienen información relacionada al número de alertas generadas por sensor, así como también información del número de alertas generadas por tipo. Para determinar si las alertas serán mostradas en la página de inicio, nos dirigimos al menú izquierdo de la aplicación y seleccionamos el ítem “Configuración”, una vez seleccionado el ítem la página de configuración será desplegada, en la sección de Gráficas habilitamos o deshabilitamos las opciones: “Mostrar alertas por sensor” y “Mostrar alertas por tipo”, cuando estas opciones se encuentran habilitadas las gráficas son mostradas en la página de inicio, en caso de no estar habilitadas las opciones un panel gris es mostrado.



Figura 120. Diagrama ir a página configuraciones, aplicación web.

Fuente: Elaboración propia.



Figura 121. Página configuraciones, habilitar gráficos, aplicación web.

Fuente: Elaboración propia.

Configurar alertas

La aplicación web cliente permite a los usuarios configurar la aplicación para que permita recibir las alertas deseadas o de mayor importancia para el usuario. Esta funcionalidad puede ser encontrada en la página de configuración en la sección “Alertas”. En esta sección podemos habilitar o deshabilitar las alertas que deseamos que lleguen hasta las diferentes aplicaciones cliente. En caso de deshabilitar la opción “Habilitar Alertas” ninguna alerta será enviada a las aplicaciones cliente.

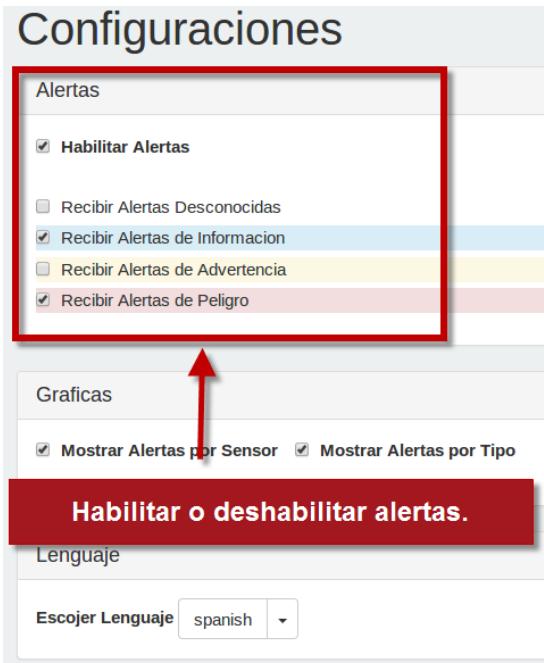


Figura 122. Página configuraciones, habilitar alertas, aplicación web.

Fuente: Elaboración propia.

Configurar idioma

La aplicación web cliente permite a los usuarios configurar la aplicación para que utilice 2 idiomas diferentes, en la página de configuraciones en la sección “Lenguaje” podemos encontrar el control que permite a los usuarios seleccionar el idioma de su preferencia, en esta versión de la aplicación se cuenta con los idiomas inglés y español, el usuario podrá seleccionar el idioma de su preferencia e instantáneamente notará que los textos utilizados en la aplicación son afectados directamente.



Figura 123. Página configuraciones, cambiar idioma.

Fuente: Elaboración propia.

Todos los cambios realizados en la página de configuraciones no serán guardados en base de datos si el botón de guardar no es presionado. Por lo que es recomendable que después de realizar cualquier tipo de configuración guarde los cambios haciendo click sobre el botón “Guardar”.

Cerrar sesión

La aplicación web cliente cuenta con el botón de cerrar sesión, el cual está ubicado en la parte derecha superior de la aplicación, una vez que el usuario presione este botón la aplicación será cerrada cambiando el estado del usuario en la base de datos, y mostrando la página de autenticación de usuarios.

