Universidad Autónoma de Madrid

Escuela Politécnica Superior

Doble Grado en Ingeniería Informática y Matemáticas

TRABAJO DE FIN DE GRADO

INTERFAZ WEB PARA LA GESTIÓN DE SONDAS DE RED DE ALTAS PRESTACIONES

Juan Sidrach de Cardona Mora
Tutor: Dr. Sergio López Buedo

Junio 2015

INTERFAZ WEB PARA LA GESTIÓN DE SONDAS DE RED DE ALTAS PRESTACIONES

Autor: Juan Sidrach de Cardona Mora
Tutor: Dr. Sergio López Buedo

High-Performance Computing and Networking Research Group
Dpto. de Tecnología Electrónica y de las Comunicaciones
Escuela Politécnica Superior
Universidad Autónoma de Madrid

Junio 2015

Agradecimientos

Me gustaría expresar aquí mi agradecimiento a todas aquellas personas que han contribuido en este trabajo y que me han acompañado en la carrera, a las que debo gran parte de lo que soy.

A Sergio, mi tutor, por permitirme hacer este Trabajo de Fin de Grado y ofrecerme ser parte del grupo de investigación en el que se enmarca.

Al HPCN y a todos sus miembros, por hacerme sentir uno más y por el apoyo brindado durante este último año.

A mis padres, Mariano y Llanos, por darme la oportunidad de venir a estudiar a Madrid y apoyarme incondicionalmente durante todos estos años.

A mi tío Juan, por acogerme y ampararme en el estudio de las matemáticas.

A mi hermana, Laura, por haberme soportado tantos años y por haberme reído las gracias inmerecidamente.

A todos los integrantes de "El Pozo", por todas las risas y por la ayuda prestada.

A Óscar, con quien pasé el verano trabajando, por saber escuchar y por su espíritu crítico.

A Richi, por haber estado ahí siempre que lo he necesitado.

A Rubén, con quien he recorrido la recta final de esta etapa y compartido las dudas sobre lo que viene después.

Y por último, a Carolina, por todos los momentos juntos, por hacerme feliz y afortunado de tenerla a mi lado.

Gracias a todos.

Abstract

A network probe is a device capable of capturing or injecting network traffic. This work is founded on a custom-made FPGA based probe. So far, the only possibility to interact with this probe was through the command line, difficulting the management of the probe for people without any previous knowledge of it. Another issue present in the probe's control was the existence of many aspects related to its operation that were being handled externally, such as storage, sorting and managing the traces or monitoring the system performance.

The proposed application facilitates the use of the network probe through a web-based graphical interface. This interface allows the user to manage the probe without any specific knowledge of its inner workings. In addition, it brings together other relevant aspects of the capture and injection of web traffic, such as trace storage, write speed of the disks or trace format conversion. The interface implementation follows design that enables it to be used on mobile devices with a similar user experience. The final product provides a simple interface to manage all the relevant aspects of the probe, which displays the state and additional information of the system visually, using graphs and statistics.

This project, however, has not exclusively consisted in the development of a graphical interface. A base architecture, which formalizes the management of network probes from a web interface, has also been designed. This proposal contemplates how to structure the components so that it is extensible to other network probes, not just the one selected. With this goal in mind, the system has been divided into two components, back-end and front-end, which communicate with each other and that can be hosted on different servers. Thereby, a REST Web Service has been designed and implemented for the back-end, which formalizes the status and functionality of the FPGA, adding also control over other important aspects mentioned above. For the front-end, an ad hoc framework has been created, which has served as a starting point for the web interface. It is intended that most of the proposed solution could be reused in similar projects, in which the management of the probes is done through command line.

Key words — web interface, responsive design, web service, API REST, capture and reproduction of network traffic, FPGA.

Resumen

Una sonda de red es un dispositivo capaz de capturar tráfico de red o de inyectarlo. Este trabajo está basado en una sonda a medida sobre una FPGA. Hasta ahora, la única posibilidad era interaccionar con esta sonda desde la línea de comandos, lo que dificultaba su gestión para personas sin conocimientos previos de la misma. Otro problema presente en el control de la sonda era la existencia de numerosos aspectos relacionados con su funcionamiento que se manejaban de forma externa, tales como el almacenamiento, clasificación y gestión de las trazas, o la monitorización del rendimiento del sistema.

La aplicación propuesta facilita la utilización de la sonda de red mediante una interfaz gráfica basada en tecnologías web. Esta interfaz permite gestionar la sonda sin tener un conocimiento específico de su funcionamiento interno. También agrupa otros aspectos relevantes de la captura y reproducción de tráfico web, como el almacenamiento de las trazas, la velocidad de escritura en disco o la conversión entre formatos de traza. La implementación de la interfaz sigue un diseño que permite que sea utilizada desde dispositivos móviles con una experiencia de usuario similar. Se ha conseguido, como producto final, una interfaz simple que permite gestionar todos los aspectos de la sonda considerados relevantes, presentando el estado e información adicional del sistema de forma visual, mediante gráficos y estadísticas.

Este proyecto, sin embargo, no ha consistido únicamente en el desarrollo de una interfaz gráfica. Se ha diseñado también una arquitectura base que formaliza la gestión de sondas de red desde una interfaz web. Esta propuesta contempla cómo estructurar los componentes de forma que sea extensible a otras sondas de red, no solo la seleccionada. Con este objetivo, se ha dividido el sistema en dos componentes, back-end y front-end, que se comunican entre sí y que pueden estar alojados en un servidor distinto cada uno. Así, se ha diseñado e implementado un Servicio Web REST en el back-end, que formaliza el estado y funcionalidad de la FPGA, añadiendo también control sobre otros aspectos relevantes mencionados anteriormente. Para el front-end se ha creado un framework propio, que ha servido de punto de partida para la interfaz web. Se pretende que gran parte de la solución propuesta sea reutilizable en proyectos similares, en los que el manejo de las sondas se realiza por línea de comandos.

Palabras clave — interfaz web, diseño responsive, servicio web, API REST, captura y reproducción de tráfico de red, FPGA.

Glosario

Back-end Componentes internos de la aplicación que procesan los datos provenientes del front-end. 3, 12–14, 21–25, 33–35, 37, 41, 44–46, 49–51, 74, 77

Bitstream En este contexto se refiere al binario que configura el Hardware de la FPGA. 35

Código abierto Software cuyo código fuente y otros derechos son publicados bajo una licencia que puede permitir a los usuarios utilizar, cambiar, mejorar el software y redistribuirlo. 5, 13–15, 47

Framework Entorno software que proporciona una funcionalidad base para facilitar la organización y desarrollo de aplicaciones similares. 14, 15, 24, 33, 38, 42, 49, 69–72, 77, 78

Front-end Interfaz, parte de la aplicación que interacciona directamente con el usuario. 2, 3, 12–15, 21–25, 33, 35, 37–39, 41, 42, 44, 46, 49–51, 74, 75, 77

Log Fichero que contiene un registro de eventos. 23, 67, 77

Proxy Servidor que sirve de intermediario entre las peticiones de recursos que realiza un cliente a otro servidor. 44, 69, 72, 78

Script Programa interpretado que se almacena en un archivo de texto plano. 74, 75, 77

Servicio Web Conjunto de métodos remotos accesibles a través de la red. 23, 25, 37, 44, 49–51, 55–59, 69, 72, 76, 79, 80

Simple Formato de traza que acepta la FPGA utilizada. 17, 18, 46, 51, 59, 61, 62, 65, 102–104, 107–109

Traza Archivo que contiene paquetes de red capturados. 1, 2, 6–9, 17, 18, 24, 27, 29–32, 35, 40, 46, 49, 51, 57–59, 61–65, 81–91, 93, 98–100, 102–111

Acrónimos

AJAX Asynchronous JavaScript and XML. 15, 72

API Application Programming Interface (métodos públicos de una aplicación). 33, 35, 44, 50, 57, 79, 80

FPGA Field-Programmable Gate Array. 1, 5, 17–19, 22, 23, 25, 33, 35–37, 44, 49–52, 55–59, 61–63, 67–69, 72, 76, 79–93, 95, 97–100

HTML HyperText Markup Language. 14, 15

HTTP Hypertext Transfer Protocol. 21, 22, 24, 69, 79

IFG InterFrame Gap (pausa temporal entre paquetes). 18, 31, 63, 86, 88, 99

JSON JavaScript Object Notation. 23, 76, 79

Pcap Packet capture (formato de traza, utilizado por programas como *Wireshark* y *tcpdump*). 18, 46, 51, 65, 102, 104, 105, 107, 109

PHP PHP Hypertext Pre-processor. 14, 15, 67, 69, 72, 74–78

RAID Redundant Array of Independent Disks. 2, 18, 26, 46, 57, 58, 64–66, 94, 95, 101

REST Representational State Transfer. 14, 22, 33, 49, 50

URL Uniform Resource Locator. 22, 46, 67, 69, 79

Índice general

Índice de tablas

Índice de figuras

1 Introducción

Este Trabajo de Fin de Grado, realizado en colaboración con el grupo de investigación *High-Performance Computing and Networking (HPCN)*, tiene como propósito el desarrollo de una interfaz web para la gestión de sondas de red de altas prestaciones. En los siguientes apartados se explican la motivación y los objetivos principales del mismo, sucedidos de una descripción de la estructura del resto de la memoria.

1.1. Motivación

Una sonda de red es un dispositivo capaz de capturar tráfico de red (sonda pasiva) o de inyectarlo (sonda activa). Este dispositivo puede ser algo tan sencillo como un ordenador convencional, en el cual se ha instalado una tarjeta *Ethernet* estándar o una tarjeta a medida basada en FPGA. La aplicación a desarrollar se basará en una sonda de este último tipo [1].

Hasta ahora, la única posibilidad existente es interaccionar con esta sonda desde la línea de comandos, lo que dificulta su gestión para personas sin conocimientos previos de la misma. Otro problema en el control de la sonda es que existen numerosos aspectos relacionados con su funcionamiento que se manejan de forma externa, tales como el almacenamiento, clasificación y gestión de las trazas.

Surge entonces la necesidad de una simplificación en el manejo de la sonda, que permita a usuarios no avanzados su utilización. Para ello, se desarrollará una interfaz gráfica basada en tecnologías web, que facilitará conocer y manejar el estado de la sonda. Además, permitirá administrar otros aspectos del sistema, como los mencionados anteriormente.

Por último, se cree interesante intentar abstraer la arquitectura de la solución propuesta, de forma que cubra un problema más general: manejar una sonda de red mediante una interfaz web. Se considera que es una extensión natural del proyecto, ya que existen aspectos comunes a cualquier sonda de red, como el manejo de las trazas, la monitorización del rendimiento o el front-end. Esto permitirá adaptar, sin demasiado esfuerzo, el proyecto realizado a otras sondas de red distintas de la seleccionada.

1.2. Objetivos

Los objetivos principales planteados en este Trabajo de Fin de Grado son los siguientes:

- Crear una arquitectura base que formalice una solución para la gestión de sondas de red desde una interfaz web. Esta propuesta contemplará cómo estructurar los componentes de forma que sea extensible a otras sondas de red, no solo a la seleccionada. Así, mediante la abstracción sobre el problema dado, se pretende que gran parte de la solución propuesta sea reutilizable en proyectos similares, en los que el manejo de la sonda se realiza mediante la línea de comandos.

- Desarrollar una interfaz web que permita la gestión de una sonda de red de altas prestaciones. Esta interfaz permitirá, de manera visual, conocer el estado actual de la sonda de red y configurarla para reproducir o capturar tráfico de red. Además, la interfaz web permitirá gestionar las trazas capturadas con la sonda mencionada. Se pretende así facilitar el control de todos los componentes que intervienen en la captura y reproducción de tráfico de red, y que no sea necesario conocer previamente el funcionamiento interno de la sonda para poder manejarla y experimentar con ella.

- Monitorizar el estado del servidor al que está conectada la sonda de red de altas prestaciones. Para ello, la aplicación mostrará al usuario estadísticas sobre el servidor relevantes en este contexto. Por una parte, se podrá conocer el espacio de almacenamiento disponible para guardar trazas, para que el usuario pueda liberar espacio es caso de ser necesario, y prevenir así que la sonda deje de poder capturar tráfico de red. Por otro lado, si el sistema de archivos en que se almacenan las trazas capturadas con la sonda está sobre un RAID, se podrá conocer la velocidad de escritura global del sistema y de cada uno de los discos que integran el RAID, ya que es un factor que puede limitar el rendimiento de la sonda.

- Registrar estadísticas sobre el uso de la aplicación, lo que ayudará a localizar y solucionar errores internos. Adicionalmente, se podrán analizar estos registros para conocer el grado de utilización de la sonda por diferentes usuarios, y plantearse entonces cómo optimizar el tiempo necesario para realizar las tareas más frecuentes.

1.3. Estructura del documento

En el capítulo 2 se realiza un análisis del estado del arte. Se analizan algunos sistemas similares de captura y reproducción de tráfico de red, haciendo énfasis en las interfaces de gestión y monitorización con las que cuentan estos sistemas, para posteriormente extraer conclusiones sobre lo estudiado.

En el capítulo 3 se define la aplicación que se va a diseñar, así como la metodología seguida y las herramientas utilizadas en el proyecto. En el capítulo 4 se describen los requisitos funcionales y no funcionales de la aplicación.

En el capítulo 5 se formaliza el diseño de la aplicación a implementar, comentando la arquitectura de la aplicación y los módulos en los que se divide. En el capítulo 6 se documenta la implementación de la aplicación, estructurada en dos partes bien diferenciadas: back-end y front-end. En el capítulo 7 se explica el proceso seguido para la verificación y validación de la aplicación construida, comprobando así que cumple los objetivos planteados y funciona correctamente. En el capítulo 8 se especifica cómo se va a realizar el mantenimiento de la aplicación.

En el capítulo 9 se exponen las conclusiones finales sobre el trabajo realizado. Por último, en el capítulo 10 se plantean posibles líneas de trabajo futuro que podrían ser abordadas con el objetivo de mejorar y ampliar diferentes aspectos de la aplicación desarrollada.

2

Estado del arte

En este capítulo se presenta *The Open Source Network Tester*, la única plataforma de captura y reproducción de tráfico de red que se basa en la misma FPGA que este Trabajo de Fin de Grado. Adicionalmente, se exponen brevemente otras herramientas similares de captura, reproducción y análisis de tráfico de red. Se intenta, por tanto, detallar de forma objetiva los aspectos positivos y las carencias de las distintas alternativas, poniendo así en contexto el proyecto a desarrollar. Dado que la finalidad de este proyecto es implementar una interfaz web, se hace especial hincapié en la interfaz de usuario de la que dispone cada uno de los sistemas analizados.

2.1. The Open Source Network Tester

The Open Source Network Tester (OSNT) [2, 3, 4] es un generador y capturador de tráfico de red. Esta aplicación se ejecuta sobre cuatro *NetFPGA-10G* (que, como ya se ha comentado, es el mismo modelo de sonda que el tratado en este proyecto). *OSNT* permite generar o capturar paquetes de cualquier tamaño y establecer la tasa a la que se realizan estas operaciones.

Una de los puntos fuertes de *OSNT* es que tanto el diseño *hardware* como el *software* son de código abierto y pueden ser modificados [5]. Se facilita así extender la aplicación para soportar nuevos protocolos. Otra característica relevante es que posee una interfaz gráfica de usuario desde la que manejar la sonda de red (ver Figura 2.1). Esta interfaz permite configurar la captura o reproducción seleccionando la velocidad de transmisión.

Sin embargo, se han detectado también algunas carencias y aspectos susceptibles de mejora en *OSNT*. En primer lugar, no gestiona algunos elementos que están relacionados

con la sonda y que podrían ser de utilidad. Por ejemplo, no controla que el sistema de archivos tenga una velocidad de escritura suficiente para la captura, lo que puede afectar al rendimiento y hacer que no se aproveche al máximo la sonda de red. Tampoco comprueba que el espacio disponible dentro del disco sea suficiente antes de programar una captura. Por último, no permite una gestión activa de las trazas almacenadas (clasificación, detalles, borrado de las mismas, etc.).

Respecto a la interfaz, se han identificado algunos aspectos que podrían ser perfeccionados. Uno de ellos es que no es posible conocer el estado de la sonda de manera intuitiva, y otro es que tampoco existe una clara separación entre captura/reproducción y módulos adicionales. Asimismo, gran parte de la información que se muestra en la interfaz podría ser agrupada en forma de gráficos, visualmente más explicativos. Se echa de menos además soporte para varios idiomas, algo que facilitaría la utilización de la aplicación a usuarios no angloparlantes. En términos de arquitectura, el sistema requiere de acceso físico (o remoto por *ssh*) al servidor para su manejo, cuando podría ser útil exponer la funcionalidad como un servicio accesible desde cualquier dispositivo. Finalmente, el hecho de que la interfaz esté sobre el propio servidor de la sonda tiene como consecuencia que si el sistema se saturase, la interfaz podría quedar inutilizable.

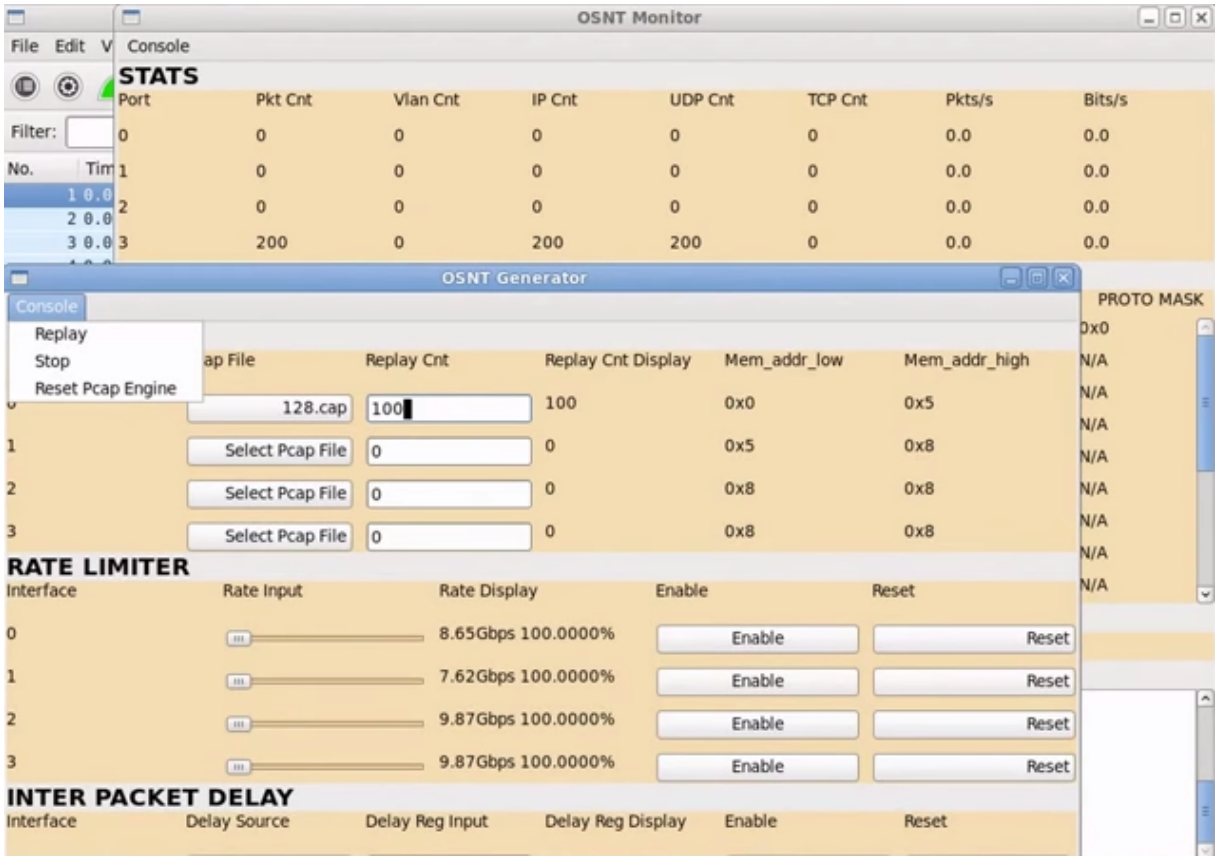


Figura 2.1: Captura de la interfaz gráfica de *OSNT*.

2.2. Otras herramientas similares

tcpdump y libpcap

Las utilidades *tcpdump* y *libpcap* [6], implementadas como librerías, permiten analizar el tráfico que circula por la red. De esta forma, el usuario puede capturar y mostrar en tiempo real los paquetes transmitidos y recibidos en una red a la que el ordenador esté conectado. Se suelen utilizar para depurar aplicaciones que envían y reciben tráfico de red, aunque tienen otros usos como leer datos no cifrados enviados por otros ordenadores.

Estas librerías poseen otras características adicionales que las hacen interesantes. Por un lado, son de código libre y tienen una comunidad detrás que añade funcionalidad y corrige errores de forma continua. Por otra parte, da la posibilidad de aplicar filtros sobre las capturas, seleccionando los paquetes que se considere oportuno. Desde el punto de vista de interfaz de usuario, *tcpdump* y *libpcap* se manejan desde la línea de comandos. Esto hace más complicado empezar a capturar tráfico de red si no se tienen conocimientos previos de estas herramientas.

Wireshark

Wireshark [7] es un analizador de protocolos de red. Provee una funcionalidad similar a la de *tcpdump*, añadiendo una interfaz gráfica y más opciones de filtrado de la información. Así, permite o bien ver todo el tráfico en tiempo real que pasa a través de una red (almacenándolo opcionalmente), o bien analizar tráfico ya capturado anteriormente.

Una de las ventajas de *Wireshark* es que soporta casi todos los protocolos de red, y permite de manera sencilla añadir protocolos adicionales. Otra es que, al contrario que *tcpdump* y *libpcap*, cuenta con una interfaz gráfica de usuario para manejar la captura y reproducción. Sin embargo, esta interfaz no es demasiado intuitiva, y para el uso de filtros se requiere la consulta del manual.

Detect-Pro

Detect-Pro [8] es una herramienta modular de análisis pasivo (sin intervenir). Analiza todo el tráfico de un enlace de red paquete a paquete, proporcionando series temporales de tráfico, análisis a nivel de flujo, análisis de tendencias y recolección selectiva de trazas. Identifica así patrones de actividad, detectando anomalías mediante estos mismos patrones.

Esta herramienta cuenta con una interfaz web intuitiva, por lo que puede ser manejada sin demasiadas complicaciones. No obstante, no está enfocada tanto a la captura y reproducción de tráfico de red sino a su análisis, por lo que se desvía un poco del propósito planteado. Por último, es un producto que requiere ser contratado, siendo por tanto su público potencial más reducido el del resto de herramientas analizadas.

2.3. Conclusiones

Las alternativas estudiadas son herramientas técnicas, en las que la experiencia del usuario no es una prioridad. Así, excepto en *Detect-Pro*, se han detectado deficiencias en las interfaces del resto (de hecho, dos se manejan por línea de comandos). La mayoría de soluciones monitorizan el proceso de captura y reproducción desde el mismo servidor que lo realiza, lo que presenta algunas desventajas respecto a hacerlo de forma externa. Por ejemplo, dificulta conocer el estado del servidor en caso de saturación del mismo.

A raíz del análisis realizado, se considera necesario plantear una arquitectura distinta a las ya expuestas, en la que se separe los componentes de captura y reproducción de la propia interfaz. Con ello se posibilitaría que cada una de estas dos partes estuviese alojada en un servidor distinto, lo que facilitaría la monitorización de caídas y bloqueos del proceso interno de captura y reproducción. Respecto a la experiencia del usuario final, se cree conveniente diseñar una interfaz gráfica que simplifique las tareas más comunes dentro de este sistema y que resuelva las carencias presentes en las existentes. Adicionalmente, esta interfaz debería agrupar también otros aspectos relevantes del sistema, como la gestión de trazas o el rendimiento de los discos utilizados para almacenarlas.