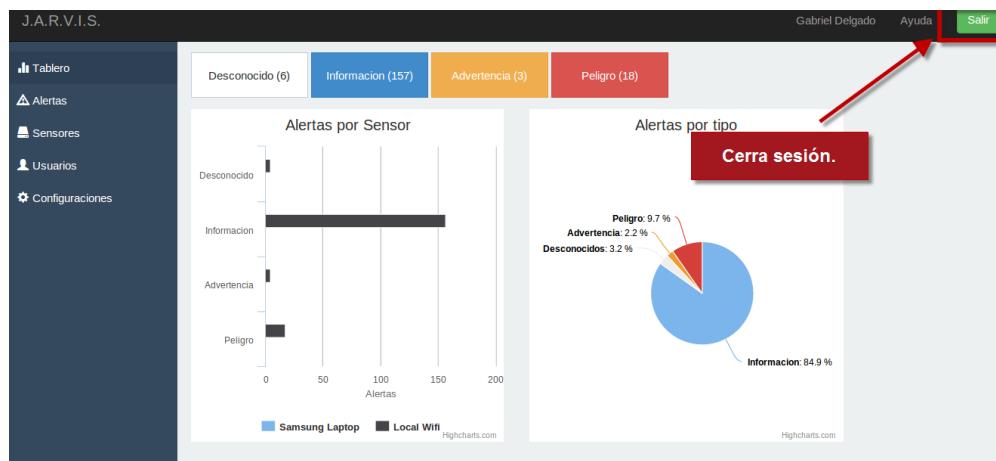


Figura 124. Diagrama cerrar sesión, aplicación web.

Fuente: Elaboración propia.

Aplicación móvil cliente

Autenticación

La página de inicio de autenticación permite a los usuarios, acceder a la aplicación móvil, esta página presenta los campos: nombre de usuario y contraseña, en los cuales cada usuario debe ingresar sus credenciales.

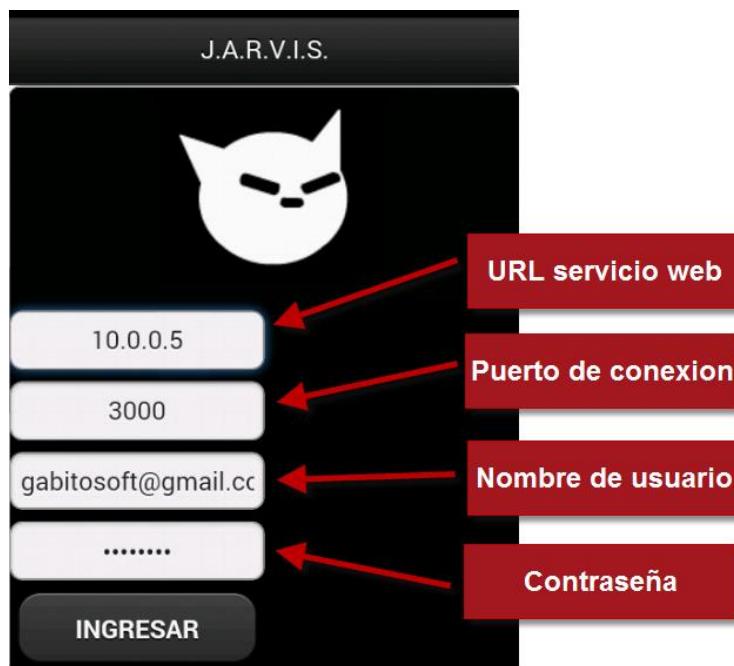


Figura 125. Pantalla autenticación, aplicación móvil.

Fuente: Elaboración propia.

Listar alertas

Para acceder al listado de alertas registradas en el sistema, debe dirigirse a la pestaña alertas de la aplicación, en esta pestaña podrá revisar la información concerniente a cada alerta generada. Cabe mencionar que únicamente las alertas habilitadas en la pestaña de configuración serán mostradas cuando estas sean generadas.

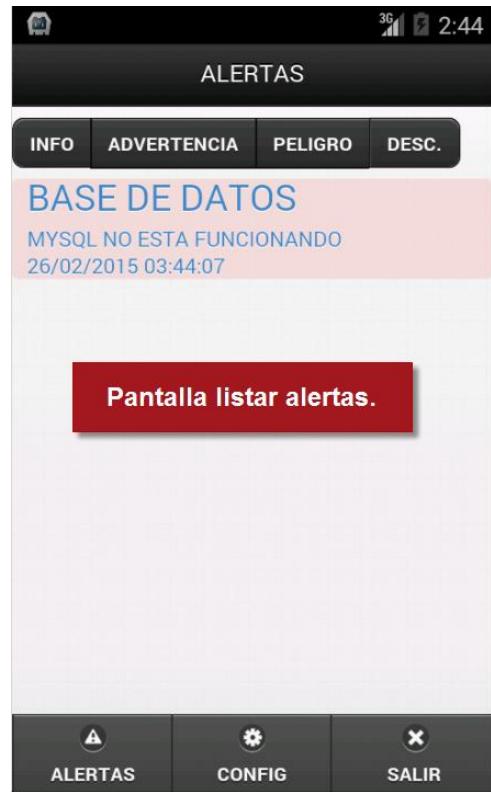


Figura 126. Pantalla listar alertas, aplicación móvil.

Fuente: Elaboración propia.

Filtrar alertas

En la pestaña de listado de alertas podemos realizar la filtración de alertas, el filtrado de alertas está enfocado en mostrar a los usuarios únicamente las alertas que considere importantes, el filtrado está basado en el tipo de alertas que se utilizan, las cuales son: peligro, advertencia, información y desconocidas. Para aplicar el filtrado de alertas utilice los botones ubicados en la parte superior de la pestaña listar alertas, estos botones representan cada uno de los tipos de alertas mencionados previamente.



Figura 127. Pantalla filtrar alertas, aplicación móvil.

Fuente: Elaboración propia.

Configurar alertas

La aplicación móvil permite a los usuarios configurar la aplicación para que permita recibir las alertas deseadas o de mayor importancia para el usuario. Esta funcionalidad puede ser encontrada en la pestaña de configuración. En esta pestaña podemos habilitar o deshabilitar las alertas que deseamos que lleguen hasta nuestro dispositivo. En caso de deshabilitar la opción “Habilitar Alertas” ninguna alerta será enviada a nuestro dispositivo. Se debe tomar en cuenta que los cambios realizados en esta pestaña serán reflejados en la página de configuración de la aplicación web cliente.



Figura 128. Diagrama ir a pantalla configuraciones, aplicación móvil.

Fuente: Elaboración propia.



Figura 129. Pantalla configuraciones, aplicación móvil.

Fuente: Elaboración propia.

Cerrar aplicación

La aplicación móvil cuenta con el botón de salir, el cual está ubicado en la parte derecha

inferior de la aplicación, una vez que el usuario presione este botón la aplicación será cerrada instantáneamente cambiando el estado del usuario en la base de datos, y mostrando la página de autenticación de usuarios.

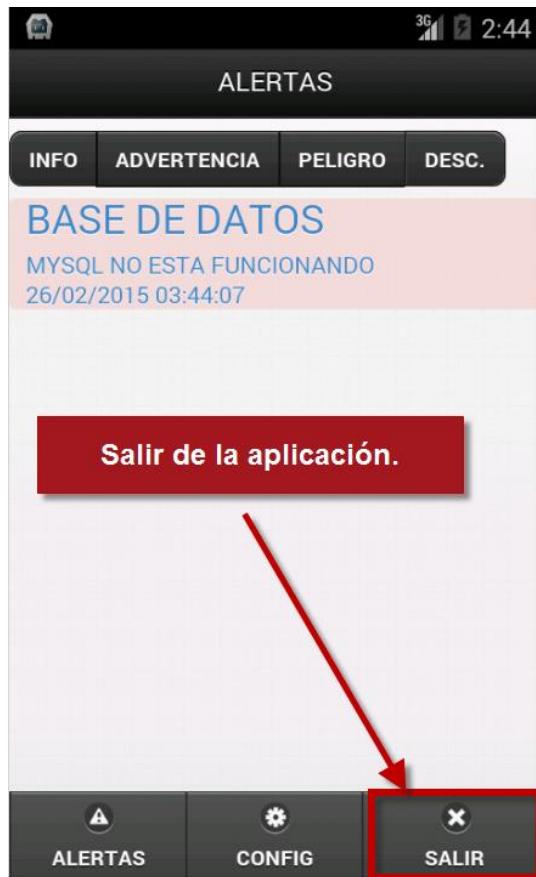


Figura 130. Diagrama cerrar aplicación, aplicación móvil.

Fuente: Elaboración propia.

Anexo 3: Manual Técnico

En este manual se enfatizara la importancia de algunos módulos y código necesarios para que el sistema de notificaciones funcione adecuadamente. Comenzaremos mostrando código relacionado a la aplicación sensor, dado que es la aplicación que generará las respectivas notificaciones.

Aplicación Sensor

Uno de los aspectos más importantes de la aplicación sensor es el hecho de poder enviar notificaciones, esta funcionalidad se encuentra directamente relacionada con la clase NotifierHTTP.java, en esta clase se definen todos los métodos y configuraciones necesarias para llevar a cabo las tareas de comunicación.