3

Definición del proyecto

En este capítulo se definirán y explicarán el alcance del proyecto, la metodología de desarrollo escogida y las herramientas utilizadas.

3.1. Alcance

Esta aplicación tiene como objetivo permitir, mediante una interfaz web, gestionar una sonda de red y conocer su estado actual. Asimismo, posibilitará manejar otros aspectos relacionados con la sonda, como el almacenamiento y gestión de las trazas capturadas. Estos aspectos son comunes a cualquier sistema de captura y reproducción de tráfico de red, por lo que se estructurará la aplicación de forma que generalice el problema dado, sirviendo como trabajo base para la creación de interfaces gráficas sobre sondas similares.

No se pretende sin embargo, dentro del contexto de este proyecto, que la interfaz web sea capaz de interactuar con cualquier tipo de sonda de captura y reproducción de tráfico de red, sino solo con la sonda seleccionada. Tampoco entra dentro del alcance de este proyecto modificar el funcionamiento interno de la sonda, ni siquiera para mejorar su rendimiento, ya que el objetivo principal es facilitar la gestión de una sonda de red de altas prestaciones y de otros componentes que intervienen en el sistema.

3.2. Metodología

Para el desarrollo de la aplicación se ha optado por seguir un ciclo de vida en cascada con retroalimentación. Este modelo ordena las etapas del proceso de desarrollo software,

de forma que solo se pueda iniciar una fase cuando se ha finalizado la anterior. Dada la naturaleza de este proyecto, era necesario un periodo amplio de estudio del problema a resolver antes de poder codificar nada, siendo por ello este modelo más recomendable que adoptar alguna metodología ágil. Por otra parte, se ha descartado utilizar un modelo iterativo o en espiral dado que conllevaría un mayor tiempo de desarrollo al tener que realizarse por módulos y de forma separada cada una de las fases definidas, en vez de simultáneamente y de forma global. Además, el hecho de tener retroalimentación permite volver a etapas anteriores en caso de que sea necesario corregir algún aspecto del sistema.

La distribución temporal de las tareas y sus dependencias se resumen en el diagrama de Gantt de la Figura 3.1. A continuación se detallan la entradas, tareas y salidas de cada una de las fases del desarrollo del proyecto.

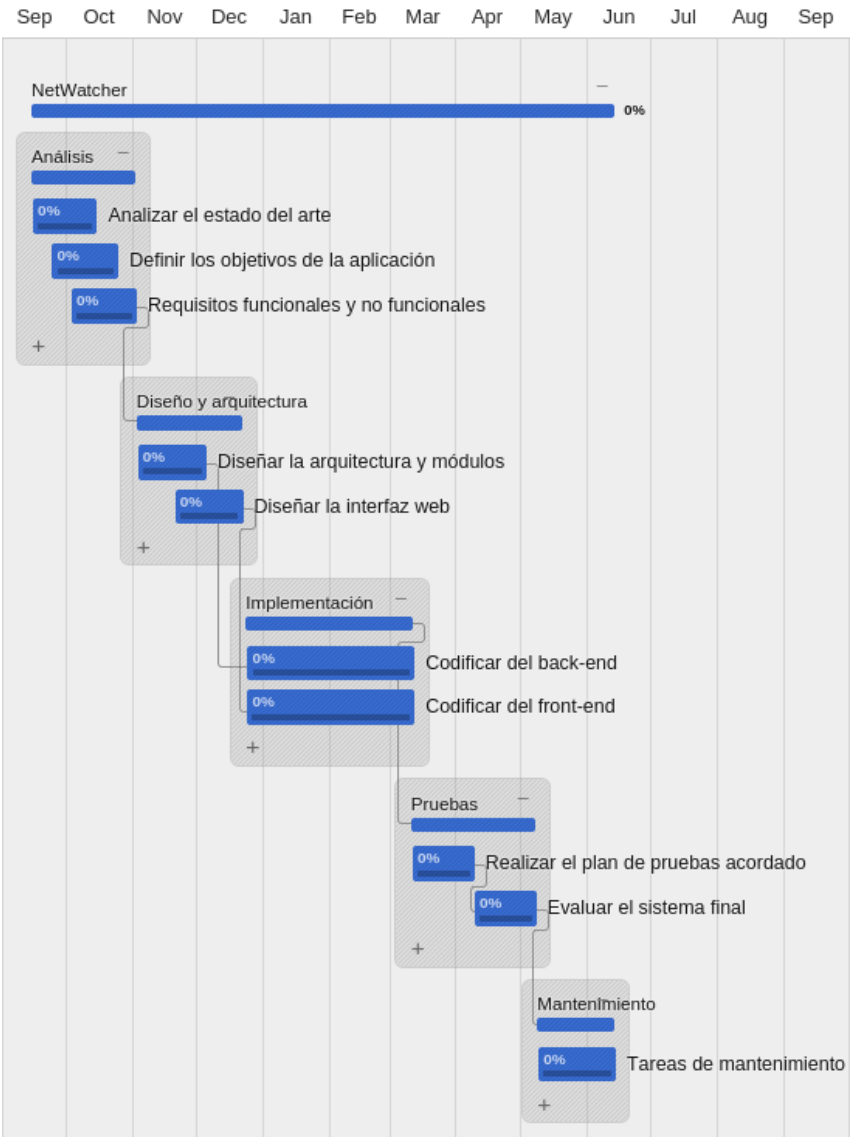


Figura 3.1: Diagrama de Gantt de la planificación temporal del proyecto.

Análisis

Tareas

- Analizar el estado del arte.
- Definir los objetivos de la aplicación.
- Especificar los requisitos funcionales y no funcionales.

Salida

- Definición, objetivos y alcance del proyecto.
- Relación de requisitos.

Diseño y arquitectura

Entrada

- Relación de requisitos.

Tareas

- Diseñar la arquitectura y módulos a implementar.
- Diseñar la interfaz web.

Salida

- Arquitectura de la aplicación.
- Diagramas de diseño.
- Maquetas de la interfaz web.

Implementación

Entrada

- Información sobre la arquitectura y el diseño de la aplicación.
- Maquetas de la interfaz web.

Tareas

- Codificar del back-end.
- Codificar del front-end.

Salida

- Código de la aplicación.
- Documentación del código de la aplicación.

Pruebas

Entrada

- Código de la aplicación.

Tareas

- Realizar el plan de pruebas acordado.
- Evaluar el sistema final.

Salida

- Código de la aplicación validado y verificado.

Mantenimiento

Entrada

- Código de la aplicación validado y verificado.

Tareas

- Analizar, implementar y probar las solicitudes de mejoras propuestas por usuarios.

Salida

- Código de la aplicación validado y verificado, con cambios menores propuestos por usuarios.

3.3. Herramientas

Para el desarrollo de este proyecto han sido necesarias herramientas que cubran las siguientes necesidades:

- Control de versiones.
- Creación de diagramas y maquetas.
- Documentación de la aplicación.
- Plataforma base para el back-end.
- Plataforma base para el front-end.

A continuación se especifican las herramientas elegidas, exponiendo su utilidad. Se detallan también otras librerías externas de las que hace uso la aplicación.

Control de versiones: GitHub

En todo proyecto software es fundamental, especialmente si se alarga en el tiempo, hacer uso de una herramienta de control de versiones para el código y la documentación. Se ha elegido con este propósito utilizar la plataforma *GitHub* [9], basada en *git* [10], un sistema distribuido de control de versiones. Esta plataforma es la más popular dentro de las herramientas de control de versiones, y tiene algunas ventajas importantes respecto a otros sistemas similares.

En primer lugar, ofrece alojamiento gratuito para proyectos de código abierto, y también para proyectos privados si se es estudiante. Gracias a esto se ha podido desarrollar todo el código de la aplicación en un proyecto privado, liberándolo al público al finalizar el desarrollo principal, de forma que cualquiera pueda utilizar y mejorar el código existente. Por otra parte, *GitHub* añade a la funcionalidad de *git* la posibilidad de crear una *wiki* del proyecto de forma sencilla, característica que ha sido utilizada en el proyecto. Finalmente, facilita la colaboración entre desarrolladores con una interfaz intuitiva y cuya curva de aprendizaje no tiene una una pendiente demasiado elevada.

Creación de diagramas y maquetas: Cacoo

Para el diseño de la aplicación se han realizado diagramas de flujo y de arquitectura del proyecto, así como maquetas de las diferentes pantallas de la interfaz web. Para ello, se ha utilizado la herramienta *Cacoo* [11], ya que ofrece una licencia gratuita para estudiantes que permite exportar estos gráficos en formato vectorial *svg*, que se pueden redimensionar sin pérdida de resolución. Otra característica interesante es que está basada en tecnologías web, con lo que es accesible desde cualquier navegador, sin ser necesario instalar ningún programa adicional.

Documentación: phpDocumentor y apiDoc

Con el objetivo de documentar la aplicación, se buscó una librería que contase con características particulares. Por un lado, que permitiese crear la documentación mediante anotaciones en el propio código, sin ralentizar demasiado la implementación del proyecto. Por otro, que generase la documentación en formato HTML, para que se pudiese acceder a ella del mismo modo que a la aplicación, desde un navegador. Debido a diferencias significativas (en arquitectura y lenguaje) entre el back-end y el front-end, se ha decidido finalmente utilizar una herramienta de documentación distinta para cada parte.

Para el back-end se ha elegido *apiDoc* [12], ya que es multilenguaje y encaja perfectamente dentro de la arquitectura interna (ver sección 5.1). Respecto al front-end, se ha seleccionado *phpDocumentor* [13], al tener una sintaxis similar a *javadoc*, herramienta utilizada en asignaturas del grado.

Plataforma base para el back-end: node.js

Se ha seleccionado *node.js* [14] como framework back-end. Esta plataforma de código abierto se ha considerado idónea para el proyecto por diversos motivos. Para empezar, utiliza *JavaScript* [15], lenguaje conocido por el estudiante. El hecho de que esté en este lenguaje permite además que se reutilice código entre el back-end y el front-end, ya que es el empleado por los navegadores web. Otra ventaja es que detrás de *node.js* existe una comunidad enorme, por lo que existen multitud de librerías también de código abierto disponibles y bien documentadas. Por último, es un framework de programación asíncrona (en la que no se tenía experiencia), por lo que su aprendizaje ha sido muy enriquecedor.

Librerías utilizadas para el back-end

Se han utilizado las siguientes librerías back-end de código abierto para *node.js*:

- **Express** [16]: framework minimalista para aplicaciones web con arquitectura REST.
- **Async** [17]: módulo que proporciona funciones para trabajar asíncronamente en *JavaScript*.
- **nodemon** [18]: supervisor que monitoriza cambios en el código de la aplicación y reinicia el servidor automáticamente.

Plataforma base para el front-end: framework propio

Se ha optado por desarrollar un framework propio en PHP como plataforma base para el front-end (ver apéndice B). Esta decisión está fundamentada en varios motivos. Por un lado, no se quería utilizar un framework que tuviese funcionalidades no necesarias para

esta aplicación concreta, y cuya curva de aprendizaje ralentizase el proyecto. Otra razón es que conocer al detalle el framework utilizado ha proporcionado una mayor flexibilidad en el proceso de desarrollo, pudiendo además modificar la estructura y arquitectura del mismo para que se adecuase perfectamente a las necesidades propias. Finalmente, se contaba ya con cierta experiencia programando en PHP, por lo que ha sido el lenguaje elegido.

Librerías utilizadas para el front-end

Además del framework desarrollado, se han utilizado diversas librerías front-end, todas ellas de código abierto. A continuación se enumeran, describiendo brevemente su propósito:

- **Bootstrap** [19]: facilita el desarrollo de aplicaciones web *responsive* [20] mediante plantillas de diseño con tipografía, formularios, botones, cuadros y menús.
- **Bootstrap table** [21]: mejora las tablas de *Bootstrap* permitiendo de manera sencilla insertar un campo de búsqueda, filtrar filas por *checkbox* o *radio button*, ordenar por columnas, paginar automáticamente los resultados, etc.
- **Bootswatch** [22]: colección de temas visuales para *Bootstrap*.
- **Bootstrap Notify** [23]: convierte los avisos de *Bootstrap* en notificaciones emergentes.
- **jQuery** [24]: simplifica la manipulación de documentos HTML, el manejo de eventos y las llamadas AJAX.
- **Chart.js** [25]: permite realizar gráficos simples y atractivos sobre conjuntos de datos.
- **Animate.css** [26]: sencillas animaciones para elementos de la interfaz web.

4

Requisitos

En este capítulo se enumeran los requisitos de la aplicación a desarrollar. Para la elaboración de esta lista de requisitos se ha realizado un análisis sobre el problema planteado: diseñar un servicio que permita gestionar y monitorizar una FPGA que captura y reproduce tráfico de red. Este análisis se ha realizado principalmente mediante la consulta directa con los potenciales usuarios de la aplicación y la evaluación del estado del arte.

Se han agrupado los requisitos en dos clases: funcionales y no funcionales. Los primeros describen el comportamiento que tendrá la aplicación, y los segundos los atributos de calidad y restricciones de la misma.

4.1. Requisitos Funcionales

Los requisitos funcionales que deberá cumplir la aplicación a desarrollar son los siguientes:

RF. 1 *Se podrá conocer en todo momento el estado actual de la FPGA.*

RF. 2 *Se podrá configurar la FPGA para la captura de tráfico de red.*

RF. 3 *Una vez configurada la FPGA para la captura de tráfico de red, se le podrá ordenar que capture tráfico de red desde un puerto específico de la FPGA. Este tráfico se irá guardando en una traza en formato simple, hasta llegar a un tamaño establecido por el usuario.*

RF. 4 *Si existe una captura en curso, se podrá parar dicha captura, borrándose la traza asociada a la captura.*

RF. 5 *Si existe una captura en curso, se podrán conocer los parámetros con los que se inició dicha captura, así como el tiempo que ha transcurrido desde el inicio y cuántos bytes se ha capturado hasta el momento.*

RF. 6 *Se podrá configurar la FPGA para la reproducción de una traza.*

RF. 7 *Una vez configurada la FPGA para la reproducción de una traza, se le podrá ordenar que reproduzca una traza concreta en formato simple. La reproducción se realizará con una una serie de parámetros dados por el usuario: máscara de puertos a los que dirigir la reproducción, IFG asociado y reproducir en bucle o solo una vez.*

RF. 8 *Si existe una reproducción de traza en curso, se podrá parar dicha reproducción.*

RF. 9 *Si existe una reproducción de traza en curso, se podrán conocer los parámetros con los que se inició dicha reproducción, así como el tiempo que ha transcurrido desde el inicio y cuántos paquetes se han reproducido hasta el momento.*

RF. 10 *Se podrá configurar y consultar en qué directorio se almacenan las trazas.*

RF. 11 *Se podrá conocer la lista de trazas existentes, así como su tamaño, fecha y tipo (simple o pcap).*

RF. 12 *Se podrá filtrar la lista de trazas existentes según su tipo.*

RF. 13 *Se podrá ordenar la lista de trazas existentes por nombre, tipo, tamaño o fecha.*

RF. 14 *Se podrán buscar trazas por su nombre.*

RF. 15 *Una traza en formato simple podrá ser convertida a formato pcap.*

RF. 16 *Una traza en formato pcap podrá ser convertida a formato simple.*

RF. 17 *Una traza podrá ser renombrada.*

RF. 18 *Una traza podrá ser borrada.*

RF. 19 *Se podrán conocer el espacio total y el espacio ocupado del sistema de archivos que contiene las trazas.*

RF. 20 *Si el sistema de archivos que contiene las trazas es un RAID, se podrá conocer la velocidad de escritura global del RAID, así como la de cada disco que lo compone.*

RF. 21 *Si el sistema de archivos que contiene las trazas es un RAID, se podrá formatear y recrear el RAID.*

4.2. Requisitos No Funcionales

Los requisitos no funcionales que deberá cumplir la aplicación a desarrollar son los siguientes:

RNF. 1 *La funcionalidad descrita en la sección 4.1 será accesible automáticamente al arrancar el servidor conectado a la sonda de red.*

RNF. 2 *La funcionalidad descrita en la sección 4.1 será accesible mediante una interfaz gráfica.*

RNF. 3 *Se podrán seleccionar al menos dos idiomas para la interfaz gráfica: inglés y español.*

RNF. 4 *Se podrán seleccionar distintos temas (aspectos visuales) para la interfaz gráfica.*

RNF. 5 *La interfaz gráfica será una web y podrá ser visualizada en distintas resoluciones de pantalla, como las de ordenadores y móviles, con una experiencia de usuario similar.*

RNF. 6 *La interfaz gráfica estará disponible aun cuando se produzca algún fallo en el servidor que aloja la FPGA y este no se encuentre operativo, e informará del error al usuario.*

RNF. 7 *La interfaz gráfica podrá ser configurada para conectarse a distintas sondas de red (todas del mismo modelo).*

RNF. 8 *Toda acción del usuario en la interfaz gráfica tendrá una respuesta visual que indique el resultado de la misma.*

5 Diseño

En este capítulo se describe el diseño de la aplicación a desarrollar. Tras analizar los requisitos especificados en el capítulo 4, se ha decidido dividir la aplicación en dos partes, alojadas cada una en un servidor distinto: una interfaz web (front-end) y un servicio web (back-end). Estos componentes están conectados por una red interna, y la comunicación entre ellos se realiza mediante llamadas HTTP (ver Figura 5.1).

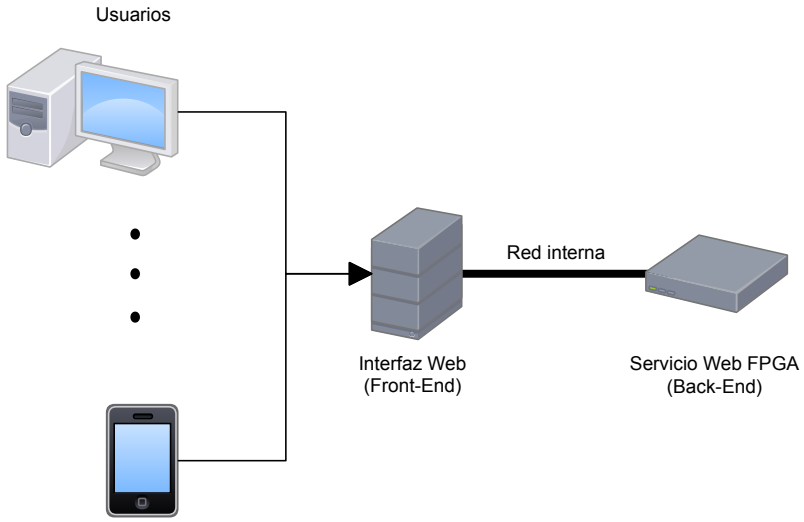


Figura 5.1: Arquitectura general de la aplicación.

La arquitectura propuesta tiene una serie de ventajas respecto a tener todos los elementos del sistema en un mismo servidor físico. En primer lugar, se sobrecarga menos el servidor de la sonda de red, minimizando así el impacto que la aplicación pueda tener

sobre el rendimiento de la captura y reproducción. En segundo lugar, una división clara entre el back-end y el front-end facilita la adopción de tecnologías distintas en sendos componentes, utilizando en cada uno las que mejor se adapten al problema dado, y sin miedo a incompatibilidades (pues se comunican entre ellos por HTTP, que es estándar). En tercer lugar, se posibilita el gestionar desde un mismo front-end distintas sondas de red que tengan instalado el mismo back-end, sin que el usuario tenga que cambiar de plataforma. Por último, al estar alojados en servidores distintos, la interfaz web podrá informar siempre al usuario del estado del sistema incluso cuando el servicio web no esté disponible.

5.1. Back-End - Servicio Web FPGA

El componente back-end se encarga de la interacción con la sonda de red (implementada en una FPGA) y con el resto de partes involucradas en la reproducción y captura de tráfico de red. Para ello, recibe peticiones HTTP del front-end (actuando en este caso como cliente), que se traducen en acciones sobre el sistema o en respuestas sobre el estado del mismo.

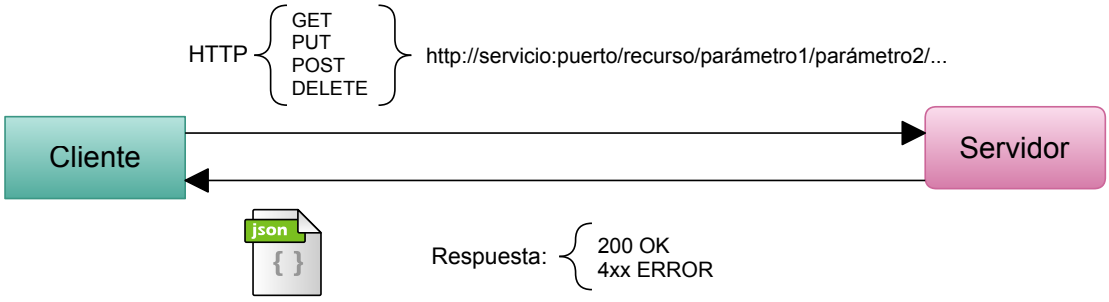


Figura 5.2: Diagrama de flujo de un servicio REST.

La arquitectura de comunicación externa del back-end se basa en el modelo de cliente-servidor REST (ver Figura 5.2). Dentro de las directrices que marca este modelo, solo se han considerado útiles para el problema dado un subconjunto de ellas:

- El protocolo entre el cliente y el servidor debe ser sin estado: cada mensaje HTTP tiene que contener toda la información necesaria para comprender la petición.
- Las operaciones se aplican sobre recursos mediante llamadas a métodos HTTP: *GET* para obtener información sobre un recurso, *POST*/*PUT* para actualizarlos o crearlos y *DELETE* para borrarlos.
- Cada recurso debe tener un identificador único (en este caso, una URL única).

Se ha decidido no adoptar el resto de directrices REST debido a que no encajaban dentro del modelo de funcionamiento de la aplicación. Así, no se facilita el descubrimiento

automático de recursos y métodos, ya que se ha considerado que no tiene sentido siendo éste un subsistema interno y no un componente público. Por otra parte, no se permiten distintas representaciones de un mismo recurso, siendo JSON la única representación utilizada. No seguir estas directrices simplifica además la implementación del back-end.

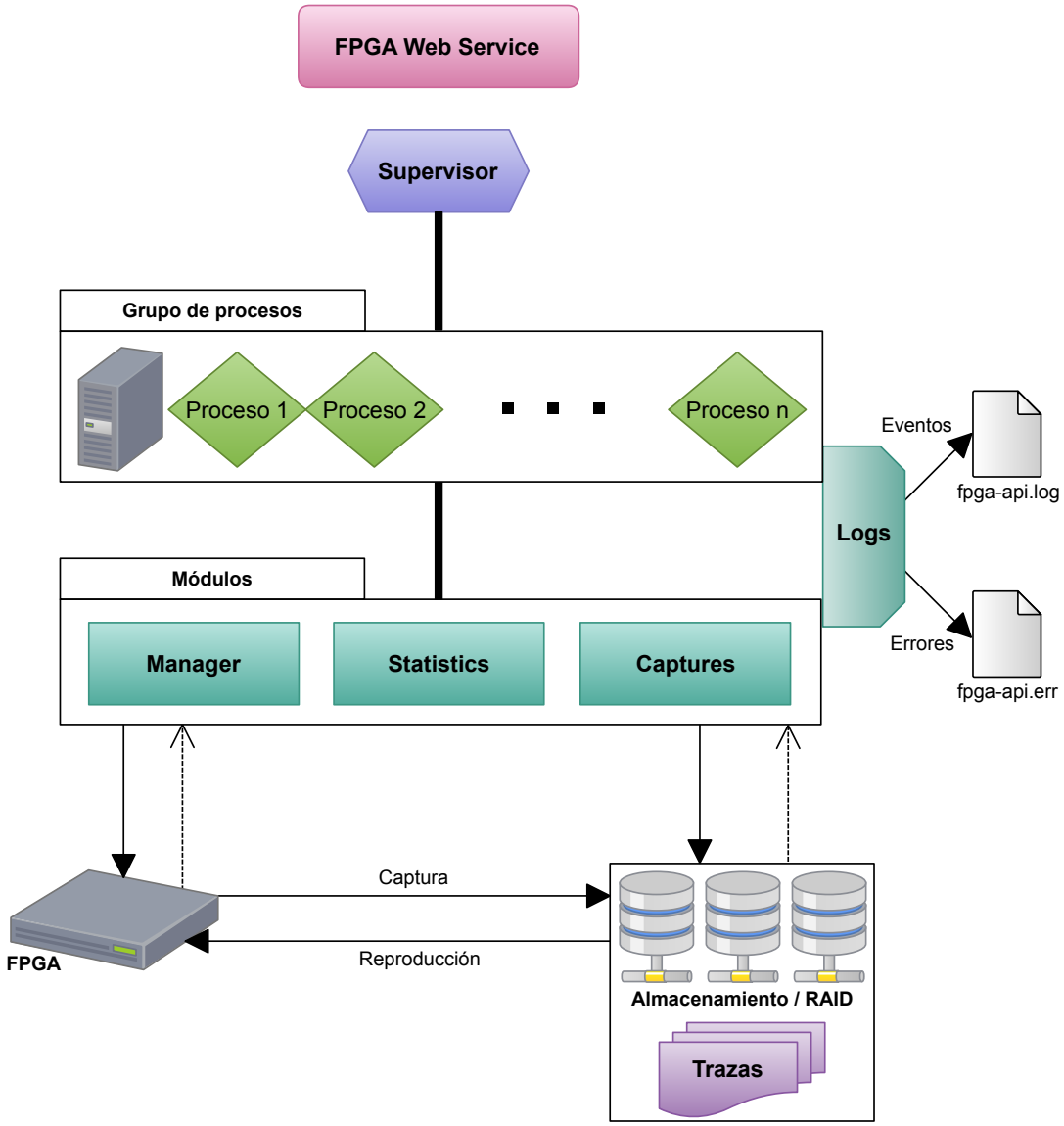


Figura 5.3: Arquitectura del Servicio Web FPGA.

Internamente, el back-end se estructura tal y como se describe en la Figura 5.3. Un supervisor se encarga de vigilar al grupo de procesos que atienden las peticiones del front-end, reemplazándolos en caso de que fallen. Este grupo tiene tantos procesos como núcleos el servidor, adaptándose así a su arquitectura interna y consiguiendo por tanto una mayor disponibilidad y un menor tiempo de respuesta. Por otro lado, con el objetivo de tener información detallada sobre el uso del servicio web, se registran todos los eventos y errores del back-end en los logs correspondientes. Por último, se ha dividido el servicio web en tres módulos, cada uno con una funcionalidad asociada: *manager*, *captures* y *statistics*.

5.1.1. Manager

Este módulo se encarga de gestionar el estado de la sonda de red y del servidor que la aloja. Permite por tanto instalar, programar y montar la sonda en modo reproducción o en modo captura, ordenarle reproducir o capturar tráfico y pararla. Adicionalmente, maneja también otros aspectos del sistema, posibilitando reiniciar el servidor y formatear el sistema de almacenamiento en caso de ser necesario.

5.1.2. Captures

Este módulo se ocupa de todos los aspectos relacionados con las trazas de tráfico de red. Permite así listar todas las trazas disponibles, mostrando su nombre, fecha, tipo y tamaño. Por otra parte, sobre una traza concreta es capaz de detectar el formato interno de la misma, convertirla entre los formatos soportados, renombrarla o borrarla para liberar espacio de almacenamiento.

5.1.3. Statistics

Este módulo tiene como objetivo informar sobre el estado actual de la sonda de red, y proporcionar estadísticas sobre la reproducción o captura (en caso de existir una en curso). Permite conocer además medidas sobre el almacenamiento: espacio ocupado, espacio disponible, velocidad de escritura global y por discos.

5.2. Front-End - Interfaz web

El front-end se encarga de mostrar una interfaz web al usuario, recoger las acciones de éste y transformarlas en peticiones HTTP al back-end, informando de forma visual del resultado de la solicitud. Este componente utiliza como punto de partida el framework propio desarrollado (ver apéndice B). Dicho framework provee de una arquitectura y funcionalidad base a la interfaz, estructurando además los módulos de la misma siguiendo el patrón modelo-vista-controlador.

En el apartado visual, se ha decidido adoptar un diseño *responsive*. Esta filosofía de diseño se basa en la idea de que se debe adaptar la forma de mostrar una página para que la experiencia del usuario sea óptima independientemente del dispositivo que utilice (móvil, ordenador de sobremesa, etc.). Para ello se utilizan una cuadrícula que cambia de posición y tamaño según sea el dispositivo desde el que se accede a la interfaz. Ésta filosofía de diseño tiene una diferencia fundamental respecto al diseño *adaptative*, que simplifica su implementación: mientras este último se basa en cargar distintos recursos de estilo según las características del dispositivo, el diseño *responsive* utiliza una única cuadrícula que se ajusta en base a dichas características.

Respecto al idioma de la interfaz web, ésta estará disponible tanto en inglés como en español, pudiendo el usuario elegir el idioma que prefiera. De forma similar, también se podrá seleccionar un tema visual para el front-end. Estos temas cambian el esquema de colores y aspectos menores de la interfaz gráfica.

5.2.1. Maquetas

En este apartado se exponen las maquetas de las páginas principales de la interfaz, que sirven como guía para la implementación del front-end. Todas ellas cuentan con una barra superior de navegación, con enlaces al resto de pantallas principales de la interfaz.

En la Figura 5.4 se muestra la maqueta de la pantalla de configuración de la aplicación. En esta página se podrán configurar, mediante un formulario, la dirección del Servicio Web FPGA (el back-end), el idioma y el tema visual de la interfaz.

Figura 5.4: Maqueta de la pantalla de configuración de la aplicación.

En la Figura 5.5 se muestra la maqueta de la pantalla de almacenamiento. Esta página mostrará gráficas sobre el espacio disponible, y sobre estadísticas del RAID en caso de estar configurado. También se dará opción a formatear el RAID en caso de que no tenga un rendimiento aceptable.

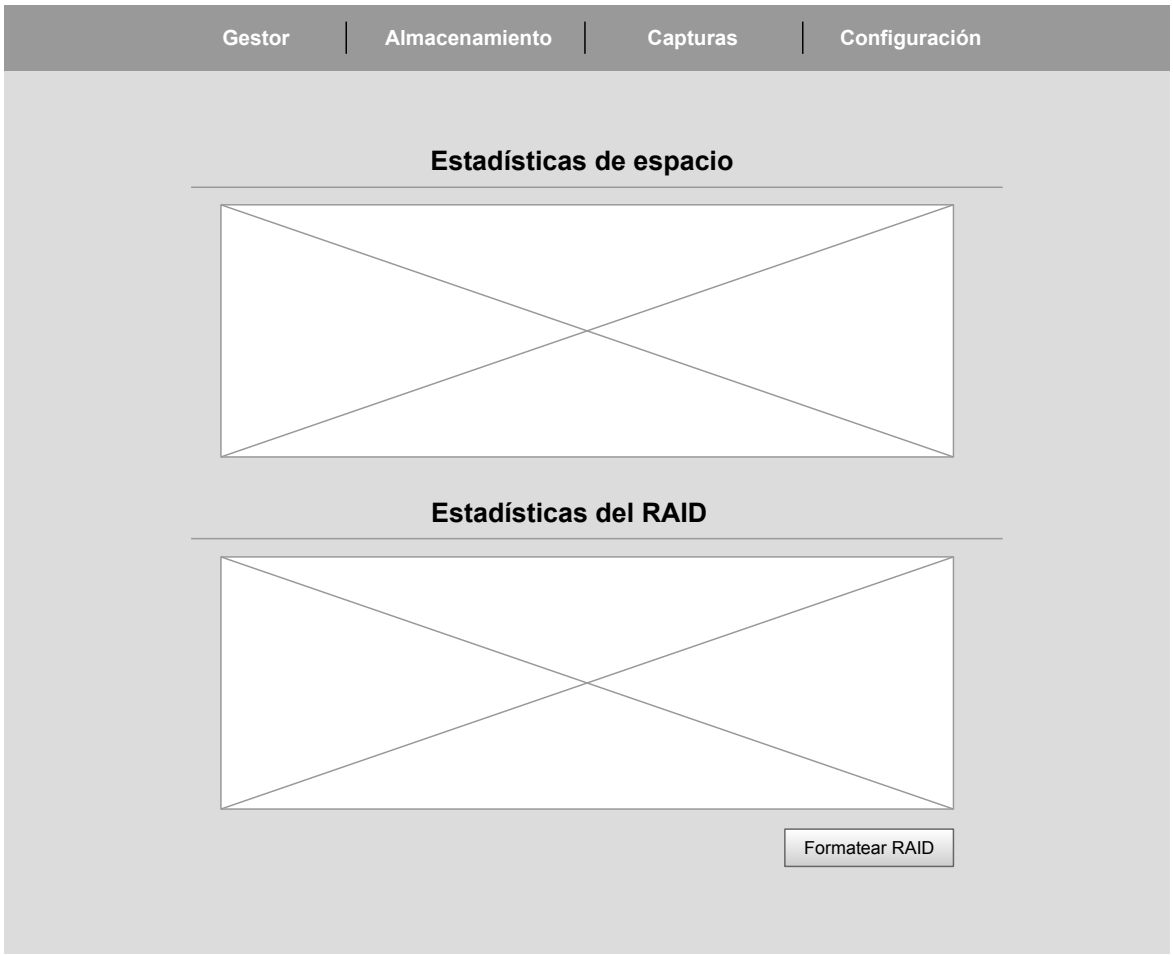


Figura 5.5: Maqueta de la pantalla de almacenamiento.

En la Figura 5.6 se muestra la maqueta de la pantalla de gestión de trazas. A la izquierda se podrán visualizar, en una tabla, todas las trazas disponibles, con su nombre, tipo, tamaño y fecha. Sobre esta tabla, una barra de acciones permitirá filtrar las trazas según su tipo, o buscar alguna concreta por su nombre. A la derecha se mostrará un panel con posibles acciones a realizar sobre la traza seleccionada de la tabla: convertirla a otro formato, renombrarla o borrarla.

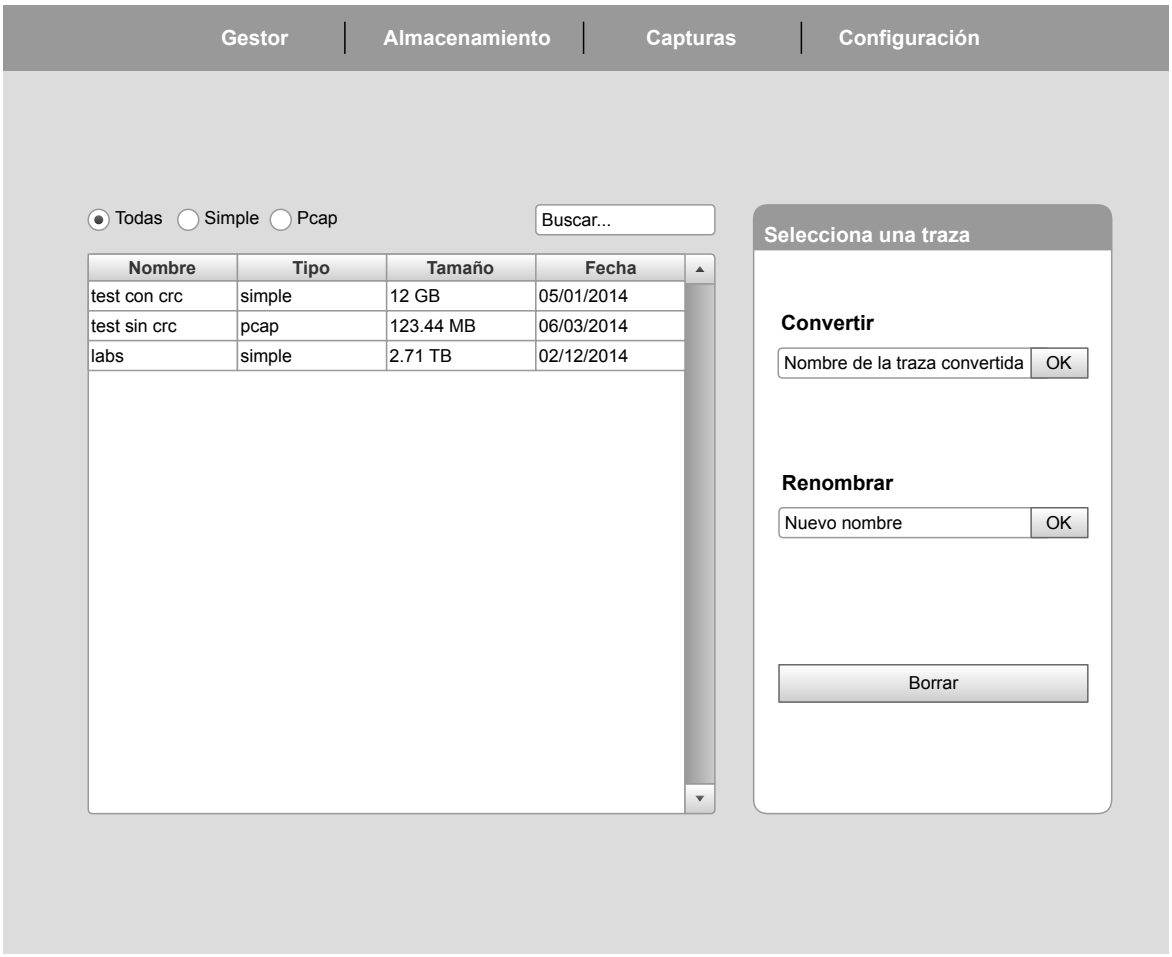


Figura 5.6: Maqueta de la pantalla de gestión de trazas.

En la Figura 5.7 se expone la maqueta de una de las pantallas de la página de gestión, que se mostrará en caso de que no se haya seleccionado aún ningún modo o cuando se quiera seleccionar otro. Esta pantalla consta de dos botones, y pulsando alguno se inicializará la sonda en el modo correspondiente.

Figura 5.7: Maqueta de la pantalla de gestión - selección de modo.

En la Figura 5.8 se expone la maqueta de una de las pantallas de la página de gestión, que se mostrará cuando la sonda haya sido inicializada en modo captura. En esta pantalla se podrá rellenar un formulario con el nombre de la traza a capturar, su tamaño y el puerto del que capturar. Pulsando el botón *Capturar* se iniciará la captura de dicha traza. También se podrá volver a la página de selección de modo, pulsando el enlace inferior *Cambiar de modo*.

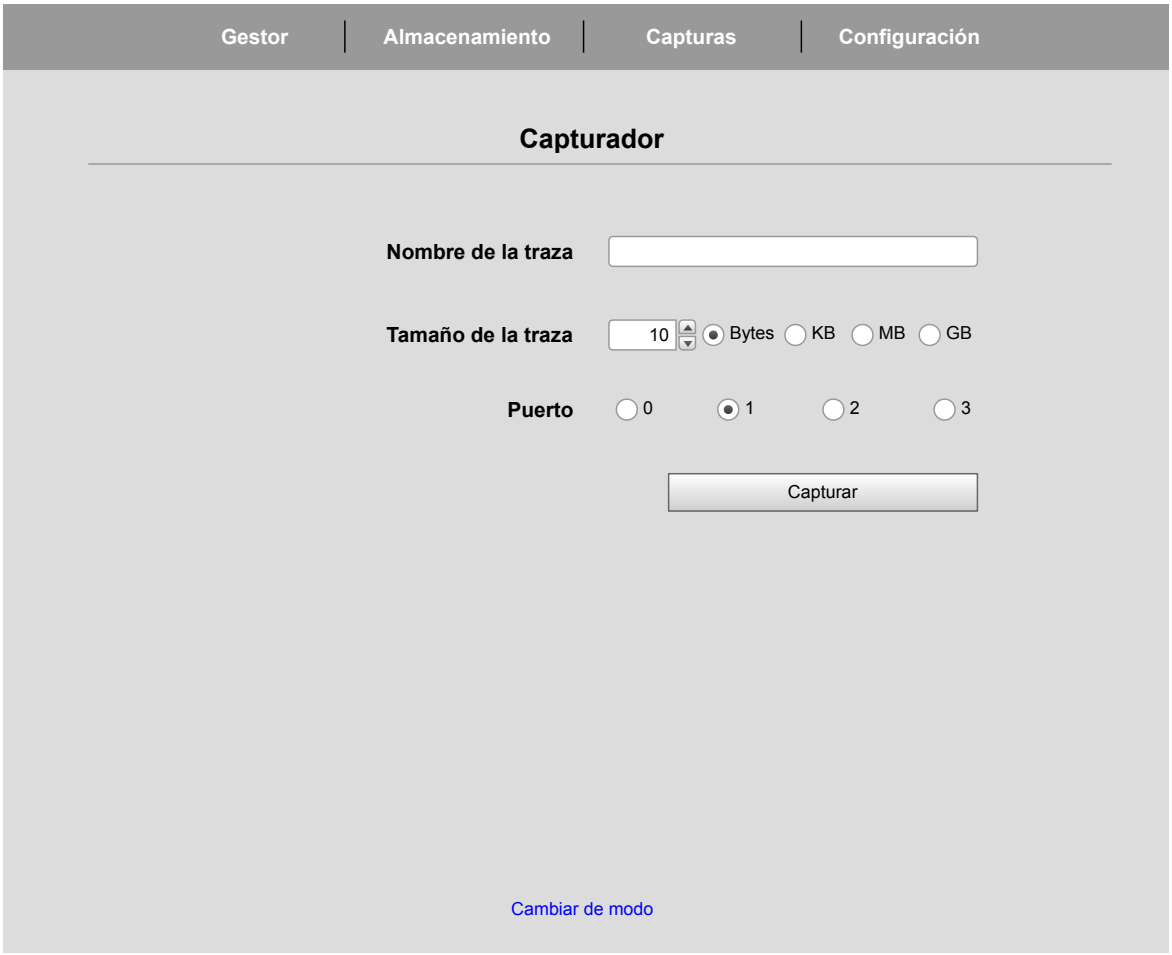


Figura 5.8: Maqueta de la pantalla de gestión - formulario para capturar.

En la Figura 5.9 se expone la maqueta de otra de las pantallas de gestión, que se mostrará cuando la sonda esté capturando tráfico de red. En esta pantalla se podrán visualizar distintas estadísticas de la captura en curso, hasta que ésta finalice. Adicionalmente, una barra de progreso indicará qué porcentaje de la traza ha sido ya capturado. También se podrá parar la captura, pulsando el botón *Detener la captura*.

Figura 5.9: Maqueta de la pantalla de gestión - capturando.

En la Figura 5.10 se expone la maqueta de otra de las pantallas de la página de gestión, que se mostrará cuando la sonda haya sido inicializada en modo reproducción. A la izquierda se podrán visualizar, en una tabla, todas las trazas disponibles para reproducir, con su nombre, tipo, tamaño y fecha. Sobre esta tabla, una barra de acciones permitirá filtrar las trazas según su tipo, o buscar alguna concreta por su nombre. A la derecha se mostrará un panel con un formulario para configurar las opciones de reproducción: en bucle o no, IFG y máscara de reproducción. Pulsando el botón *Reproducir* se iniciará la reproducción de la traza seleccionada. También se podrá volver a la página de selección de modo, pulsando el enlace inferior *Cambiar de modo*.

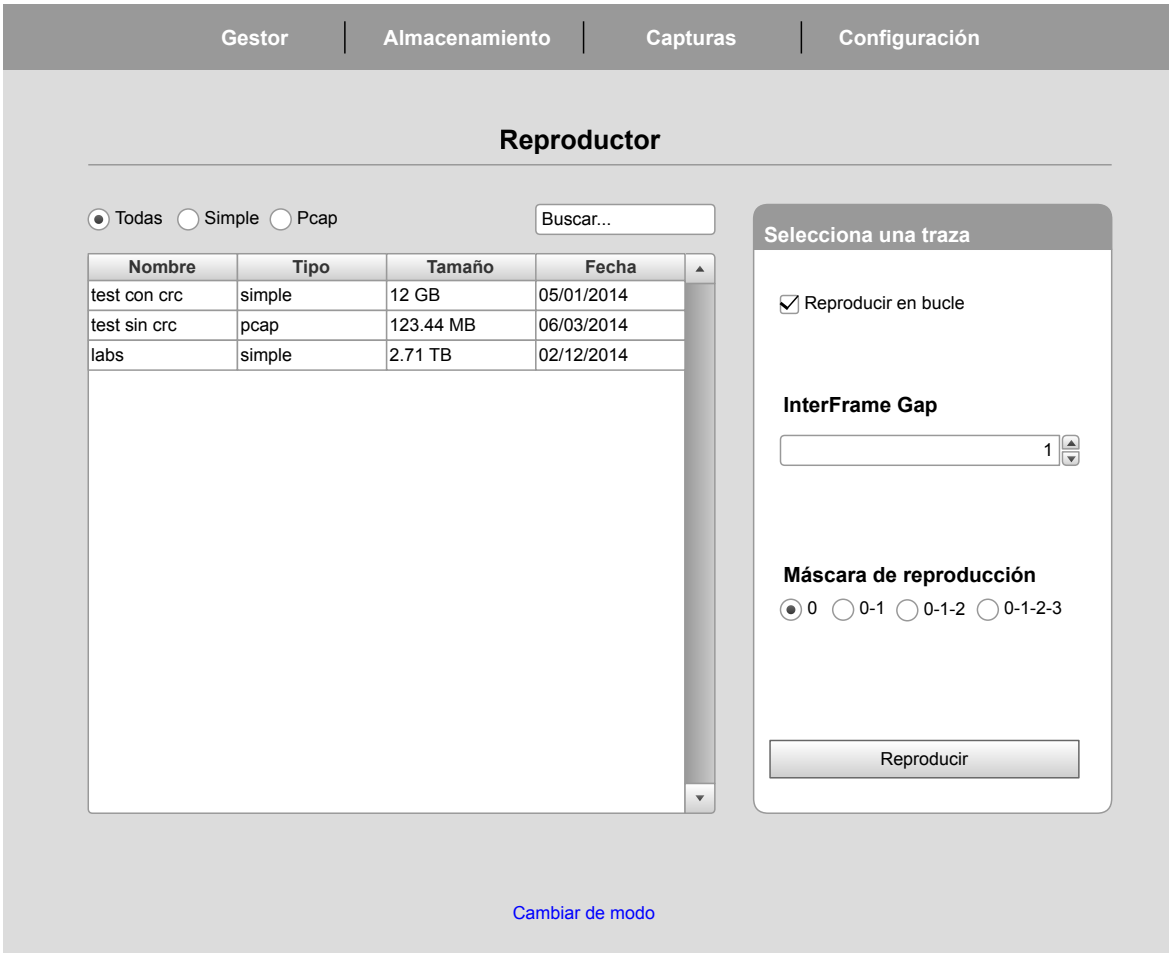


Figura 5.10: Maqueta de la pantalla de gestión - formulario para reproducir.

En la Figura 5.11 se expone la maqueta de otra de las pantallas de gestión, que se mostrará cuando la sonda esté reproduciendo una traza. En esta pantalla se podrán visualizar distintas estadísticas de la reproducción en curso, hasta que ésta finalice. También se podrá parar la reproducción, pulsando el botón *Detener la reproducción*.