Clase NotifierHTTP

```
public JSONObject sendPOST(JSONObject json, String url) throws IOException {

    JSONObject result;
    URL object = new URL(url);
    HttpURLConnection connection = (HttpURLConnection) object.openConnection();
    connection.setDoOutput(true);
    connection.setDoInput(true);
    connection.setRequestProperty("Content-Type", "application/json; charset=utf8");
    connection.setRequestProperty("Accept", "application/json");
    connection.setRequestMethod("POST");

    // Send json
    OutputStreamWriter wr = new OutputStreamWriter(connection.getOutputStream());
    wr.write(json.toString());
    wr.flush();

    // Display what returns the POST request
    StringBuilder sb = new StringBuilder();
    int HttpResult = connection.getResponseCode();
    result = new JSONObject();

    // Send json
    OutputStreamWriter wr = new OutputStreamWriter(connection.getOutputStream());
    wr.write(json.toString());
    wr.flush();

    // Display what returns the POST request
    StringBuilder sb = new StringBuilder();
    int HttpResult = connection.getResponseCode();
    result = new JSONObject();

    if (HttpResult == HttpURLConnection.HTTP_OK) {

        String line;
        try ( BufferedReader br = new BufferedReader(new InputStreamReader(connection.getInputStream(), "utf-8"))
              ) {
            while ((line = br.readLine()) != null) {
                sb.append(line);
                sb.append("\n");
            }
        }

        result.put("result", sb.toString());
    }
} else {

    result.put("result", HttpResult);
    // TODO add to logger
}

return result;
```

Clase Configurator

Otro aspecto importante de la aplicación sensor es el hecho de brindar a los usuarios la oportunidad de cambiar la configuración relacionada a la dirección URL del proveedor de servicios de esta forma la aplicación no está sujeta a generar alertas para un servicio específico, el código en la clase Configurator.java se encarga de llevar a cabo esta configuración de manera transparente para el usuario.

```
public Configurator() {

    this.settings = new JSONObject();
    settings.clear();

    try {
        FileManager fileManager = new FileManager();
        if (fileManager.existFile(CONFIG_PATH)) {

            ArrayList<String> content = fileManager.readFile(CONFIG_PATH);
            settings.put(KEY_URL, content.get(0));
            settings.put(KEY_USERNAME, content.get(1));
            settings.put(KEY_PASSWORD, content.get(2));
            settings.put(KEY_TOKEN, content.get(3));
        } else {
            settings.put(KEY_URL, DEFAULT_ANSWER);
            settings.put(KEY_USERNAME, DEFAULT_ANSWER);
            settings.put(KEY_PASSWORD, DEFAULT_ANSWER);
            settings.put(KEY_TOKEN, DEFAULT_ANSWER);
        }
    } catch(Exception ex) {

        System.out.println(ex.getMessage());
        // TODO add to logger
    }
}
```

Toda la información que es manejada tanto en el lado del cliente como en el lado del servidor es en formato JSON, para permitir a distintas aplicaciones trabajar facilmente con esta información. En el código mostrada podemos notar que los valores relacionados a la URL, nombre de usuario, contraseña y TOKEN son leídos desde un archivo de configuración el cual es manejado por la clase FileManager.java.

Aplicación Web de servicios

La aplicación web de servicios representa el corazón del sistema de notificaciones, esta aplicación es la encargada de gestionar las distintas solicitudes y respuestas que son generadas desde las distintas aplicaciones cliente hacia el servidor.

User.js (/api/user/create) Crear usuario

```
//POST create
app.post('/api/user/create', function(req, res) {
  if (!req.body.password || req.body.password !== req.body.confirmation) {
    res.send(500, 'password-not-match');
    return;
  }

  var userToken = 'default';
  try {
    token = crypto.randomBytes(16);
    console.log('Have %d bytes of random data: %s', token.length, token);
  } catch (ex) {
    res.send(500, ex);
  }

  bcrypt.hash(req.body.password, 10,
  function passwordEncrypted(err, encryptedPwd) {
    if (err) return res.send(500, err);
    User.create({
      name: req.body.name,
      email: req.body.email,
      admin: false,
      online: false,
      encryptedPassword: encryptedPwd,
      token: userToken,
      settings: {
        allAlerts: true,
        unknowAlerts: true,
        informationAlerts: true,
        warningAlerts: true,
        dangerAlerts: true,
        chartSensor: true,
        chartType: true,
        language: 'English'
      }
    }, function(err) {
      if (err) {
        res.send(err);
      }
    });
  });
}

res.send(200);
});
```