Figura 5.11: Maqueta de la pantalla de gestión - reproduciendo.

6 Implementación

En este capítulo se explica cómo se han implementado los dos componentes principales de la aplicación, tal y como se diseñaron y estructuraron en el capítulo 5: un servicio web que se comunica con la sonda (back-end) y una interfaz web (front-end). Se detalla además qué documentación se ha generado en esta fase, tanto interna del proyecto como para el usuario final de la aplicación.

6.1. Back-End

Para la implementación del back-end se han utilizado las librerías en *JavaScript* mencionadas en la sección 3.3. Así, se ha creado una API REST con la ayuda de la librería *Express*, sobre el framework *node.js*. Adicionalmente, se hace uso de la librería *Async* para facilitar el manejo de funciones asíncronas, y de *nodemon* para cubrir la función de supervisor descrita en el diseño. Finalmente, se ha encapsulado el componente dentro de un servicio que se inicia automáticamente al arrancar el ordenador, haciendo que el back-end esté activo siempre que el servidor de la sonda de red se encuentre operativo.

El objetivo principal del back-end es formalizar el estado de la FPGA, permitiendo así su consulta y modificación. Para programar este aspecto, se han modelado los posibles estados internos del sistema en una Máquina de Estados Finitos (ver Figura 6.1). Las transiciones existentes entre los estados son:

[1] Iniciar el servidor sin seleccionar la opción de arrancar con *HugePages* habilitadas.

[2] Reiniciar el servidor, seleccionando la opción de arrancar con *HugePages* habilitadas.

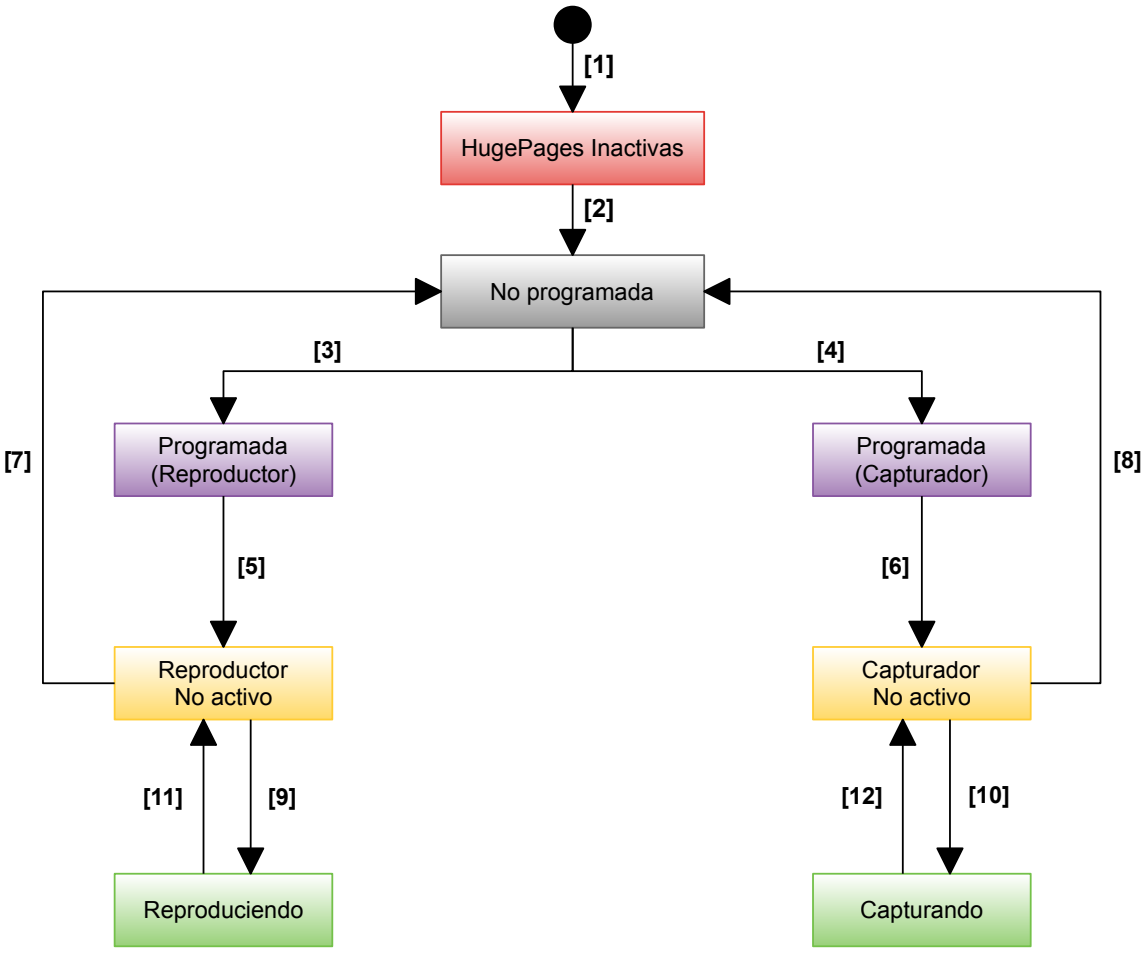


Figura 6.1: Máquina de Estados Finitos del back-end.

[3] Programar la FPGA en modo reproductor con el bitstream correspondiente y reiniciar el servidor.

[4] Programar la FPGA en modo capturador con el bitstream correspondiente y reiniciar el servidor.

[5] Montar la FPGA sobre el servidor.

[6] Montar la FPGA sobre el servidor.

[7] Reiniciar el servidor.

[8] Reiniciar el servidor.

[9] Ordenar a la sonda reproducir una traza.

[10] Ordenar a la sonda capturar una traza.

[11] Reproducción finalizada, o porque se ha acabado el contenido de la traza o porque ha sido detenida el usuario.

[12] Captura finalizada, o porque se ha capturado todo lo que se quería o porque es detenida por el usuario.

Para conocer el estado actual de esta máquina de estados y verificar qué operaciones puede realizar la FPGA, se plantearon dos alternativas: mantener sincronizadas la sonda de red y el servicio web, de forma que el servicio web detectase cualquier cambio en la FPGA y almacenase el estado de esta de forma persistente, o que cada vez que se necesitase conocer el estado de la sonda se determinase de nuevo. Se ha elegido la segunda opción, ya que evita que pueda haber incoherencias entre el estado real de la FPGA y el guardado en el servicio. Además, así el servicio web tendrá que consultar a la sonda solo en caso de que haya una petición dirigida al back-end, no utilizando recursos en la monitorización de la FPGA si no ha sido invocado. Se ha implementado, siguiendo esta segunda opción, un árbol de decisión binario que determina el estado actual de la FPGA mediante consultas a la propia sonda y al sistema (ver Figura 6.2).

Los archivos de código más relevantes de la implementación del back-end se muestran en la Figura 6.3a. El fichero de código principal es *server.js*, ya que se encarga de lanzar el grupo de procesos que responde a las peticiones del front-end, delegándolas según corresponda a uno de los tres módulos existentes. Cada uno de estos módulos consta de dos archivos en *JavaScript*. El primero, con el nombre del propio módulo, contiene las funciones de la API del back-end (ver Figura 6.4). El segundo fichero, con el nombre del módulo seguido de *_utils*, recoge funciones auxiliares. Por último, las funciones comunes a todos los módulos han sido codificadas en un único archivo, *_common.js*.

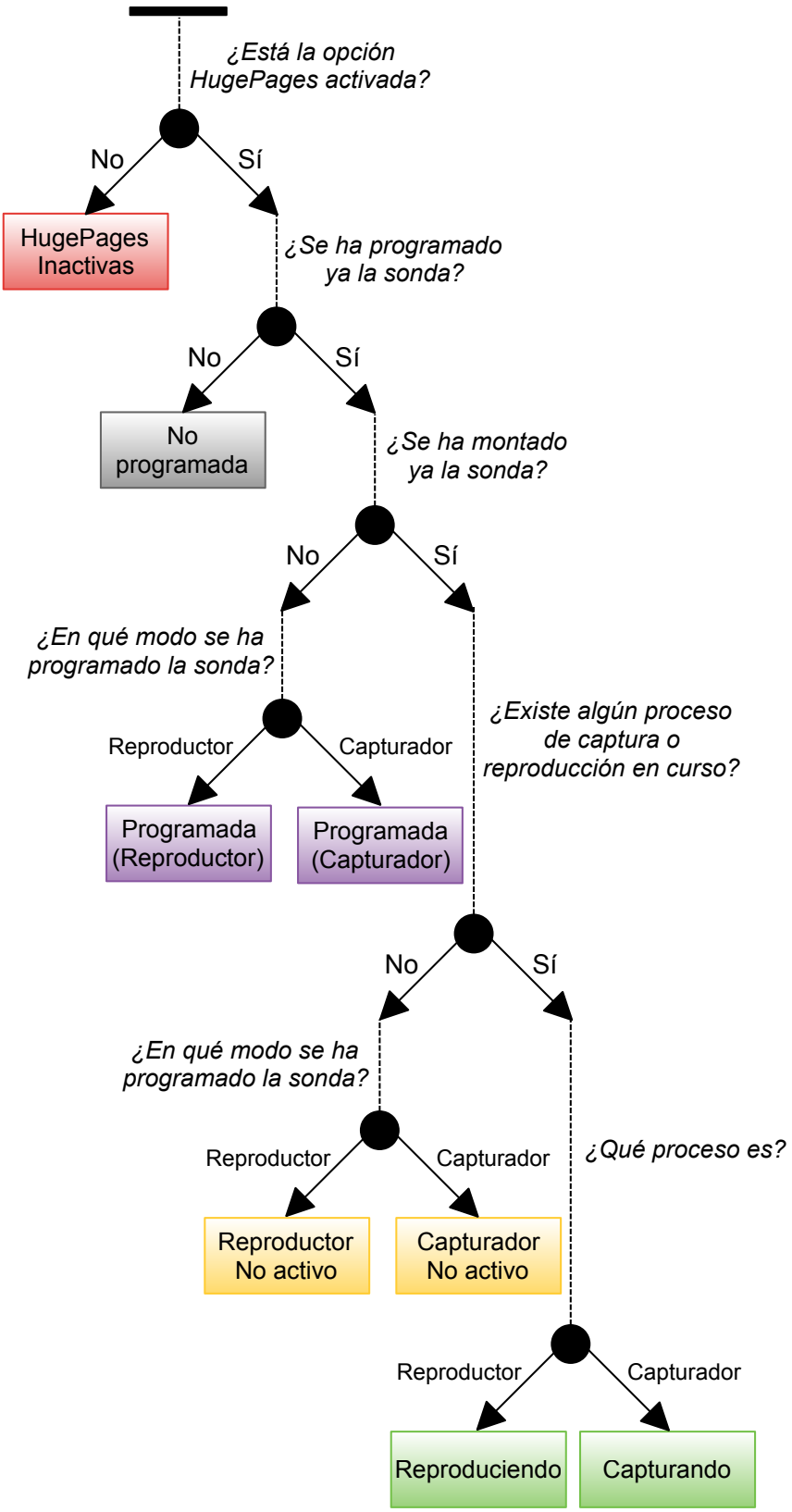
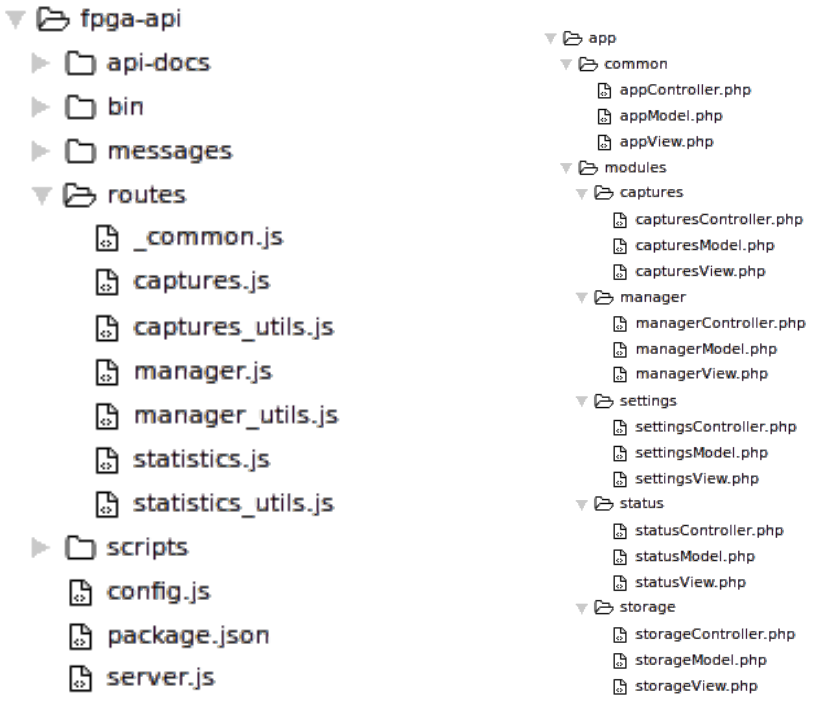


Figura 6.2: Árbol de decisión para determinar el estado de la FPGA.



(a) Árbol del back-end. (b) Árbol del front-end.

Figura 6.3: Árboles con los principales archivos de código.

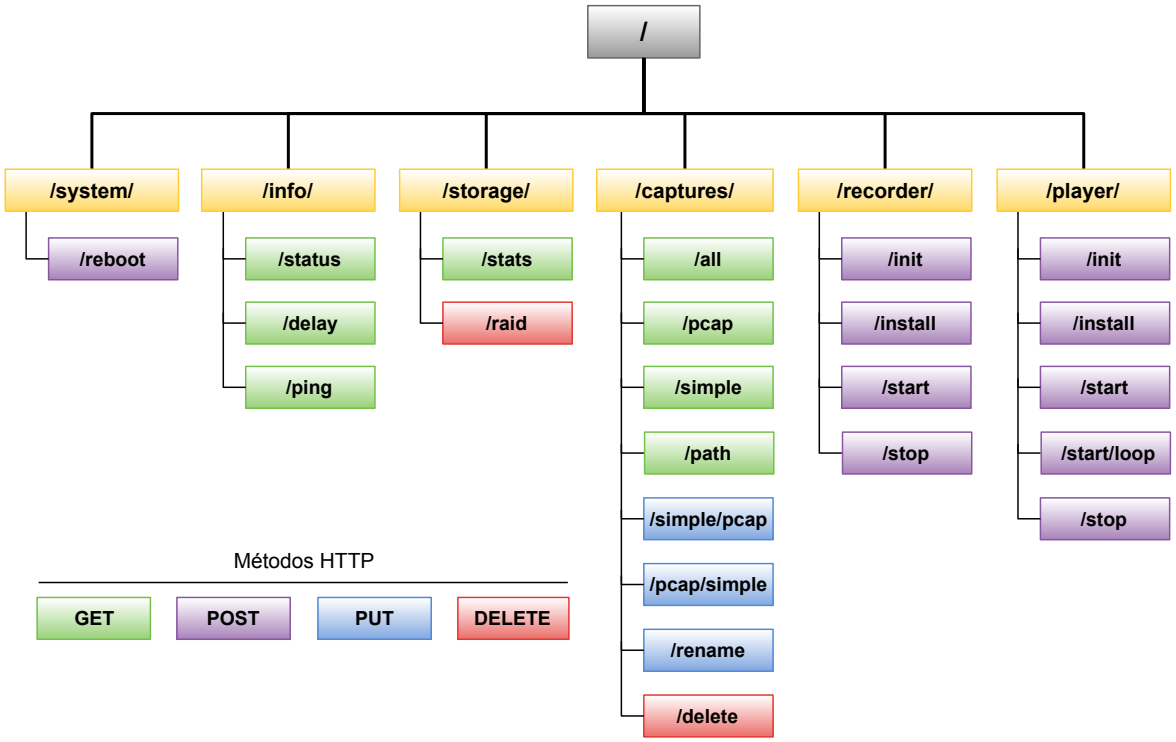


Figura 6.4: Métodos públicos del Servicio Web FPGA.

6.2. Front-End

Para la implementación del front-end se ha partido del framework propio desarrollado, utilizándose las herramientas en *JavaScript*, *CSS* y *HTML5* descritas en la sección 3.3. Así, se ha hecho uso de la librería *jQuery* para la gestión de los elementos dinámicos de la interfaz web y de *Chart.js* para la realización de gráficas sobre el rendimiento y el almacenamiento disponible.

Se ha codificado la interfaz gráfica siguiendo un diseño *responsive*, tal y como se comentó en la sección 5.2. Para ello se han empleado las librerías *Bootstrap* y *Bootstrap table* (extensión de *Bootstrap* que mejora las tablas). Se consigue así que se pueda manejar la sonda de red desde un móvil, no solo desde un ordenador convencional (ver Figura 6.5).

Figura 6.5: Página de la aplicación visualizada desde un dispositivo móvil.

Con la intención de facilitar al usuario el manejo de la interfaz, se ha traducido el front-end al español, quedando, por tanto, la interfaz disponible en dos idiomas: inglés y español. Para ello, se ha utilizado la librería *gettext*, incluida en el framework base, y se ha creado un catálogo con todas las cadenas de texto de la interfaz en español (proceso explicado en la sección B.4).

Otro aspecto que se ha considerado importante, desde el punto de vista de la usabilidad, es que el usuario pueda conocer en todo momento el estado del sistema. Con este objetivo en mente, toda acción del usuario tiene como respuesta una confirmación

visual del resultado de la misma. Las operaciones mayores implican una actualización o cambio de la página actual, mientras que para las acciones menores se han utilizado notificaciones emergentes, para las cuales se han utilizado las librerías *Bootstrap Notify* y *Animate.css*.

Como parte de la configuración que puede ajustar el usuario, se le permite seleccionar un estilo gráfico para la aplicación. Esta opción se ha implementado usando *Bootswatch*, una colección de temas para *Bootstrap* (ver ejemplo en Figura 6.6).

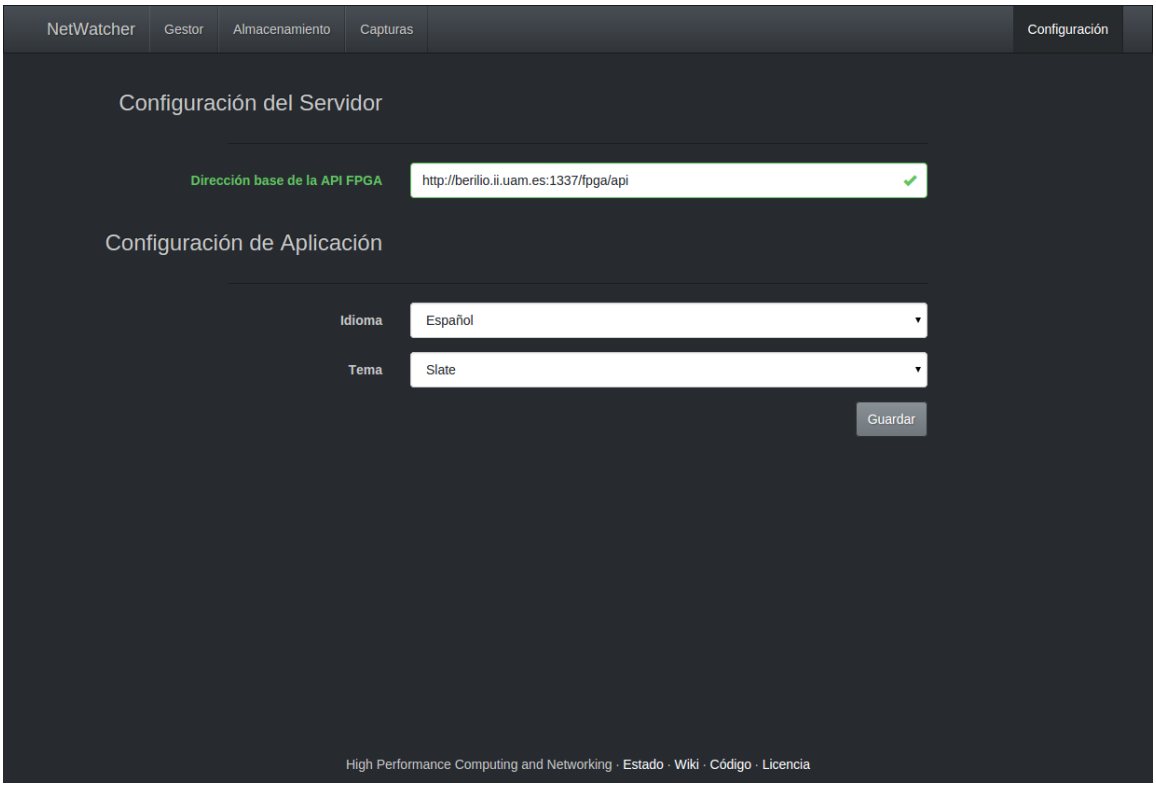


Figura 6.6: Página de la aplicación con un tema oscuro seleccionado.

Los submódulos (páginas) de la interfaz web se estructuran internamente siguiendo el modelo de arquitectura modelo-vista-controlador, implementado de forma abstracta en el módulo *common*. Los archivos de código más relevantes de esta implementación del front-end se muestran en la Figura 6.3b. De esta forma, cada uno de los submódulos consta tres archivos de código: *Model*, *View* y *Controller*.

En las figuras 6.8 y 6.7 se muestran dos capturas de pantalla de la interfaz gráfica implementada, con el tema visual por defecto y en español. El manual de usuario, disponible en el apéndice A, contiene el resto de capturas de las páginas de la interfaz web y una explicación detallada de cada una.

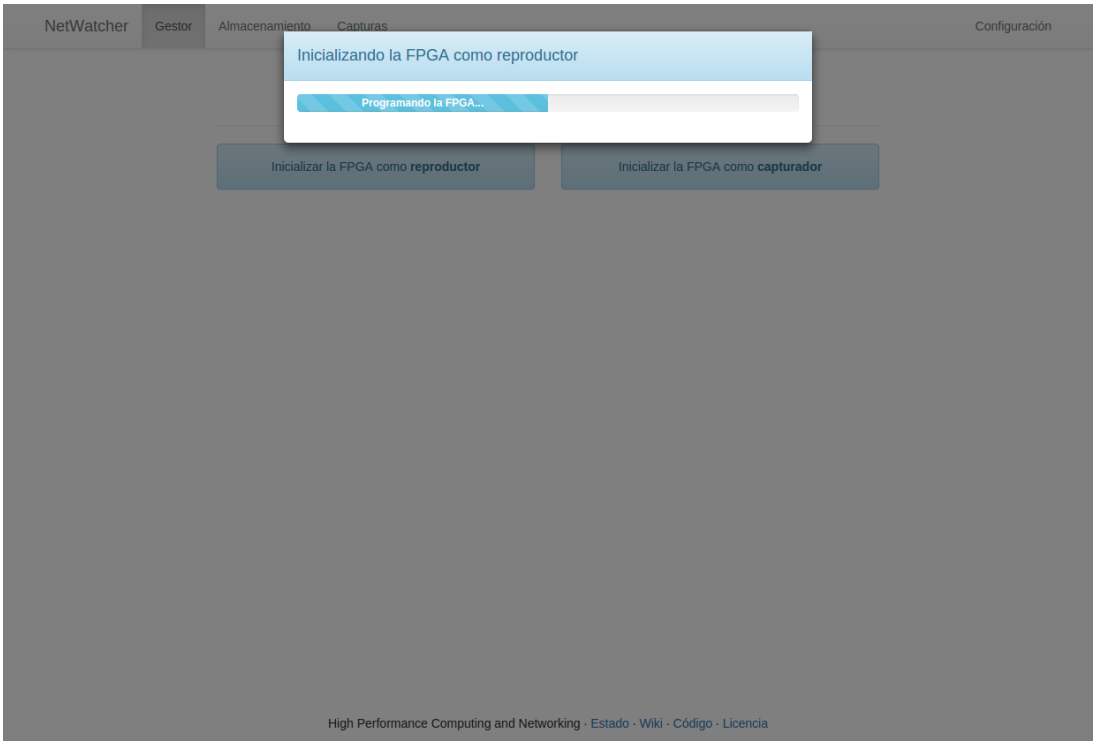


Figura 6.7: Página de gestión - selección de modo en progreso (reproductor).