User.js (/api/user/:id) Modificar usuario

```
//UPDATE User
app.put('/api/user/:id', function(req, res) {
  if (!req.body._id) {
    res.send(500, 'error-update-user: key not found');
  }

  bcrypt.hash(req.body.password, 10,
    function passwordEncrypted(err, encryptedPwd) {
      User.update({ _id: req.body._id }, {
        name: req.body.name,
        email: req.body.email,
        admin: req.body.type,
        online: req.body.online,
        encryptedPassword: encryptedPwd,
        settings: {
          allAlerts: req.body.allAlerts,
          unknowAlerts: req.body.unknowAlerts,
          informationAlerts: req.body.informationAlerts,
          warningAlerts: req.body.warningAlerts,
          dangerAlerts: req.body.dangerAlerts,
          chartSensor: req.body.chartSensor,
          chartType: req.body.chartType,
          language: req.body.language
        }
      },function (error) {
        if (error) {
          res.send(500, 'error-update-user' + error);
        }

        res.send(200, 'user-updated');
      });
    
```

User.js (/api/user/login) Autenticar usuario

```
// POST login
app.post('/api/user/login', function(req, res) {

  if (!req.param('username') || !req.param('password')) {
    res.send(500, 'invalid-parameters');
    return;
  }

  User.findOne({ email: req.param('username') }, function(err, user) {

    if (err) {
      res.send(500, 'problems-findOne');
      return;
    }

    if (!user) {
      res.send(500, 'user-not-found');
      return;
    }

    // Compare password from the form params to the encrypted password of the user found.
    bcrypt.compare(req.param('password'), user.encryptedPassword, function(err, valid) {
      if (err) {
        res.send(500, 'encryptedPassword-failed');
        return;
      }
    })
  })
})
```

Verificación de parámetros y
encriptación de contraseña

```
// Change status to online
var token = req.session.id;
User.update({ email: user.email }, { online : true, token: token }, function(err) {
  if (err) {
    res.send(500, 'login-fail-update');
  }

  // Inform other sockets (e.g. connected sockets that are subscribed) that this user is now logged in
  for (var username in app.connections) {
    app.connections[username].emit('connected', user.email);
    app.connections[username].emit('alert', 'User: ' + user.email + ' now is connected');
  }

  // Log user in
  Session.create({
    id: token,
    username: req.param('username')
  }, function(err) {
    if (err) {
      res.send(err);
    }
  });
}

res.send(200, { message: 'access-granted', token: user.token });
});
```

Cambia el estado del
usuario a conectado

Envia notificaciones a todos
los usuarios

Se crea la sesión de usuario

User.js (/api/user/settings) Configuraciones

```
// POST User settings
app.post('/api/user/settings', function(req, res) {

    var data = req.body;
    var email = req.body.email;

    if (!data && !email) {
        res.send(500, 'user-settings-failed');
    }

    User.update({ email: email },
    {
        settings: {
            allAlerts: data.allAlerts,
            unknowAlerts: data.unknowAlerts,
            informationAlerts: data.informationAlerts,
            warningAlerts: data.warningAlerts,
            dangerAlerts: data.dangerAlerts,
            chartSensor: data.chartSensor,
            chartType: data.chartType,
            language: data.language
        }
    },
    function(err) {
        if (err) {
            res.send(500, 'user-settings-failed');
        }
        res.send(200);
    }
);
});
```

User.js (/api/user/:id) Eliminar usuario

```
//DELETE User
app.delete('/api/user/:id', function(req, res){
    User.remove({
        id: req.params.id
    }, function(err, user){
        if (err) {
            res.send(500, 'user-error-delete' + err);
        }
        res.send(200, 'user-deleted');
    });
});
```

User.js (/api/user) Obtener usuarios

```
// GET Users
app.get('/api/user', function(req, res) {
  User.find(function(err, users) {
    if (err) {
      res.send(500, err);
    }
    res.json(users);
  });
});
```

Alert.js(/api/alert/create) Crear alerta

```
User.findOne({ email: req.body.email }, function(error, user) {
  if (user.token != req.body.token) {
    console.log('Invalid Token');
    res.send(500, 'Invalid Token');
    return;
  }
});

// POST create
Alert.create({
  source: req.body.source,
  date: req.body.date,
  type: req.body.type,
  title: req.body.title,
  description: req.body.description,
  read: false
}, function(err, alert) {
  if (err) {
    res.send(500, 'error-post-alert' + err);
    return;
  }

  // Send the alert to the client
  for (var username in app.connections) {
    app.connections[username].emit('alert', alert);
  }
});

res.send(200);
```