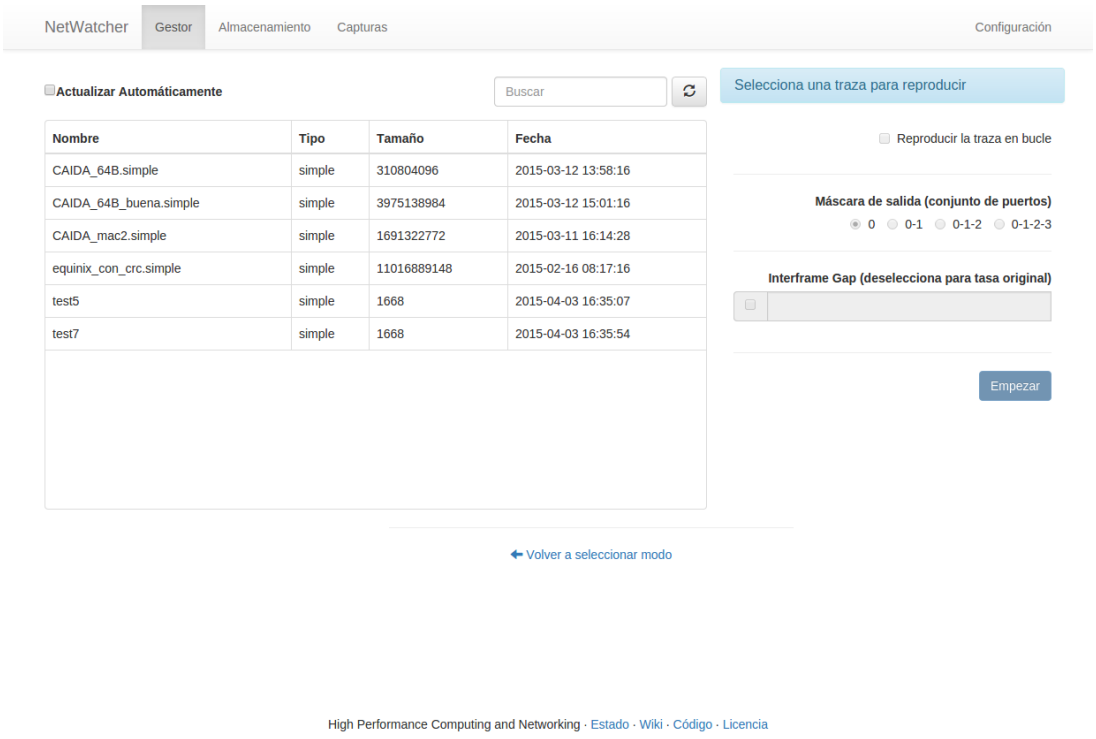


Figura 6.8: Página de gestión - formulario de reproducir traza.

6.3. Documentación

Se ha creado distinta documentación del proyecto según a quién esté dirigida. Por un lado, se ha escrito un manual de usuario (disponible en el apéndice A), que pretende ser una guía completa y suficiente para la instalación, configuración y uso de la aplicación. Adicionalmente, se han publicado en el repositorio de *GitHub* del proyecto (`github.com/JSidrach/NetWatcher`) una serie de páginas en formato *wiki* (ver Figura 6.9). Estas páginas, en inglés, recogen los aspectos más importantes de la aplicación, tanto para el usuario final como para desarrolladores.

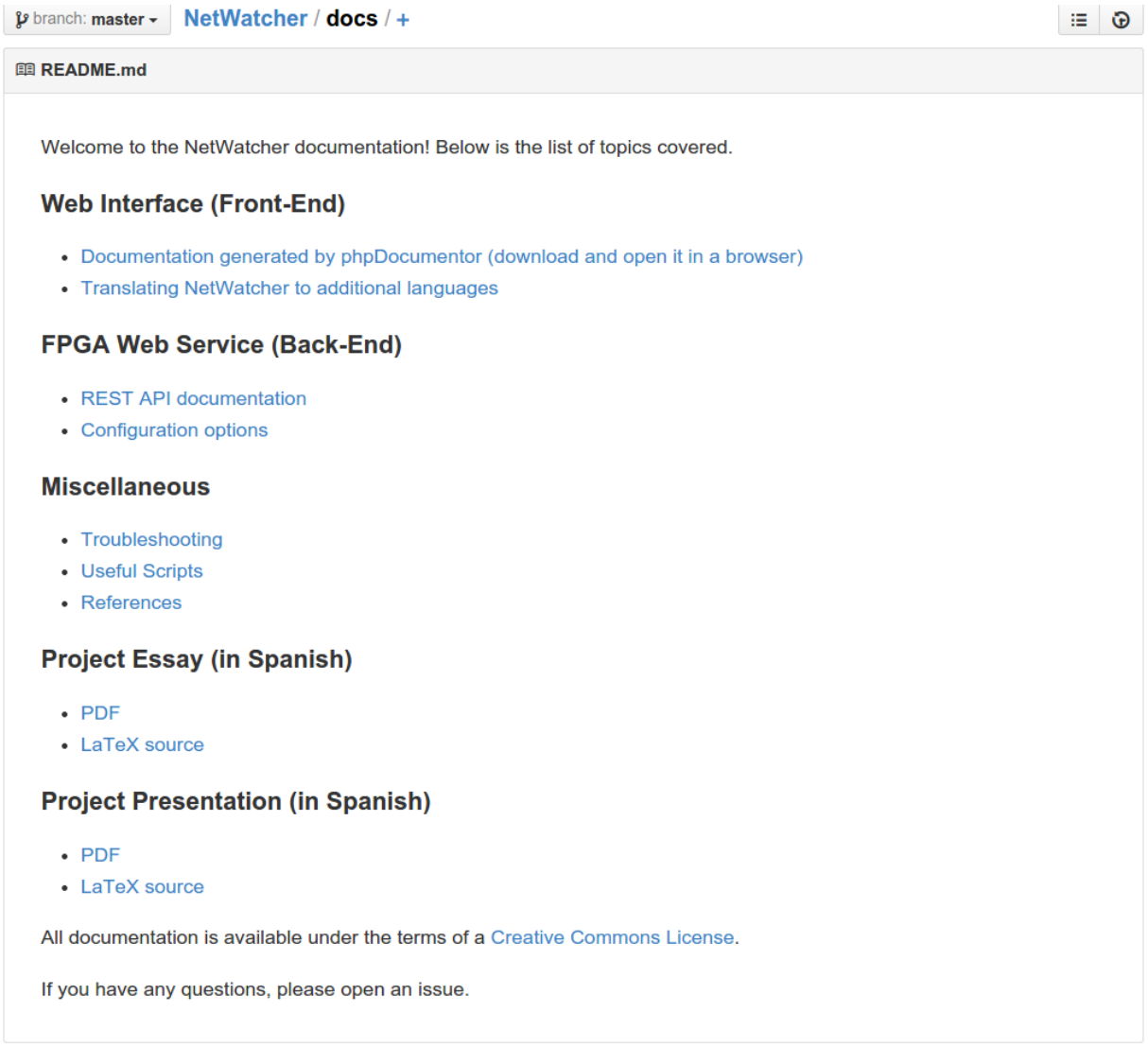


Figura 6.9: Captura de una de las páginas de la wiki del proyecto.

A nivel específico de desarrollador, se han utilizado dos herramientas para crear la documentación interna, accesible a través del navegador y en inglés. La documentación del front-end se ha generado con *phpDocumentor* (ver Figura 6.10), y la del back-end con *apiDoc*. Esta última se incluye también, traducida al español, en el apéndice C. Por

último, en el apéndice B se explican la arquitectura y funcionalidad del framework base para el front-end.

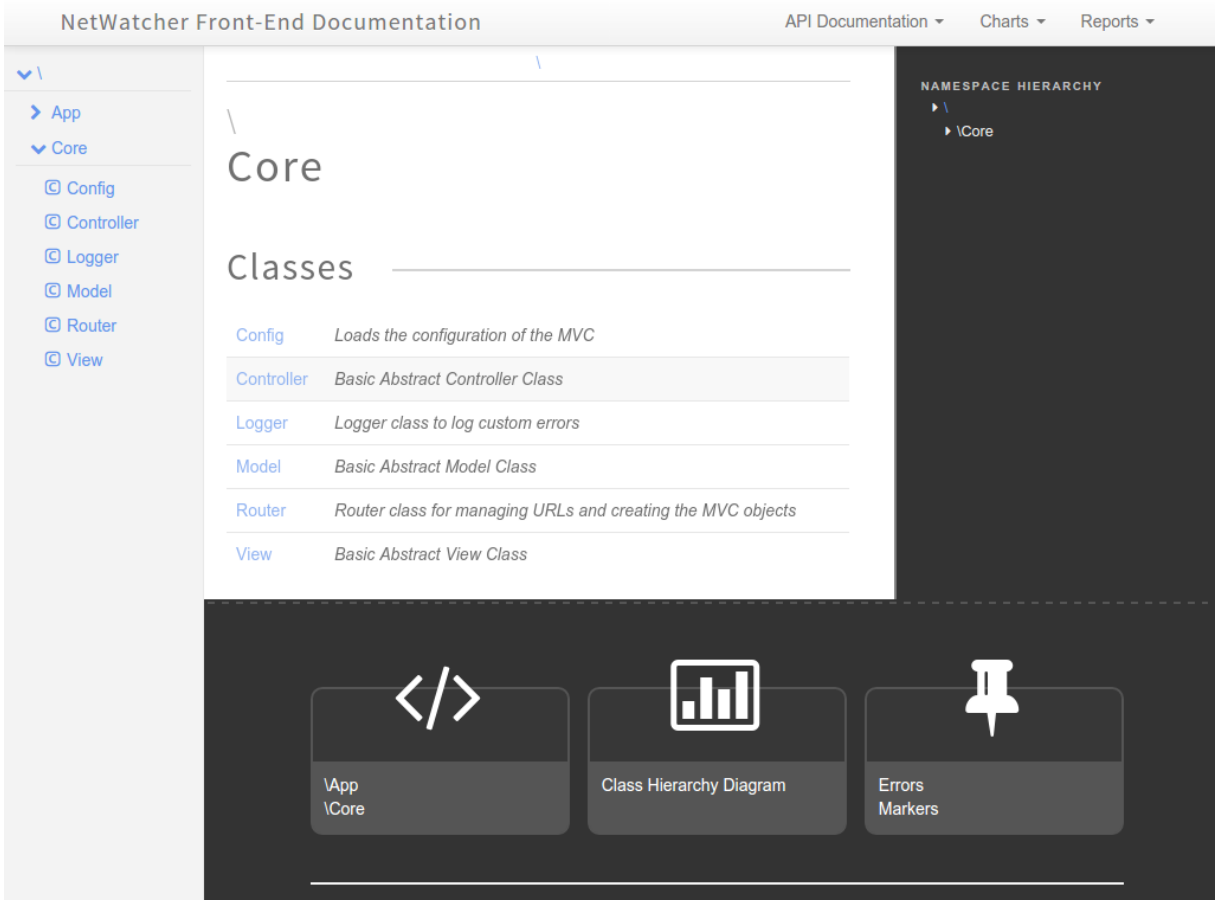


Figura 6.10: Documentación web del front-end.

7

Verificación y validación

En este capítulo se explica el proceso seguido para la verificación y validación de la aplicación. Este proceso tiene como objetivo asegurar la corrección del sistema, comprobar que satisface los requisitos y mejorar la implementación evaluada.

7.1. Verificación

La verificación del proyecto busca comprobar que la aplicación concuerda con su especificación, es decir, que la implementación es correcta. Dentro de este ámbito, se han planteado y realizado inspecciones del código, pruebas de caja blanca, pruebas de caja negra, pruebas de integración y pruebas sobre la interfaz de usuario.

Inspección del código

La inspección del código es una técnica que consiste en revisar el código fuente. Se ha aplicado en todos los archivos de código del proyecto, con el objetivo mejorar la estructura interna y el estilo del código fuente. Sin embargo, no se ha utilizado para descubrir errores, ya que para eso se ha recurrido a otro tipo de pruebas, como las de caja negra y las de caja blanca.

La estrategia seguida ha sido realizar una inspección en cada fichero, una vez ha pasado al menos una semana desde su codificación. Así, al no tenerla tan cercana, se ha facilitado el evaluar de manera más objetiva la implementación.

Pruebas de caja blanca

Las pruebas de caja blanca son las que se realizan teniendo en cuenta la estructura interna del programa. Tienen como objetivo verificar el correcto funcionamiento del código y detectar errores. Concretamente, se han llevado a cabo aquellas basadas en la comprobación de los posibles flujos de ejecución de una función, mediante su invocación con valores típicos, aleatorios y límite.

Debido al alto coste temporal de plantear y ejecutar este tipo de pruebas para todo el sistema, se ha decidido aplicarlas en un único submódulo, considerado crítico para la aplicación: la máquina de estados finitos que formaliza el estado de la FPGA. De esta forma se ha comprobado, mediante pruebas más exhaustivas que las de caja negra, que la parte esencial de la aplicación tenga el comportamiento esperado.

Pruebas de caja negra

Las pruebas de caja negra, en contraposición con las de caja blanca, son aquellas que se realizan sin tener en cuenta la estructura interna del programa. Para este tipo de pruebas, se ha seguido una estrategia ascendente, comprobando primero las funciones de más bajo nivel. Dada la arquitectura de la aplicación, se ha decidido separar las pruebas de caja negra entre el back-end y el front-end.

Dentro de los componentes back-end, se han realizado primero pruebas en los métodos básicos, comunes a todos los módulos (*common.js*). Posteriormente, se han comprobado las funciones auxiliares de cada módulo (*_utils.js*). Por último, se ha creado y ejecutado un conjunto de pruebas sobre la API del Servicio Web FPGA con *Postman* [27]. Esto permite que se puedan replicar fácilmente, ya que pueden ser exportadas a un único fichero de configuración para después importarlo cuando se deseen volver a ejecutar las pruebas.

Para el front-end, se ha elegido efectuar pruebas de caja negra sobre un único módulo, ya que se han considerado de más utilidad aplicar pruebas de contenido sobre la interfaz de usuario (ver apartado 7.1). De este modo, se ha verificado el funcionamiento del proxy haciendo uso de las mismas pruebas de *Postman* creadas para el front-end, comprobando que se obtenía un resultado idéntico.

Pruebas de integración

Las pruebas de integración tienen como objetivo verificar que los componentes de la aplicación encajan correctamente entre sí. Tras las pruebas de caja blanca y caja negra, se ha realizado este tipo de pruebas de forma ascendente, empezando con los componentes más básicos y terminando con la integración entre el back-end y el front-end.

Pruebas sobre la interfaz de usuario

Para la interfaz web se han realizado pruebas basadas en el contenido. En primer lugar, se ha verificado que la interfaz web se encuentra disponible e informa correctamente al usuario del estado del sistema, incluso cuando el back-end no está operativo. En segundo lugar, se ha comprobado que se producen los cambios esperados en la interfaz como respuesta a las acciones del usuario. Finalmente, se han evaluado todos los formularios en la interfaz rellenando los parámetros disponibles con valores válidos e inválidos, constatando que el usuario recibe información sobre qué parámetros son correctos y cuáles no.

7.2. Validación

La validación de la aplicación consiste en constatar que satisface el propósito para el que fue planteada. Se comprueba, por tanto, que cumpla los requisitos especificados en el capítulo 4, tanto funcionales como no funcionales.

En este contexto, se ha confirmado que existe una forma de satisfacer, mediante la interfaz web, cada uno de los requisitos funcionales. Respecto a los requisitos no funcionales, se ha validado uno a uno su cumplimiento, ya que estaban todos relacionados con la interfaz web. Adicionalmente, se ha evaluado la usabilidad de la aplicación comprobando que un grupo de usuarios pudiese utilizar toda la funcionalidad ofrecida por interfaz, con la ayuda opcional del manual.

Con este propósito se ha determinado, mediante un formulario de respuesta múltiple, con qué facilidad los usuarios pueden completar tareas desde la interfaz (ver Tabla 7.1). Han colaborado, para ello, personas pertenecientes al grupo de investigación en el que se enmarca este proyecto. Dado que para realizar pruebas se cuenta con una única sonda de red, cada test solo puede comenzar una vez ha terminado el anterior. Por este motivo, se ha decidido reducir el conjunto de tareas a las más significativas, disminuyendo el tiempo necesario. Destacar que no se especifica en ningún sitio la página concreta a la que debe acceder el usuario para completar cada acción requerida.

La dificultad con la que se completa cada tarea se clasifica en una de las siguientes categorías:

- **Fácil**: el usuario ha completado la tarea sin consultar el manual, en menos de un minuto.
- **Media**: el usuario ha completado la tarea sin consultar el manual, tardando más de un minuto para ello.
- **Difícil**: el usuario ha completado la tarea consultando el manual.
- **No lograda**: el usuario ha desistido y no ha conseguido completar la tarea.

	Dificultad			
Tarea	**Fácil**	**Media**	**Difícil**	**No lograda**
Configurar la interfaz para conectarse a una URL				
Programar la sonda en modo reproducción				
Reproducir una traza en bucle, por el puerto 0 y IFG 0				
Parar la reproducción de la traza				
Reprogramar la sonda en modo captura				
Ordenar capturar una traza de 200 GB por el puerto 3				
Parar la captura de la traza				
Convertir una traza simple a formato pcap				
Renombrar una traza				
Borrar una traza				
Comprobar el espacio de almacenamiento disponible				
Comprobar la velocidad de escritura del RAID				

Tabla 7.1: Conjunto de preguntas que conforman el test de validación.

7.3. Resultados

Las distintas pruebas de verificación han tenido las siguientes consecuencias directas:

- Se ha refactorizado parte de la implementación, mejorando su estructura interna, claridad y mantenibilidad, fruto de las inspecciones de código.
- Se han corregido errores menores gracias a las pruebas de caja blanca y de caja negra.
- Se han podido detectar y subsanar errores menores relacionados con la comunicación entre los componentes, como resultado de las pruebas de integración. Otra consecuencia ha sido el descubrimiento y la corrección del hecho de que funcionalidad no esencial implementada en el back-end no tuviese correspondencia en el front-end, por lo que nunca se llegaba a aprovechar.

Respecto a las pruebas de validación, el formulario ha permitido comprobar cómo de fácil es manejar la interfaz web. Los resultados han sido satisfactorios, ya que si bien no todas las tareas han sido clasificadas como "Fácil", ninguna de ellas no ha podido ser completada por el usuario. Por lo general, la gran mayoría de tareas ha tenido una dificultad "Fácil" o "Media", por lo que no han requerido la ayuda del manual de usuario. Se valida así el cumplimiento de uno de los objetivos del proyecto, simplificar la gestión de la sonda de red.

8

Mantenimiento

Se expone a continuación el plan de mantenimiento para la aplicación desarrollada. Dentro de las diferentes categorías de mantenimiento (perfectivo, adaptativo, preventivo y correctivo), solo entra dentro del alcance de este proyecto realizar un mantenimiento correctivo hasta máximo un mes después de entregar el producto final (al menos por parte del estudiante). Esta decisión ha sido fundamentada en el límite de tiempo que se recomienda para la elaboración del Trabajo de Fin de Grado. Se diagnosticarán y corregirán por tanto errores durante el periodo establecido, como respuesta a solicitudes de los usuarios (a través de la apertura de peticiones en el repositorio del código).

Se han identificado, no obstante, líneas de trabajo futuro y posibles mejoras (ver capítulo 10) que no entrarían dentro de este tipo de mantenimiento, pero cuyo planteamiento se ha considerado interesante. Por ello, se ha decidido liberar el código de la aplicación en *GitHub* (`https://github.com/JSidrach/NetWatcher`) bajo la licencia *MIT* [28], de forma que el proyecto pueda ser continuado por otras personas. Se ha seleccionado *GitHub* como plataforma para alojar el código porque facilita la colaboración entre desarrolladores y porque el propio proyecto ya estaba alojado allí de manera privada, no requiriéndose una migración. Se ha elegido la licencia *MIT* por ser una de las menos restrictivas dentro de las de código abierto [29], permitiendo que cualquiera pueda utilizar y modificar la aplicación siempre que se reconozca la autoría original. Se espera así que en un futuro otras personas puedan ampliar el trabajo realizado.

9 Conclusiones

Este Trabajo de Fin de Grado ha consistido en el desarrollo de una interfaz para la gestión de sondas de red de altas prestaciones. Para ello, se ha analizado el problema, estudiado el estado del arte, definido de forma rigurosa y completa el proyecto a realizar, diseñado la solución, implementado la misma, llevado a cabo una verificación y validación del sistema y, por último, se ha ejecutado un mantenimiento correctivo.

La aplicación propuesta facilita la utilización de la sonda de red mediante una interfaz gráfica basada en tecnologías web. Esta interfaz puede ser utilizada por cualquier usuario con conocimientos informáticos, sin ser necesario un conocimiento específico de la sonda en sí. Se cumple así el objetivo de simplificar el empleo de la sonda seleccionada que anteriormente solo se podía manejar por línea de comandos. También agrupa otros aspectos relevantes de la captura y reproducción de tráfico web, como la gestión del almacenamiento de las trazas, la velocidad de escritura en disco o la conversión entre formatos de traza. La implementación de la interfaz sigue un diseño *responsive*, lo que hace que se pueda utilizar desde dispositivos móviles sin empeorar la experiencia de usuario. Se ha conseguido, como producto final, una interfaz elegante que permite gestionar todos los aspectos de la sonda considerados relevantes, presentando el estado e información adicional del sistema de forma visual, mediante gráficos y estadísticas.

Sin embargo, este proyecto no ha consistido únicamente en el desarrollo de una interfaz. Se ha dividido el sistema en dos componentes, back-end y front-end, que se comunican entre sí y que pueden estar alojados en un servidor distinto cada uno. Así, se ha diseñado e implementado un Servicio Web REST en el back-end, que formaliza el estado y funcionalidad de la FPGA, añadiendo también control sobre otros aspectos relevantes mencionados anteriormente. Para el front-end, se ha creado un framework propio, sobre el que se ha implementado la interfaz web. Esta arquitectura de la aplicación cuenta con ciertas ventajas respecto a no separar físicamente el front-end y el back-end. Al monitorizar

el estado de la FPGA, la interfaz estará disponible aunque el servidor que aloja la sonda no se encuentre operativo. Por otra parte, al trasladar la interfaz a un servidor distinto, se libera parte de los recursos del servidor conectado a la sonda de red, afectando menos a su rendimiento. Este proyecto, por tanto, proporciona una base sólida sobre la que se puede incluir soporte a otras sondas de red sin partir desde cero, y siendo solo necesario modificar una parte bien diferenciada del back-end.

Para la implementación de la aplicación, se han utilizado diversos lenguajes de programación y herramientas, lo que constituye, sin duda, una valiosa experiencia para el futuro profesional del estudiante. En el back-end se ha empleado *node.js*, una librería para construir aplicaciones de red en *JavaScript* (tradicionalmente utilizado en el cliente, no en el servidor). La codificación sobre esta librería se basa en la programación asíncrona, paradigma que ha sido enriquecedor aprender. El front-end se ha implementado con varios lenguajes de programación: *PHP*, *JavaScript*, *HTML* y *CSS*. Se han empleado en este componente las librerías *Bootstrap* y *jQuery*, comunes en el desarrollo de páginas web.

Durante el desarrollo de este proyecto, se han podido poner en práctica conocimientos adquiridos en diversas asignaturas del Grado en Ingeniería Informática. Por ejemplo, se ha seguido un proceso completo de desarrollo *software*, aprendido en Ingeniería del Software. Asignaturas como Arquitectura de Computadores y Redes de Comunicaciones I y II han sido claves, al ser el trabajo base del que parte este proyecto una sonda de red (que a pesar de no haberse modificado, sí ha sido necesario entender su funcionamiento). Conceptos como Servicio Web, API REST y algunos lenguajes de programación web fueron estudiados en Sistemas Informáticos I y II, lo que ha agilizado su adopción.

Por último, destacar que se ha liberado el código fuente del proyecto en la plataforma *GitHub* (`https://github.com/JSidrach/NetWatcher`). Esto no ha sido un simple gesto, ya que se ha invertido tiempo adicional en documentar todo el código de la aplicación en inglés. Además, se han creado páginas *wiki* dentro de la propia plataforma explicando el funcionamiento interno de la aplicación, junto con manuales de instalación, configuración, etc. Se espera, con este esfuerzo, fomentar la participación de otros desarrolladores de la comunidad en la ampliación y mejora del sistema implementado (se dan algunas ideas en el capítulo 10). Liberar el proyecto ha consistido también en mejorar y poner a disposición de otros alumnos la plantilla en *LaTeX* creada para esta memoria, que se encuentra disponible en otro repositorio público de *GitHub* (`https://github.com/JSidrach/tfg-plantilla`). A fecha de hoy, esta plantilla está siendo utilizada en varios trabajos más.

10

Líneas de trabajo futuro

En el contexto de este Trabajo de Fin de Grado, se ha desarrollado una interfaz web para el manejo de sondas red de altas prestaciones. Gracias al trabajo realizado, se han identificado áreas de interés que podrían ser consideradas con el objetivo de mejorar y ampliar la aplicación en el futuro, y que no han podido ser abordadas en el mismo por la limitación del tiempo disponible. Se describen a continuación algunas de estas posibles.

Ampliación del Servicio Web a otras sondas de red

La aplicación implementada gestiona un dispositivo concreto de captura y reproducción de tráfico de red. Aunque algunos componentes son específicos para la FPGA utilizada, también se han desarrollado componentes más genéricos como los de gestión de capturas y del almacenamiento, o el propio front-end. Es por ello que una posible área de ampliación del proyecto sería incluir en el Servicio Web (back-end) soporte para otras sondas de red, distinguiendo y documentando qué métodos son comunes e independientes de la sonda seleccionada y cuáles son específicos para cada tipo de sonda.

Soporte de subtipos de trazas pcap adicionales

El sistema de gestión de trazas actual soporta los formatos simple y pcap. Las trazas en formato pcap tienen sin embargo subtipos, cada uno con características distintas que en el sistema actual se descartan. En línea con la estandarización del Servicio Web, poder distinguir entre los distintos subtipos de trazas pcap facilitaría obtener información adicional propia de cada subformato, permitiendo además clasificar y convertir entre cada uno de los subtipos.

Registro de estadísticas adicionales

El sistema actual consta de un módulo que proporciona estadísticas en tiempo real sobre el estado de la FPGA y de los distintos componentes que intervienen en el proceso de captura y reproducción. Estos datos no se almacenan de forma persistente una vez obtenidos. Una opción sería guardar en una base de datos estas estadísticas y parámetros de utilización de la FPGA. Esto permitiría un análisis posterior de estas estadísticas almacenadas para sacar conclusiones sobre distintos parámetros como el rendimiento o las operaciones más frecuentes.

Internacionalización en otros idiomas

El trabajo base para dar soporte a diferentes idiomas en la interfaz gráfica ya ha sido realizado, y actualmente la aplicación está disponible en español e inglés. Por tanto, es posible añadir idiomas adicionales a la interfaz traduciendo las distintas cadenas de texto a otros idiomas, sin ser necesario esfuerzo adicional a nivel de diseño e implementación.

Módulo de autenticación

Dado que la interfaz web está pensada para ser utilizada en redes internas, sin acceso desde el exterior, no se ha planteado implementar un módulo de autenticación que impida a usuarios no autorizados el acceso a la aplicación. Desarrollar este módulo de autenticación haría posible instalar el servidor en una dirección pública, sin ceder por ello el control del sistema a una persona ajena. Esto permitiría que un usuario autorizado pudiera utilizar la interfaz desde cualquier punto con conexión a internet.

Bibliografía

[1] J.F. Zazo y col. «TNT10G: a high-accuracy 10 GbE traffic player and recorder for multi-Terabyte traces». En: *ReConFigurable Computing and FPGAs (ReConFig), International Conference*. 2014.

[2] G. Antichi y col. «OSNT: open source network tester». En: *Network, IEEE* 28.5 (2014), págs. 6-12. ISSN: 0890-8044. DOI: 10.1109/MNET.2014.6915433.

[3] Muhammad Shahbaz y col. «Architecture for an Open Source Network Tester». En: *Proceedings of the Ninth ACM/IEEE Symposium on Architectures for Networking and Communications Systems*. ANCS '13. San Jose, California, USA: IEEE Press, 2013, págs. 123-124. ISBN: 978-1-4799-1640-5.

[4] J.W. Lockwood y col. «NetFPGA–An Open Platform for Gigabit-Rate Network Switching and Routing». En: *Microelectronic Systems Education, 2007. MSE '07. IEEE International Conference on*. 2007, págs. 160-161. DOI: 10.1109/MSE.2007.69.

[20] Ethan Marcotte. *Responsive Web Design*. [Accedido: 22/05/2015]. URL: http://alistapart.com/article/responsive-web-design.

[33] Trygve Reenskaug. *The Model-View-Controller (MVC). Its Past and Present*. [Accedido: 22/05/2015]. URL: http://heim.ifi.uio.no/~trygver/2003/javazone-jaoo/MVC_pattern.pdf.

[34] A. Barth. *The Web Origin Concept*. RFC 6454. [Accedido: 22/05/2015]. RFC Editor, 2011. URL: https://tools.ietf.org/html/rfc6454.

[43] R. Fielding y J. Reschke. *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content (Section 4 - Request Methods)*. RFC 7231. [Accedido: 22/05/2015]. RFC Editor, 2014. URL: https://tools.ietf.org/html/rfc7231#section-4.

[44] R. Fielding y J. Reschke. *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content (Section 6 - Response Status Codes)*. RFC 7231. [Accedido: 22/05/2015]. RFC Editor, 2014. URL: https://tools.ietf.org/html/rfc7231#section-6.

Referencias

Las siguientes direcciones web enlazan a información extendida sobre las herramientas, lenguajes de programación y librerías mencionadas en la memoria.

[5] *OSNT*. [Accedido: 22/05/2015]. URL: http://osnt.org.

[6] *tcpdump y libpcap*. [Accedido: 22/05/2015]. URL: http://www.tcpdump.org.

[7] *Wireshark*. [Accedido: 22/05/2015]. URL: https://www.wireshark.org.

[8] *Detect-Pro*. [Accedido: 22/05/2015]. URL: http://www.naudit.es/index.php?s=3&p=5&l=0.

[9] *GitHub*. [Accedido: 22/05/2015]. URL: https://github.com.

[10] *Git*. [Accedido: 22/05/2015]. URL: http://git-scm.com.

[11] *Cacoo*. [Accedido: 22/05/2015]. URL: https://cacoo.com.

[12] *apiDoc*. [Accedido: 22/05/2015]. URL: http://apidocjs.com.

[13] *phpDocumentor*. [Accedido: 22/05/2015]. URL: http://www.phpdoc.org.

[14] *Node.js*. [Accedido: 22/05/2015]. URL: https://nodejs.org.

[15] *JavaScript*. [Accedido: 22/05/2015]. URL: https://developer.mozilla.org/docs/Web/JavaScript.

[16] *Express*. [Accedido: 22/05/2015]. URL: http://expressjs.com.

[17] *Async*. [Accedido: 22/05/2015]. URL: https://github.com/caolan/async.

[18] *nodemon*. [Accedido: 22/05/2015]. URL: http://nodemon.io.

[19] *Bootstrap*. [Accedido: 22/05/2015]. URL: http://getbootstrap.com.

[21] *Bootstrap table*. [Accedido: 22/05/2015]. URL: http://bootstrap-table.wenzhixin.net.cn.

[22] *Bootswatch*. [Accedido: 22/05/2015]. URL: https://bootswatch.com.

[23] *Bootstrap Notify*. [Accedido: 22/05/2015]. URL: http://bootstrap-notify.remabledesigns.com.

[24] *jQuery*. [Accedido: 22/05/2015]. URL: https://jquery.com.

[25] *Chart.js*. [Accedido: 22/05/2015]. URL: http://www.chartjs.org.

[26] *Animate.css*. [Accedido: 22/05/2015]. URL: http://daneden.github.io/animate.css.

[27] *Postman*. [Accedido: 22/05/2015]. URL: https://www.getpostman.com.

[28] *The MIT License (MIT)*. [Accedido: 22/05/2015]. URL: http://opensource.org/licenses/MIT.

[29] *Licenses comparison*. [Accedido: 22/05/2015]. URL: http://choosealicense.com/licenses.

[30] *Debian distribution*. [Accedido: 22/05/2015]. URL: http://www.debian.org.

[31] *The Apache HTTP Server Project*. [Accedido: 22/05/2015]. URL: http://httpd.apache.org.

[32] *Apache module mod_rewrite*. [Accedido: 22/05/2015]. URL: http://httpd.apache.org/docs/current/mod/mod_rewrite.html.

[35] *GNU gettext*. [Accedido: 22/05/2015]. URL: http://www.gnu.org/software/gettext.

[36] *Poedit*. [Accedido: 22/05/2015]. URL: http://poedit.net.

[37] *Composer*. [Accedido: 22/05/2015]. URL: https://getcomposer.org.

[38] *BowerPHP*. [Accedido: 22/05/2015]. URL: http://bowerphp.org.

[39] *Packagist*. [Accedido: 22/05/2015]. URL: https://packagist.org.

[40] *PSR-4*. [Accedido: 22/05/2015]. URL: http://www.php-fig.org/psr/psr-4.

[41] *Bower*. [Accedido: 22/05/2015]. URL: http://bower.io.

[42] *Bower packages*. [Accedido: 22/05/2015]. URL: http://bower.io/search.

[45] *JSON specification*. [Accedido: 22/05/2015]. URL: http://json.org.

Apéndices

A

Manual de usuario

Este manual pretende ser una guía para la instalación, configuración y uso de la interfaz web para la gestión de sondas de red de altas prestaciones.

A.1. Instalación del Servicio Web FPGA

Requisitos

El servidor que aloje el Servicio Web FPGA debe cumplir los siguientes requisitos:

- La FPGA para capturar/reproducir tráfico debe estar conectada.
- El sistema operativo debe estar basado en una distribución *Debian* [30] y tener una arquitectura de 64 bits.
- La opción por defecto en el gestor de arranque debe ser iniciar con la opción *HugePages* activa.
- El usuario *root* debe existir.

Instalación

Para instalar el Servicio Web FPGA, es necesario que se cumplan todos los requisitos y que se sigan las instrucciones descritas a continuación:

1. Descargar el código fuente del repositorio del proyecto (github.com/JSidrach/NetWatcher). La instalación se realiza de forma remota, así que no es necesario descargárselo en el propio servidor, aunque sí en un entorno con terminal.
2. Descomprimir el archivo *.zip*.
3. Editar el archivo `./fpga-api/scripts/update_server.sh`, estableciendo los parámetros `SERVER_IP` y `SERVER_PATH` como la dirección del servidor remoto que alojará el Servicio Web FPGA y la ruta donde guardar el código, respectivamente.
4. Situarse en la carpeta `./fpga-api/`.
5. Desplegar el servidor ejecutando el siguiente comando:

   ```
   ./scripts/update_server.sh
   ```
6. Iniciar sesión en el servidor remoto.
7. Iniciar el Servicio Web ejecutando el siguiente comando:

   ```
   sudo service fpga-api start
   ```
8. Comprobar que el servidor está activo con el siguiente comando:

   ```
   sudo service fpga-api status
   ```

A.2. Configuración del Servicio Web FPGA

Para configurar el Servicio Web FPGA, inicie sesión en el servidor en el que se instaló este servicio. Los distintos parámetros de configuración vienen recogidos en el archivo `config.js`, dentro de la carpeta raíz del servicio (el contenido de la variable `SERVER_PATH`, establecida en la instalación). Edite las distintas variables de este archivo (explicadas en la Tabla A.1) para configurar el servicio. No es necesario reiniciar el servicio para que los cambios en el archivo `config.js` se reflejen en el servidor.

A.3. Instalación de la interfaz web

Requisitos

El servidor que aloje la interfaz web debe cumplir los siguientes requisitos:

- *Apache httpd* [31] debe estar instalado.
- La dirección del Servicio Web FPGA debe ser accesible desde este servidor.

Variable	Tipo	Descripción
BASE_PREFIX	Cadena de texto	Prefijo base de la API
PORT	Número entero	Puerto del servicio
MAX_DELAY	Número entero	Retraso máximo entre el *timestamp* de las peticiones y el *timestamp* del servidor. Si es menor o igual que 0, no se descartará ninguna petición basándose en el *timestamp*
IMPACT_BIN	Cadena de texto	Ruta al ejecutable `impact` de *Xilinx*
CAPTURES_DIR	Cadena de texto	Directorio donde se guardarán las trazas (debe acabar en /)
RAID	Booleano	Bandera que indica si el RAID está activo o o no. Establezca esta variable como *true* solo si `CAPTURES_DIR` está sobre un RAID y las variables `RAID_DEV` y `RAID_DISKS` están asignadas.
RAID_DEV	Cadena de texto	Ruta al RAID
RAID_DISKS	Array de cadenas	Discos físicos del RAID (por ejemplo: `/dev/sdc`, `/dev/sdd`, etc.)

Tabla A.1: Variables de configuración del Servicio Web FPGA

Instalación

Para instalar la interfaz web, comprobar que se cumplan todos los requisitos y seguir las instrucciones que se describen a continuación:

1. Descargar el código fuente del repositorio del proyecto (github.com/JSidrach/NetWatcher).
2. Descomprimir el *.zip* y mover la carpeta base *NetWatcher* al directorio público de *Apache* (normalmente `/var/www/html/`).
3. Situarse en la carpeta base *NetWatcher*.
4. Instalar los paquetes y librerías necesarios ejecutando el siguiente comando:

   ```
   sudo ./scripts/build.sh -install
   ```

A.4. Configuración de la interfaz web

Se puede configurar la interfaz web accediendo en el navegador a la página de configuración, `IP_SERVIDOR_APACHE/NetWatcher/settings`. En esta pantalla (Figura A.1) se puede configurar el idioma, el aspecto visual (tema) y la dirección del Servicio Web FPGA. Para que los cambios se reflejen en la interfaz web es necesario guardarlos. En el resto del manual se presupone que el idioma seleccionado es español.

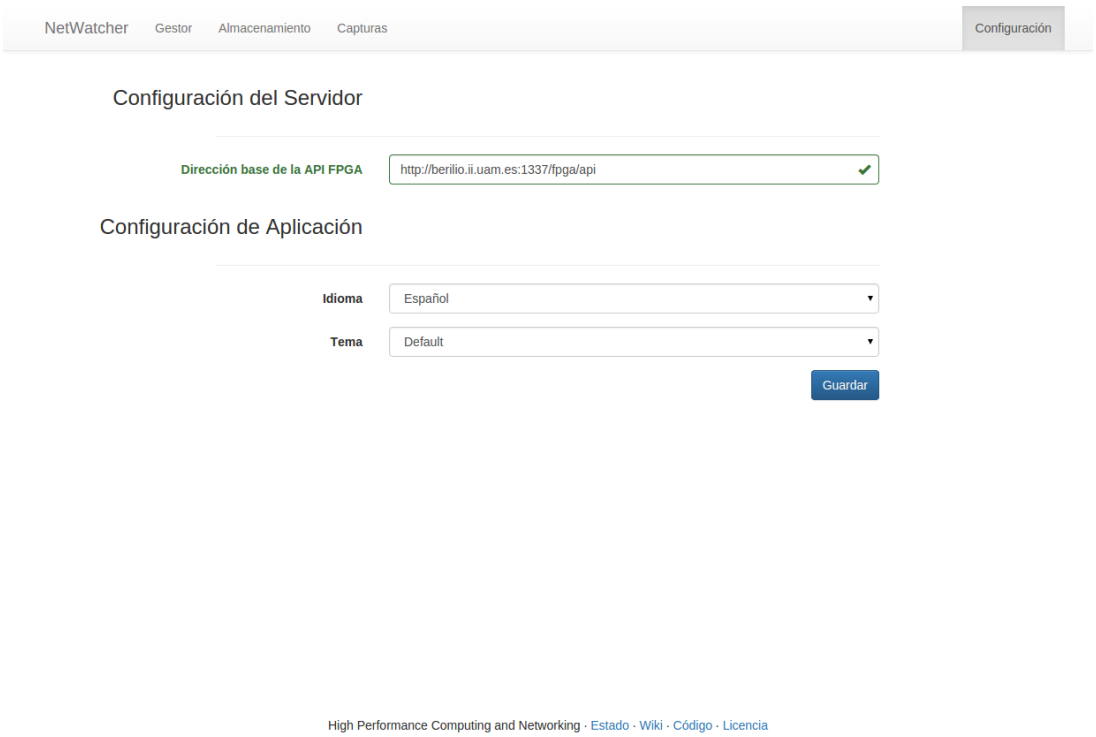


Figura A.1: Página de configuración de la interfaz web.

A.5. Uso de la aplicación

Una vez instalados y configurados tanto el Servicio Web FPGA como la interfaz web, ya se puede utilizar la interfaz. Esta interfaz se puede usar desde cualquier navegador, y tanto en ordenador como en móvil. Todas las pantallas tienen las mismas barras de navegación.

Desde la barra de navegación superior se puede acceder a las siguientes pantallas:

- **Gestor**: administración de la FPGA.
- **Almacenamiento**: estadísticas de almacenamiento y del RAID, si está activo.
- **Capturas**: gestión de las trazas almacenadas.
- **Configuración**: explicada en la sección A.4.

La barra de navegación inferior contiene los siguientes elementos:

- **Estado**: enlace a la pantalla con estado actual del sistema.
- **Wiki**: enlace a la documentación del proyecto.
- **Código**: enlace al repositorio de código del proyecto.

- **Licencia**: despliega la licencia del proyecto.

Adicionalmente, se puede acceder a la documentación interna autogenerada del proyecto (en inglés) mediante las siguientes rutas relativas a la dirección base de la interfaz web:

- **Documentación del Servicio Web FPGA**: `/docs-back-end/`.
- **Documentación de la interfaz web**: `/docs-front-end/`.

En las siguientes subsecciones se explica cómo utilizar las principales pantallas interactivas de la interfaz web.

A.5.1. Gestor

En esta pantalla se controla el estado de la FPGA. El contenido de esta pantalla, y por tanto las acciones disponibles, cambian según el estado actual de la FPGA.

Si la FPGA no ha sido inicializada, la pantalla de gestión permitirá seleccionar un modo en el que inicializarla:

- **Reproductor**: permite reproducir trazas en formato simple.
- **Capturador**: permite capturar tráfico web, almacenándolo en una traza en formato simple.

Para elegir un modo basta con pulsar uno de los dos botones de la interfaz (ver Figura A.2). Esta elección de modo no es definitiva, ya que se puede cambiar de modo en cualquier momento siempre que la FPGA no esté capturando o reproduciendo tráfico.

Una vez seleccionado un modo, un cuadro de diálogo muestra el progreso de la inicialización: programando la FPGA, reiniciando el servidor y montando la FPGA (ver Figura A.3).

Si la FPGA ha sido inicializada en modo capturador, la pantalla de gestión mostrará un formulario en el que se configurarán los parámetros de una captura (ver Figura A.4). Este formulario contiene los siguientes campos, todos ellos obligatorios:

- **Nombre de la nueva traza**: nombre que tendrá la traza en la que se almacenará el tráfico capturado, en formato simple.
- **Bytes a capturar**: número total de bytes que se capturarán, y su unidad (Bytes, KB, MB, GB).
- **Puerto a capturar**: puerto del que se capturará el tráfico entrante (0, 1, 2, 3).

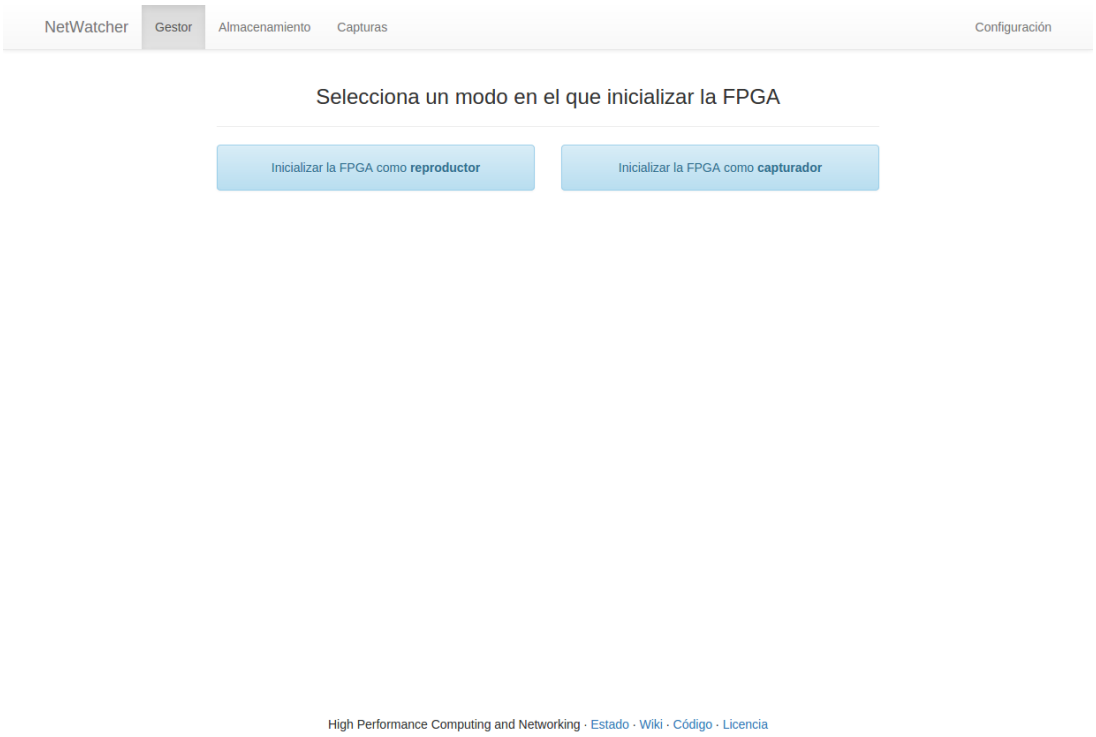


Figura A.2: Página de gestión - selección de modo.