Alert.js (/api/alert/:id) Modificar alerta

```
//UPDATE Alert
app.put('/api/alert/:id', function(req, res) {
  if (!req.body._id) {
    res.send(500, 'error-update-alert: key not found');
  }

  Alert.update({ _id: req.body._id }, {
    source: req.body.source,
    date: req.body.date,
    type: req.body.type,
    title: req.body.title,
    description: req.body.description,
    read: req.body.read
  },function (error) {
    if (error) {
      res.send(500, 'error-update-alert' + error);
    }

    res.send(200, 'alert-updated');
  });
});
```

Alerts.js (/api/alert/:id) Eliminar alerta

```
//DELETE Alert
app.delete('/api/alert/:id', function(req, res) {
  Alert.remove({
    _id: req.params.id
  }, function(err, alert){
    if (err){
      res.send(500, 'error-delete-alert' + err);
    }
    res.send(200, 'alert-deleted');
  });
});
```

Sensor.js (/api/sensor/create) Crear sensor

```
// POST create an Sensor

app.post('/api/sensor/create', function(req, res) {

  Sensor.create({
    name: req.body.name,
    address: req.body.address,
    description: req.body.description
  }, function(err) {
    if (err) {
      res.send(err);
    }
  });

  res.send(200);
});
```

Sensor.js (/api/sensor/:id) Eliminar sensor

```
// DELETE a Sensor
app.delete('/api/sensor/:id', function(req, res) {
  Sensor.remove({
    _id: req.params.id
  }, function(err, sensor){
    if (err){
      res.send(500, 'error-delete-sensor' + err);
    }
    res.send(200, 'sensor-deleted');
  });
});
```

Sensor.js (/api/sensor) Obtener sensores

```
app.get('/api/sensor', function(req, res) {
  Sensor.find(function(err, sensors){
    if (err) {
      res.send(err);
    }
    res.json(sensors);
  });
});
```

Sensor.js (/api/sensor/:id) Modificar sensor

```
Sensor.update({ _id: req.body._id }, {  
    name: req.body.source,  
    address: req.body.date,  
    description: req.body.type  
}, function (error) {  
    if (error) {  
        res.send(500, 'error-update-sensor' + error);  
    }  
  
    res.send(200, 'sensor-updated');  
});  
});
```

Aplicación Web cliente

La aplicación cliente web permite a los usuarios acceder a toda la funcionalidad que ofrece el servicio web, a través de navegadores web, esta aplicación es la encargada de ofrecer a los usuarios la posibilidad de habilitar o deshabilitar las alertas. Haciendo uso de las páginas de configuración los usuarios fácilmente deben llevar a cabo las configuraciones mencionadas. Los aspectos más importantes de esta aplicación están relacionados al hecho de obtener información y enviar información a las respectivas rutas para que estas se encarguen de mostrar las páginas adecuadas.

Main.js (MainController) Definición de rutas

```
app.config(function($routeProvider) {  
    $routeProvider.  
        when('/dashboard', { templateUrl:'../alert/summary.html' }).  
        when('/alert', { templateUrl: '../alert/list.html' }).  
        when('/sensor', { templateUrl: '../sensor/list.html' }).  
        when('/newSensor', { templateUrl: '../sensor/new.html' }).  
        when('/setting', { templateUrl: '../setting/index.html' }).  
        when('/user', { templateUrl: '../user/list.html' }).  
        when('/newUser', { templateUrl: '../user/new.html' }).  
        when('/profile', { templateUrl: '../user/profile.html' }).  
        when('/help', { templateUrl: '../user/help.html' }).  
        otherwise({ redirectTo: '/', templateUrl: '../alert/summary.html' });  
});
```

Main.js (AlertController) Cargar alertas

```
$scope.loadAlerts = function() {
  $http.get('http://localhost:3000/api/alert')
    .then(function(result) {
      $scope.alerts = result.data;
    });
};
```