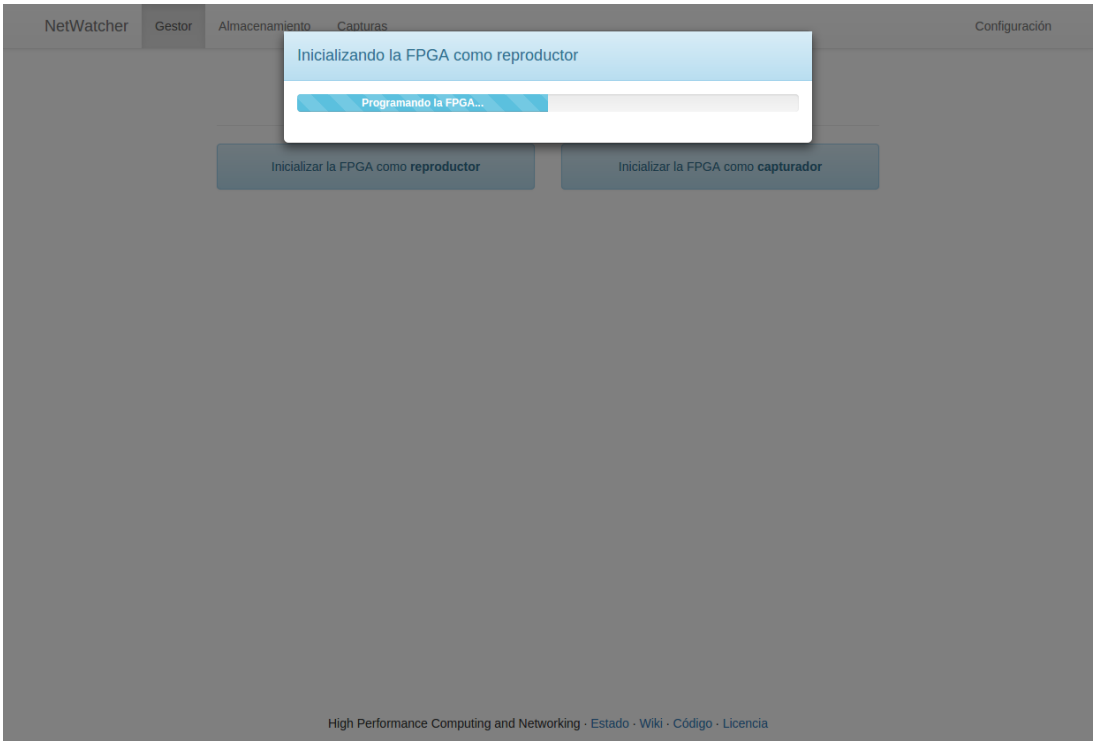


Figura A.3: Página de gestión - selección de modo en progreso.

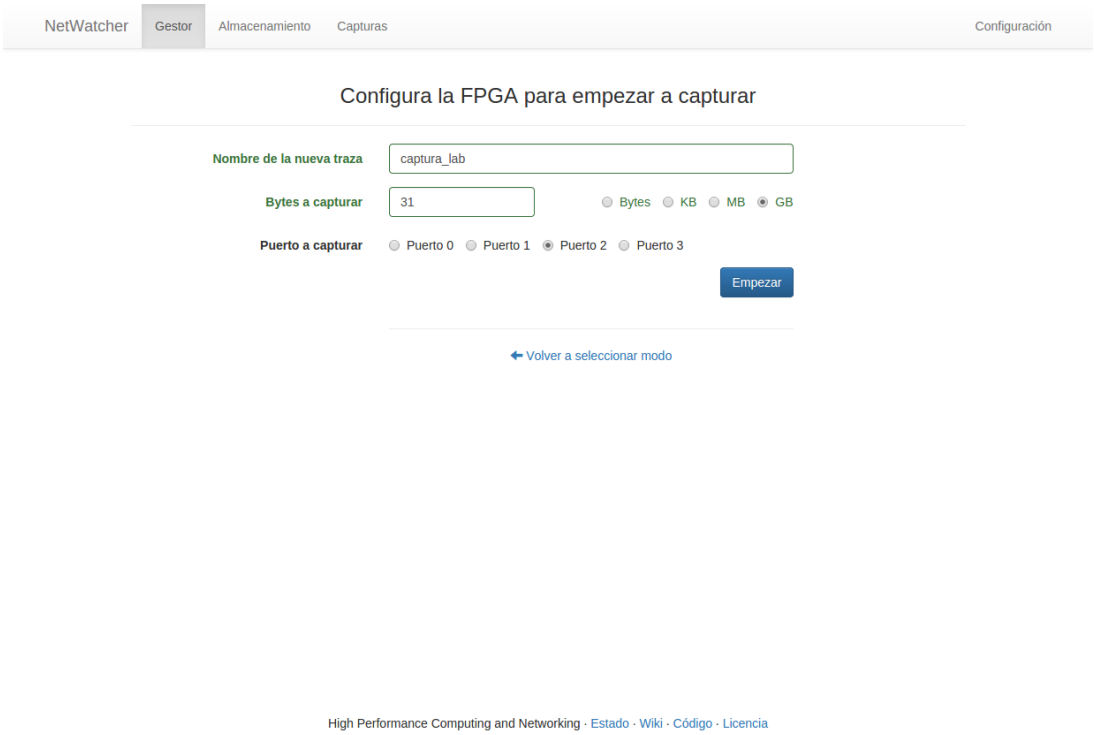


Figura A.4: Página de gestión - capturar tráfico.

Los dos primeros campos se iluminarán en verde cuando sean introducidos correctamente y en rojo cuando sean incorrectos. Cuando todos los campos sean válidos se activará el botón de *Empezar*, y si se pulsa la FPGA comenzará a capturar tráfico con los parámetros indicados.

También es posible, en vez de capturar tráfico, volver a seleccionar modo pulsando el correspondiente enlace debajo del formulario.

Si la FPGA está capturando tráfico, la pantalla de gestión mostrará el progreso de la captura en curso (ver Figura A.5). Se podrán visualizar las siguientes estadísticas de la captura en curso:

- **Nombre de la traza**: nombre de la traza en la que se está almacenando el tráfico capturado, en formato simple.
- **Puerto**: puerto del que se está capturando el tráfico entrante.
- **Tiempo transcurrido**: contador del tiempo que ha transcurrido desde que se inició la captura.
- **Bytes Capturados**: número de bytes que se han capturado ya.
- **Bytes Totales**: número total de bytes a capturar.
- **Ratio Medio**: velocidad media a la que se está capturando (estimación a partir de los bytes capturados y el tiempo transcurrido).

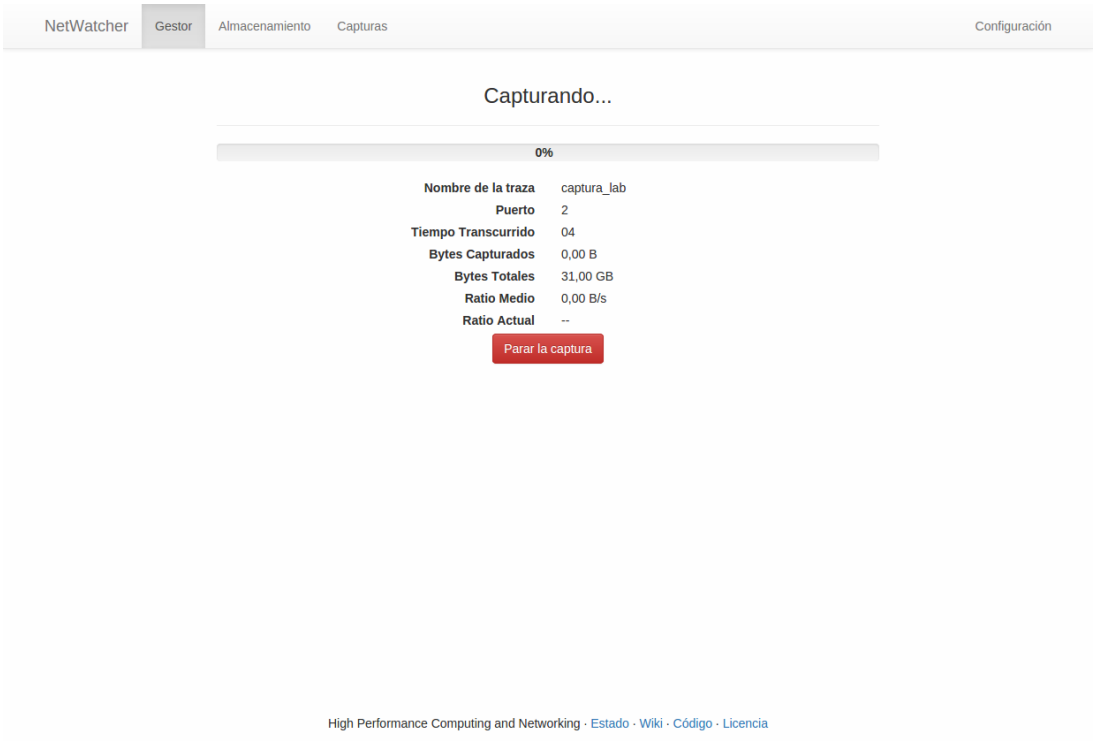


Figura A.5: Página de gestión - capturando tráfico.

- **Ratio Actual**: velocidad a la que se ha capturado el tráfico desde la última actualización.

Se puede detener la captura en curso pulsando el botón de *Parar la captura*, y se borrará lo almacenado hasta el momento en la traza.

Si la FPGA ha sido inicializada en modo reproductor, la pantalla de gestión mostrará una tabla y un formulario (ver Figura A.6). La tabla contiene una fila por cada traza disponible, con su nombre, tipo (simple), tamaño y fecha. Además, una barra superior asociada a esta tabla permite controlar el contenido de la misma mediante las siguientes acciones (de izquierda a derecha): activar la actualización automática de las trazas disponibles, buscar una traza por su nombre y actualizar manualmente las trazas disponibles. Pulsando sobre una fila de la tabla se seleccionará la traza correspondiente para su reproducción. Por otra parte, el formulario contiene los siguientes campos:

- **Reproducir la traza en bucle**: si se habilita, la traza se reproducirá en un bucle infinito.
- **Máscara de salida**: conjunto de puertos a los que se reproducirá la traza (0, 0-1, 0-1-2, 0-1-2-3).
- **Interframe Gap**: pausa temporal entre paquetes (si se deshabilita, tasa original con la que se capturó la traza).

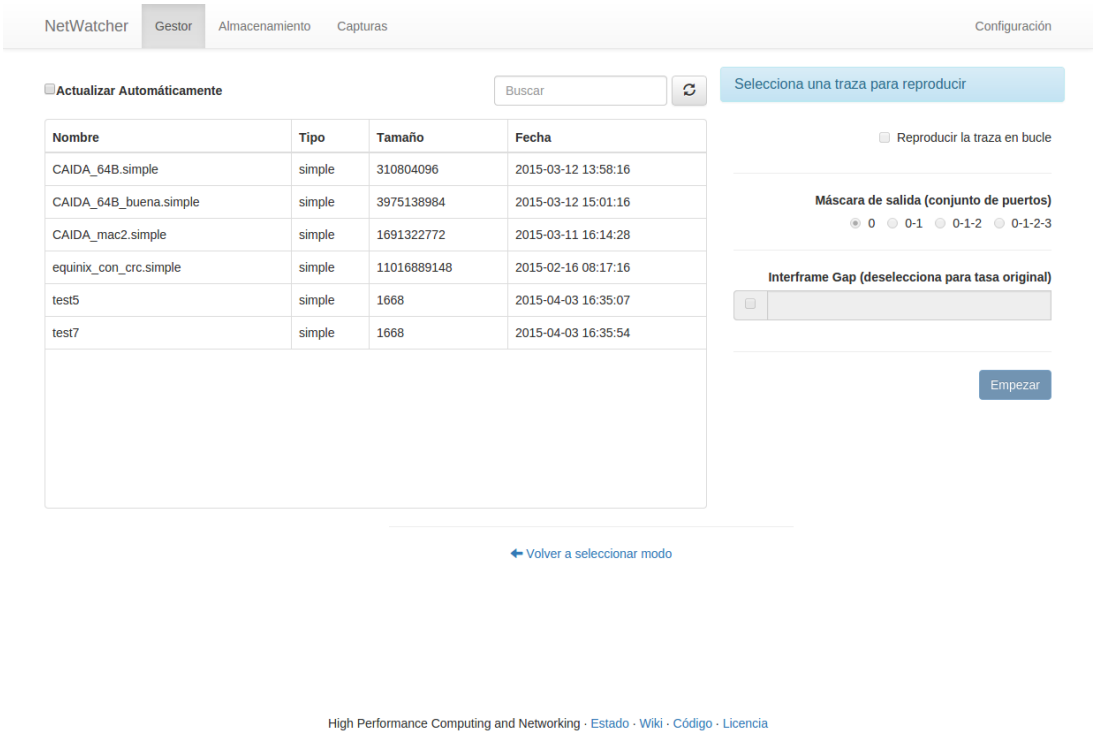


Figura A.6: Página de gestión - reproducir traza.

Cuando se haya seleccionado una traza de la tabla y todos los campos del formulario sean válidos se activará el botón de *Empezar* y, si se pulsa, la FPGA comenzará a reproducir la traza seleccionada con los parámetros indicados.

También es posible, en vez de reproducir tráfico, volver a seleccionar modo pulsando el correspondiente enlace debajo del formulario.

Si la FPGA está reproduciendo una traza, la pantalla de gestión mostrará el progreso de la reproducción en curso (ver Figura A.7). Se podrán visualizar las siguientes estadísticas de la reproducción en curso:

- **Nombre de la traza**: nombre de la traza que se está reproduciendo.
- **Tamaño**: número de bytes que ocupa la traza que se está reproduciendo.
- **Fecha**: fecha en que se creó la traza que se está reproduciendo.
- **Tiempo Transcurrido**: contador del tiempo que ha transcurrido desde que se inició la captura.
- **Paquetes Enviados**: número de paquetes que se han enviado en la reproducción actual.
- **Reproducción en Bucle**: indica si se está reproduciendo la traza en un bucle infinito o no.
- **Interframe Gap**: valor del IFG en la reproducción actual.

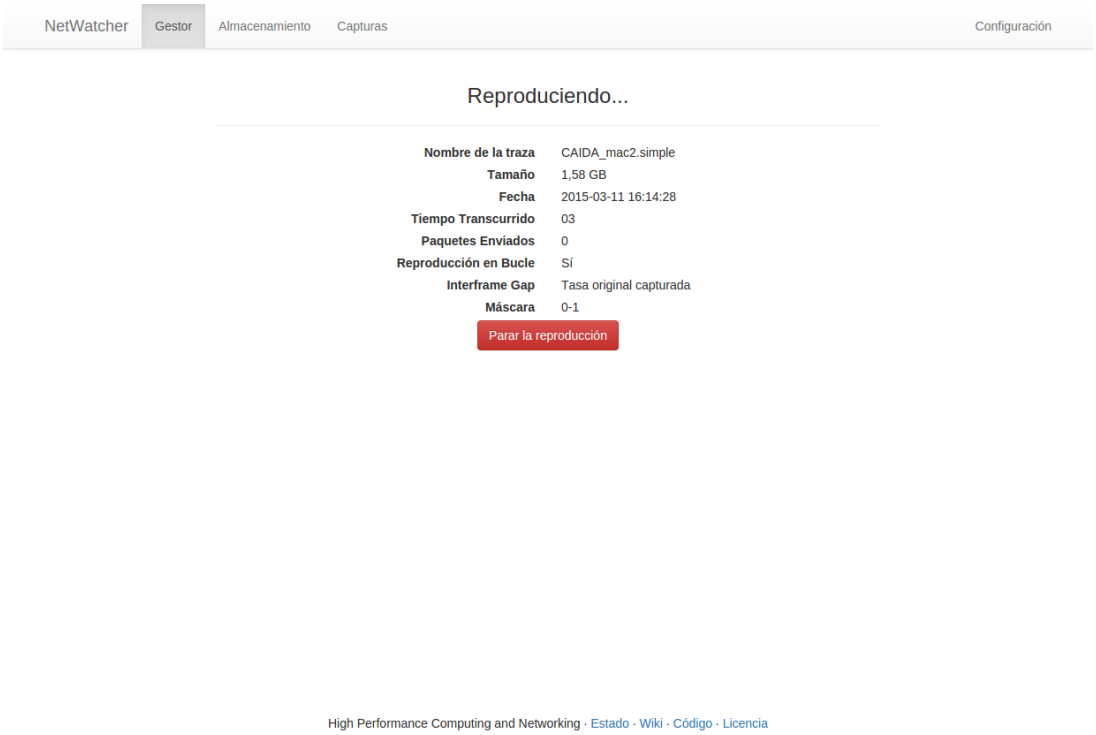


Figura A.7: Página de gestión - reproduciendo traza.

- **Máscara**: conjunto de puertos en los que se está reproduciendo la traza seleccionada (0, 0-1, 0-1-2, 0-1-2-3).

Se puede detener la reproducción en curso pulsando el botón *Parar la reproducción*.

A.5.2. Almacenamiento

En esta pantalla se pueden visualizar distintas estadísticas de almacenamiento del sistema. Está compuesta por dos paneles:

- **Estadísticas de espacio (Figura A.8)**: este panel muestra estadísticas del espacio total, ocupado y disponible, resumido además en un gráfico circular (en rojo la proporción de disco ocupado y en turquesa la proporción de disco disponible).
- **Estadísticas del RAID (Figura A.9)**: solo se mostrarán estas estadísticas si el sistema de almacenamiento está configurado como un RAID (ver sección A.2). En este panel, un gráfico de barras muestra la velocidad de escritura de cada disco del RAID. Debajo de este gráfico se indica la velocidad global de escritura del RAID. El color de esta cifra depende de la velocidad de escritura: verde (velocidad superior a la recomendada), amarillo (velocidad suficiente) o rojo (velocidad por debajo del mínimo aceptable). Si la velocidad es insuficiente se mostrará un cuadro de diálogo adicional para formatear y recrear el RAID pulsando el botón *Formatear el RAID*

(cuidado: formatear el RAID borrará todos los datos del mismo). Este diálogo se puede ocultar pulsando el botón *Descartar*.

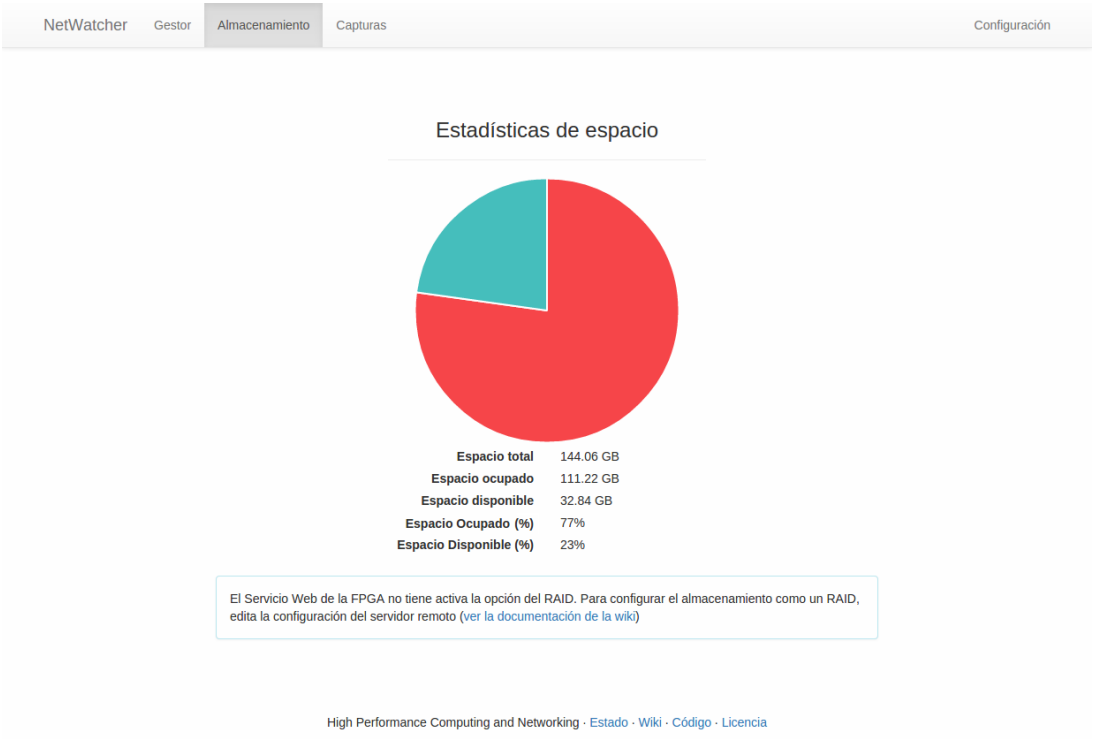


Figura A.8: Página de almacenamiento, con RAID no activo.

A.5.3. Capturas

En esta pantalla se pueden gestionar las trazas almacenadas (ver Figura A.10), mediante una tabla y un panel de acciones. La tabla contiene una fila por cada traza disponible, con su nombre, tipo, tamaño y fecha. Además, una barra superior asociada a esta tabla permite controlar el contenido de la misma mediante las siguientes acciones (de izquierda a derecha): filtrar las trazas que se muestran según su tipo, activar la actualización automática de las trazas disponibles, buscar una traza por su nombre y actualizar manualmente las trazas disponibles. Pulsando sobre una fila de la tabla se seleccionará la traza correspondiente, activándose el panel de acciones. Este panel permite, mediante cada uno de sus subpaneles, las siguientes operaciones:

- **Convertir**: crea una nueva traza a partir de la traza seleccionada cambiando el tipo (si la original tiene formato simple, la convertida tendrá formato pcap, y viceversa).
- **Renombrar**: cambia el nombre de la traza seleccionada al nuevo nombre introducido.
- **Borrar**: borra del disco la traza seleccionada.

Figura A.9: Página de almacenamiento, con RAID activo.

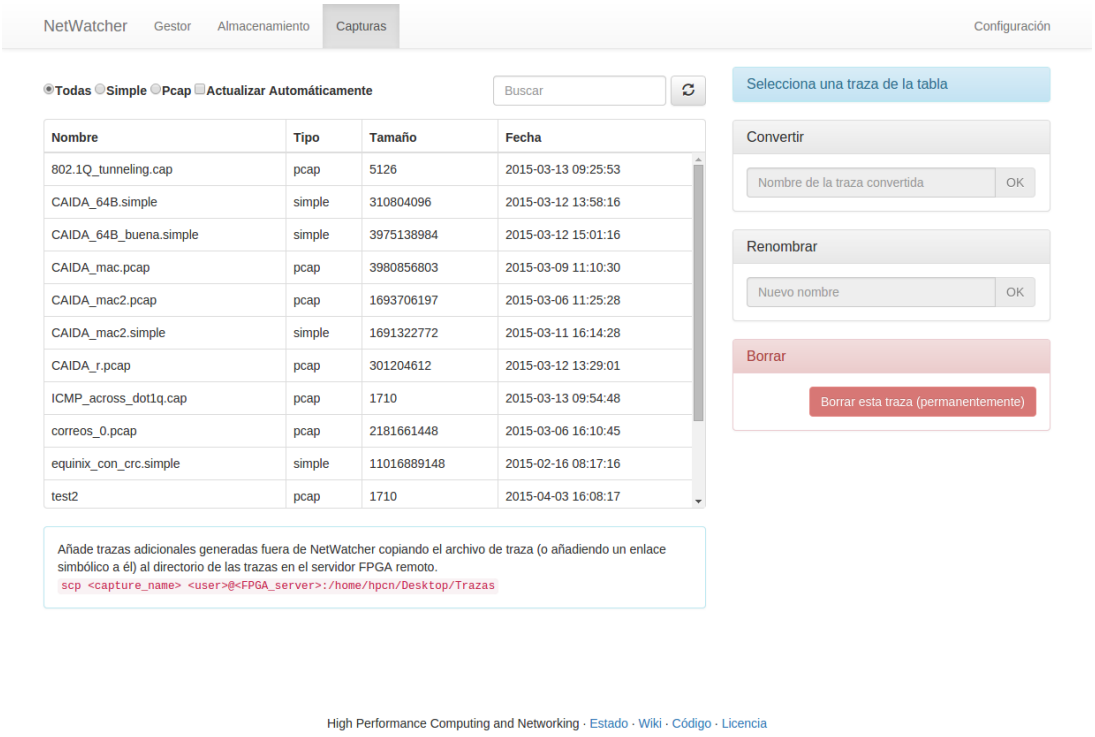


Figura A.10: Página de capturas.