Main.js (SettingsController) Cargar y guardar configuraciones

```
app.controller('SettingsController', function($scope, $http){
  $scope.loadUserSettings = function() {
    $http.get('http://localhost:3000/api/user/' + $scope.user.email)
      .then(function(result) {
        $scope.settings = result.data.settings;
      });
  };
  $scope.saveSettings = function() {
    settings = {
      email: $scope.user.email,
      allAlerts: $scope.settings.allAlerts,
      unknowAlerts: $scope.settings.unknowAlerts,
      informationAlerts: $scope.settings.informationAlerts,
      warningAlerts: $scope.settings.warningAlerts,
      dangerAlerts: $scope.settings.dangerAlerts,
      chartSensor: $scope.settings.chartSensor,
      chartType: $scope.settings.chartType,
      language: $scope.settings.language
    };
    $http({
      method: 'POST',
      url: 'http://localhost:3000/api/user/settings',
      data: settings
    }).success(function (data, status, headers, config) {
      if (status === 200) {
        $('#settings-saved').addClass('show');
        $('#settings-saved').removeClass('hidden');
      }
    });
  };
});
```

función para cargar las configuraciones almacenadas en base de datos.

Función para guardar los cambios en base de datos.

Clases que serán mostradas si los cambios fueron guardados.

Main.js (SensorController) Cargar sensores

```
$scope.sensors = [];
$scope.loadSensors = function() {
  $http.get('http://localhost:3000/api/sensor')
    .then(function(result) {
      $scope.sensors = result.data;
    });
};
```

Main.js (UserController) Autenticar usuario

```

$scope.loginUser = function() {
  $scope.user = {
    username: $scope.username,
    password: $scope.password
  };

  $http({
    method: 'POST',
    url: SERVER_URL + '/api/user/login',
    data: $scope.user
  }).then(
    success(function (data, status, headers, config) {
      console.log('data-login', data);
      Session.updateSession($scope.username);
      $window.location.href = 'partial/user/index.html?user=' + $scope.username + '&token=' + data.token;
    }),
    error(function (data, status, headers, config) {
      console.log(data);
      if (status === 500) {

        if (data === 'user-not-found' || data === 'username-password-mismatch') {

          $('#login-error').addClass('hidden');
          $('#login-error').removeClass('show');

          $('#login-failed').removeClass('hidden');
          $('#login-failed').addClass('show');

          return;
        }
      }
    })
  );
}

```

Aplicación móvil cliente

La aplicación cliente móvil permite a los usuarios recibir las notificaciones con las respectivas alertas generadas, esta aplicación ofrece la posibilidad de mantener a los usuarios conectados a los servicios para contar con toda la información que es generada en tiempo real. Cabe mencionar que el aspecto más importante de esta aplicación es el hecho de mantener una conexión a través de WebSockets con el servicio web, el cual habilita un canal abierto para una comunicación bidireccional.

Utils.js (login) Autenticación de usuarios

```
$ajax({
    url: 'http://' + server + ':' + port + '/api/user/login',
    data: { username : usr, password: pwd },
    type: 'post',
    dataType: 'json',

    beforeSend: function() {
        // This callback function will trigger before data is sent
        $.mobile.loading( 'show' );
    },
    complete: function() {
        // This callback function will trigger on data sent/received complete
        $.mobile.loading( 'hide' );
    },
    success: function (request, result) {
        window.location.href = '#alerts';
        init();
    },
    error: function (request, error) {
        // This callback function will trigger on unsuccessful action
        alert('Network error has occurred please try again');
    }
})
```

Si la autenticación fue realizada, la página listar alertas será mostrada

Connection.js (init) Creación WebSocket

```
loadScript('js/socket.io.js', function () {

    var socket = io.connect($('#server').val() + ':' + $('#port').val());

    socket.on('alert', function (data) {
        $('#alertList').append('<li class="alert-item list-group-item-' + data.type +
            '<h3 class="list-group-item-heading">' + data.title + '</h3>' +
            '<p class="list-group-item-text">' + data.description + '</p>' +
            '<p class="list-group-item-text">' + data.date + '</p>' +
            '</a></li>');
    });

    socket.emit('message', { message: 'Client was notified' });

    window.plugin.notification.local.add({
        id: data._id,
        title: data.title,
        message: data.description,
        autoCancel: true,
        sound: 'android.resource://sounds-767-arpeggio'
    });
});

socket.on('connect', function () {
    $('#status').html('Connected');
    socket.emit('username', $('#username').val());
});
```

Definimos la ruta de conexión

Insertamos un elemento nuevo en la lista de alertas