Los dos primeros subpaneles permitirán realizar su acción (se activará el correspondiente botón de *OK*) cuando el campo de texto asociado a cada operación sea introducido correctamente.

A.5.4. Estado

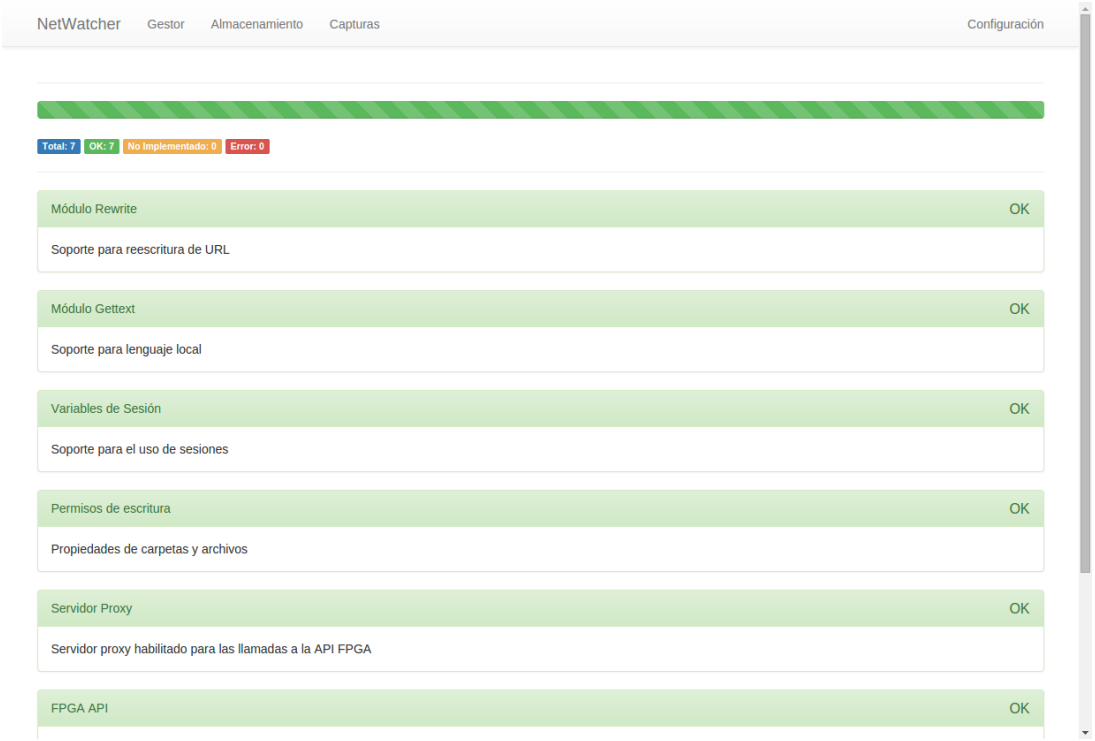


Figura A.11: Página de estado del sistema.

En esta pantalla se puede comprobar el estado de los distintos componentes que forman la aplicación (ver Figura A.11). Cada componente tiene un test asociado cuyo resultado se refleja en un panel. Cada test puede tener tres resultados distintos: *OK* (test pasado, en verde), *No Implementado* (test no implementado, en amarillo) y *Error* (test fallado, en rojo). Los componentes sobre los que se comprueba su estado son los siguientes:

- **Módulo Rewrite**: soporte para reescritura de URL.
- **Módulo Gettext**: soporte para localización (traducción a distintos idiomas).
- **Variables de Sesión**: soporte para el uso de sesiones en PHP.
- **Permisos de escritura**: permisos para escribir logs y archivos de configuración.
- **Servidor Proxy**: servidor proxy habilitado para llamadas a la API FPGA.
- **FPGA API**: servidor de la FPGA activo.

- **Relojes Sincronizados**: diferencia de relojes entre el cliente y el servidor FPGA dentro del umbral permitido.

Adicionalmente, una barra de progreso encima de todo los paneles resume el estado global del sistema.

A.6. Solución de problemas

Si tras seguir las instrucciones paso a paso algo impide el correcto funcionamiento de la aplicación, se puede consultar la página de solución de problemas (en inglés), disponible dentro del repositorio del proyecto en:

github.com/JSidrach/NetWatcher/blob/master/docs/wiki/Troubleshooting.md

B

Framework desarrollado

En este apéndice se explican los distintos componentes del framework desarrollado en el que se ha implementado la interfaz web. Este framework está implementado en PHP sobre *Apache httpd* [31]. Sirve de base para la organización y desarrollo de la aplicación, proveyéndola de un gestor de rutas, un patrón de arquitectura para los módulos de la aplicación (modelo-vista-controlador), un proxy simplificado para las llamadas al Servicio Web FPGA, soporte para la internacionalización de la interfaz, gestión automática de dependencias, y registro de eventos (ver Figura B.1).

B.1. Gestión de rutas

El componente de gestión de rutas del framework se encarga de interpretar las peticiones realizadas por el usuario y delegarlas al módulo apropiado. Con este fin, todas las peticiones HTTP, a excepción de las que van dirigidas al *proxy* (ver sección B.3), se redirigen utilizando *mod_rewrite* [32] al archivo *index.php*. Este archivo, a su vez, invoca el método estático *Router->dispatch()*.

El método *Router->dispatch()* divide la URL solicitada siguiendo el siguiente formato:

URL_BASE_APLICACIÓN/módulo/método/parámetro1/parámetro2/...

Una vez identificadas las partes de la URL, se crea una instancia del módulo correspondiente y se llama al método indicado con los parámetros de la petición (si existiesen), delegándole el control de la solicitud (ver Figura B.2). Por otro lado, si la URL acaba en la URL base de la aplicación, se cargan el módulo y método por defecto; y si la URL acaba en el módulo, se carga el método por defecto (ver sección B.6). En caso de que la URL contenga un módulo o método no existente, se redirecciona la petición a

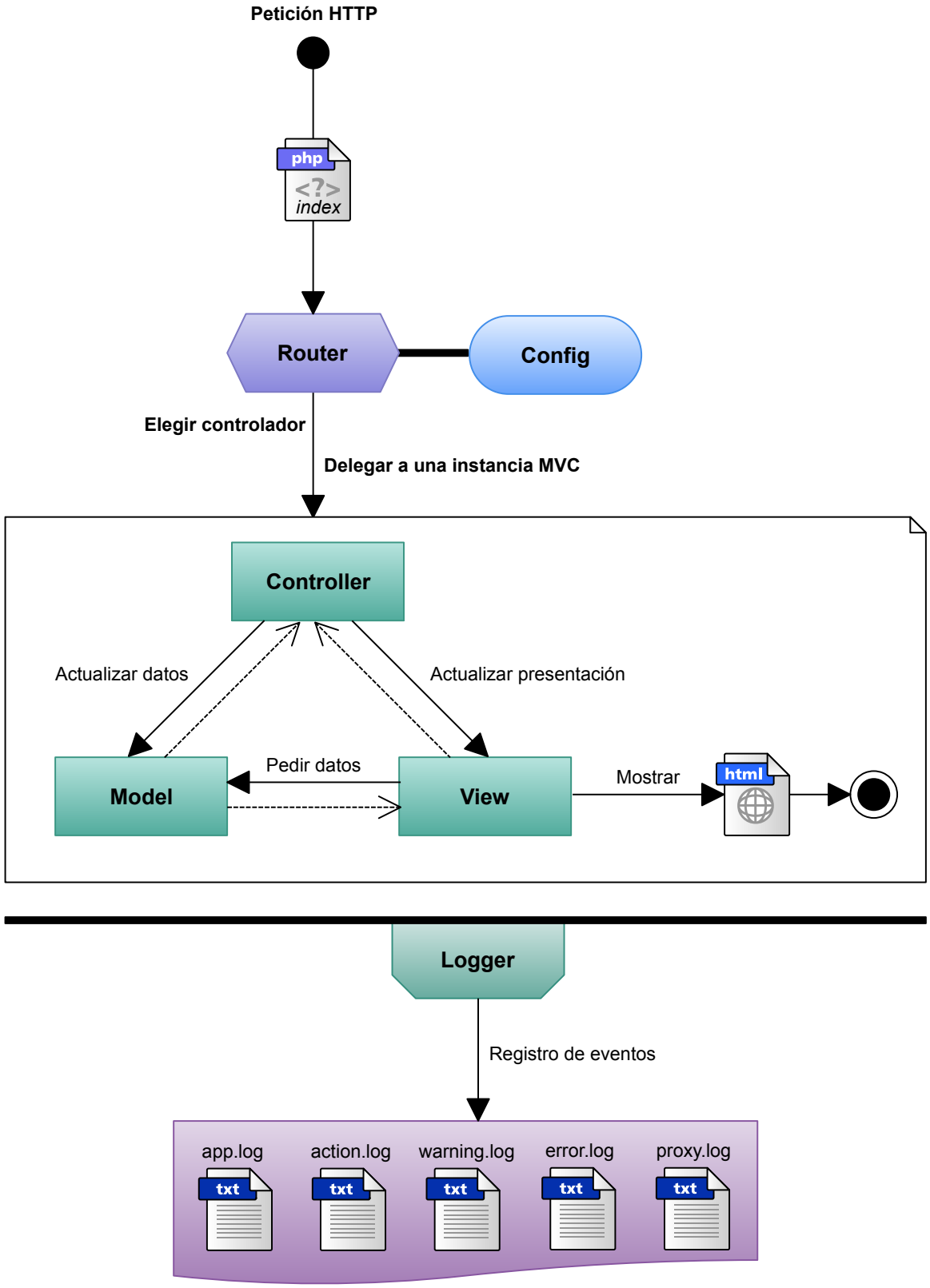


Figura B.1: Arquitectura del framework desarrollado.

la página de *Error 404 - Página no encontrada.*

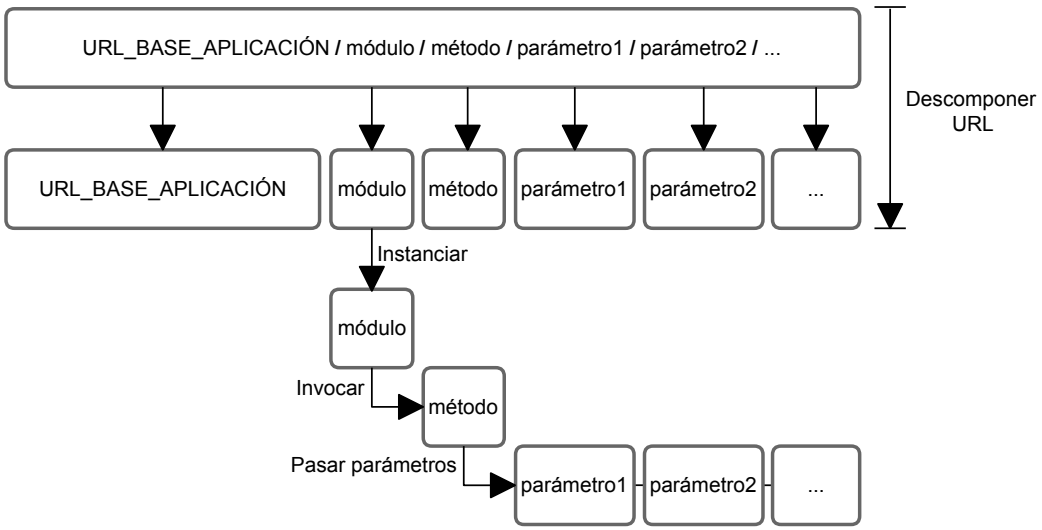


Figura B.2: Esquema de flujo del método *Router->dispatch()*.

B.2. Patrón de arquitectura para los módulos

Las aplicaciones desarrolladas sobre el framework se estructuran en módulos que siguen el patrón de arquitectura modelo-vista-controlador [33]. Este patrón separa la lógica de negocio de la interfaz de usuario, facilitando la evolución por separado de ambos aspectos e incrementando la reutilización y flexibilidad de los módulos de la aplicación.

Los componentes o capas de este patrón de arquitectura son:

- **Modelo**: contiene el estado, los datos y la lógica interna de negocio.
- **Vista**: construye la representación del modelo.
- **Controlador**: gestiona las peticiones del usuario.

Cuando un usuario realiza una acción en la interfaz web, el controlador gestiona la solicitud notificando la acción al modelo, que se actualiza en consecuencia. Posteriormente, el controlador ordena a la vista mostrar al usuario el nuevo estado de la aplicación. La vista consulta entonces el nuevo modelo y lo representa, en el caso de este framework en forma de página web.

Las clases abstractas *Model*, *View* y *Controller* incluidas en el framework proporcionan una implementación base de este patrón. Cada módulo de la aplicación debe heredar cada una de las tres clases e implementar en ellas su funcionalidad propia.

B.3. Redireccionamiento de peticiones AJAX

Por motivos de rendimiento (no sobrecargar el servidor del Servicio Web FPGA incluyéndole también la interfaz) y disponibilidad (la interfaz web debe ser accesible aunque el Servicio Web FPGA no esté operativo), el Servicio Web FPGA y la interfaz web no están alojados en el mismo servidor. Esto plantea un problema, ya que se necesitan realizar peticiones AJAX desde el cliente al Servicio Web FPGA y los navegadores web no permiten, por motivos de seguridad, que un cliente se comunique con un dominio distinto a la web original solicitada [34].

Para resolver esto, el framework contiene un servidor proxy simplificado. Este proxy no es genérico, ya que siempre redirecciona a la dirección IP del Servicio Web FPGA. Así, cuando la interfaz web quiera realizar una petición AJAX al Servicio Web FPGA, hará la petición al servidor proxy, éste redirigirá la petición al Servicio Web FPGA, obtendrá la respuesta y la reenviará de vuelta al cliente (ver Figura B.3).

B.4. Internacionalización de la interfaz

El framework desarrollado facilita la traducción de la aplicación a distintos idiomas mediante la librería *gettext* [35] para PHP. Esta librería proporciona funciones ("_" y "*gettext*") que reciben cadenas de texto y devuelven su equivalente en el idioma de la aplicación seleccionado, consultando un archivo previamente creado (catálogo) que contiene cadenas de texto originales y su traducción.

Es necesario un catálogo por cada idioma que se quiera añadir a la aplicación. Los catálogos se almacenan en la carpeta del proyecto `./locale/`. Para crear y gestionar los catálogos es recomendable utilizar la herramienta *Poedit* [36], que permite editarlos con una interfaz gráfica. Para obtener las cadenas de texto que se necesitan traducir, se escanean todos archivos del proyecto en busca de las palabras clave, y se añaden las cadenas de texto al catálogo para su traducción por parte del desarrollador.

En el Código B.1 se describe un ejemplo de cómo imprimir una cadena traducida a otro idioma mediante una llamada a la función "_". Si el idioma base de los catálogos es el inglés y el idioma establecido en la aplicación es español, la función "_" busca la cadena en inglés (idioma base) *'Example of string'* en el catálogo español y devuelve la cadena de texto *'Ejemplo de cadena'*, que finalmente se imprime con *echo*.

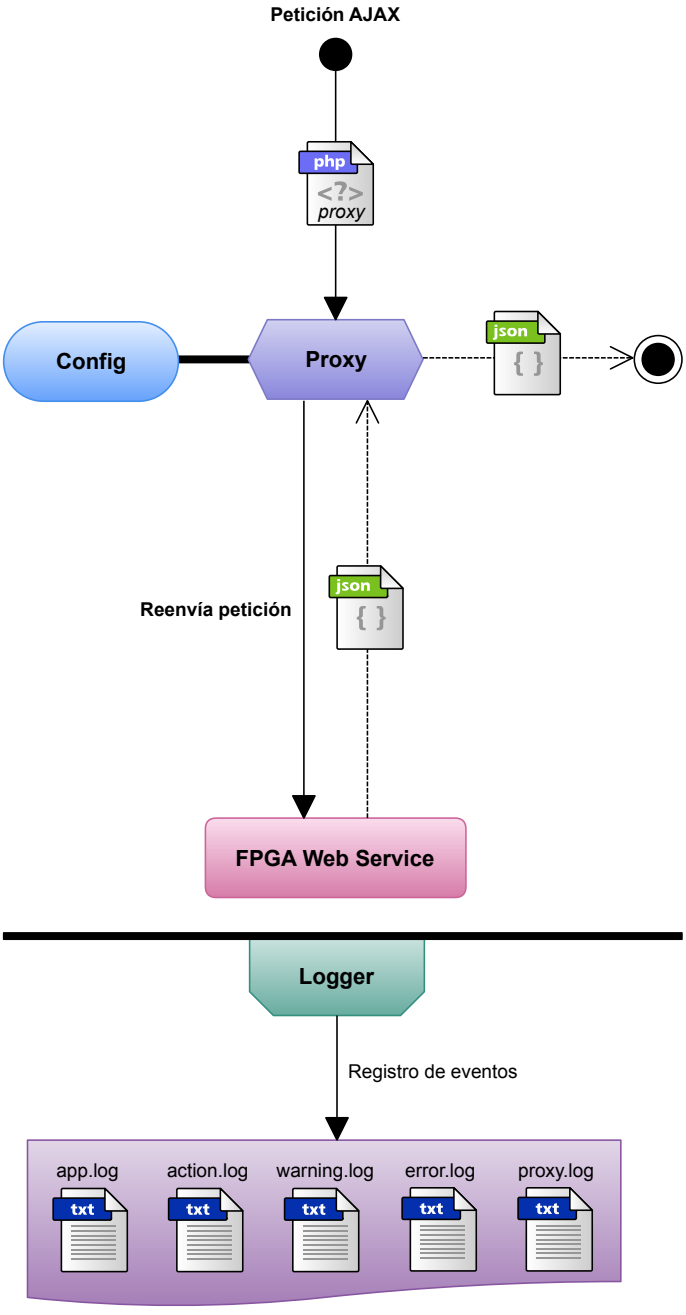


Figura B.3: Arquitectura del proxy desarrollado.

Código B.1: Ejemplo de traducción en PHP con la librería *gettext*

```
/* Set the app language */
putenv('LANG=' . 'es_ES.utf8');
putenv('LANGUAGE=' . 'es_ES.utf8');
setlocale(LC_ALL, 'es_ES.utf8');
bindtextdomain('messages', 'locale');
bind_textdomain_codeset('messages', 'utf-8');
/* Print the example string */
echo _('Example␣of␣string');
```

B.5. Gestión de dependencias

Como en la mayoría de proyectos, es conveniente poder utilizar librerías externas y no tener que invertir tiempo en resolver problemas que ya han sido solucionados por otros anteriormente, siempre que la solución encaje dentro de la propia aplicación. Para manejar la descarga e instalación local de estas librerías externas se han elegido dos gestores de dependencias: *Composer* [37] para el back-end y *BowerPHP* [38] para el front-end. Estos gestores permiten además tener un control sobre la versión exacta necesaria de cada librería, evitando así incompatibilidades.

Composer

Composer es un gestor de dependencias y requisitos back-end para PHP. Las librerías externas back-end que se necesitan para la aplicación se declaran, una vez localizadas en el repositorio de paquetes de *Composer* [39], en el archivo `composer.json` (ver Código B.2). *Composer* posibilita también, siguiendo el estándar PSR-4 [40], incluir en una sola línea de código PHP tanto las dependencias de librerías externas como módulos propios:

```
require_once ('vendor/autoload.php');
```

Las dependencias se descargan e instalan de forma local al proyecto ejecutando el script `./scripts/build.sh --install`.

Código B.2: Ejemplo de fichero *composer.json*

```
{
  "name": "NetWatcher",
  "type": "project",
  "license": "MIT",
  "authors": [
    {
      "name": "JSidrach",
      "email": "juan.sidrach@gmail.com",
      "role": "Developer"
    }
  ],
  "config": {
    "vendor-dir": "vendor/"
   },
  "require": {
    "php": ">=5.5.0",
    "beelab/bowerphp": "dev-master"
  },
  "autoload": {
    "psr-4": {
      "Core\\": "lib/"
    }
  }
}
```

BowerPHP

BowerPHP es una implementación en PHP de *Bower* [41], un gestor de dependencias front-end para aplicaciones web. Las librerías externas que se necesitan para la aplicación se declaran, una vez localizadas en el repositorio de paquetes de *Bower* [42], en el archivo `bower.json` (ver Código B.3). Las dependencias se descargan e instalan de forma local al proyecto ejecutando el script `./scripts/build.sh --install`.

Código B.3: Ejemplo de fichero *bower.json*

```
{
  "name": "NetWatcher",
  "authors": [
    "JSidrach <juan.sidrach@gmail.com>"
  ],
  "private": true,
  "dependencies": {
    "jquery": "2.*",
    "bootstrap": "3.*",
    "bootstrap-table": "1.*",
    "remarkable-bootstrap-notify": "3.*",
    "animate.css": "3.*",
    "chartjs": "1.*"
  }
}
```

B.6. Configuración

Para la configuración interna de la aplicación se utiliza la clase *Config*, compuesta por métodos estáticos. En esta clase se definen también las variables globales de la aplicación: nombres de carpetas y archivos, módulo a cargar por defecto, método a invocar por defecto, dirección IP del Servicio Web FPGA, idiomas disponibles, idioma por defecto, temas visuales disponibles y tema visual por defecto.

Las variables que puede cambiar el usuario (idioma por defecto, tema visual por defecto y dirección IP del Servicio Web FPGA) no se inicializan por definición en el código sino que se leen de distintos ficheros de la carpeta `./config/` en formato JSON.

La configuración global de la aplicación se carga siempre al principio, mediante una llamada al método estático *load* de la clase *Config*.

B.7. Registro de eventos

Registrar todos los eventos asociados a la aplicación es muy útil, ya que permite conocer cómo el usuario utiliza la aplicación y solucionar problemas internos. Para ello, se utiliza la clase *Logger*, que contiene métodos estáticos con los que registrar eventos manualmente dentro del código del proyecto. Adicionalmente, utilizando las funciones estándar de PHP *set_exception_handler* y *set_error_handler*, se redirigen los errores y excepciones a funciones que los registran (de la clase *Logger* también).

Cada evento se guarda en un registro (línea de texto) precedido de la fecha y hora en que se produjo, así como la dirección IP del usuario. Los registros se almacenan en diferentes ficheros dentro de la carpeta `./log/`:

- `app.log`: registro general, contiene todos los eventos de la aplicación.
- `action.log`: contiene registros de los eventos relacionados con acciones del usuario.
- `proxy.log`: contiene registros de los eventos relacionados con las peticiones al módulo proxy.
- `warning.log`: contiene registros de los eventos relacionados con avisos y advertencias.
- `error.log`: contiene registros de los errores de la aplicación.

B.8. Scripts adicionales

Para automatizar tareas que se realizan con frecuencia en el desarrollo de la aplicación, este framework contiene una carpeta (`./scripts/`) con scripts con este propósito, que pueden ser invocados desde la carpeta raíz del proyecto ejecutando el archivo `./scripts/build.sh` con distintos parámetros:

- `--doc`: genera la documentación automática a partir del código del front-end y del back-end, en la carpeta `./docs/`.
- `--upgrade`: actualiza las librerías externas necesarias para la aplicación.
- `--install`: instala las dependencias externas (paquetes y librerías) necesarias para la aplicación.
- `--check`: realiza un análsis léxico y sintáctico sobre todo el código PHP de la aplicación.
- `--permissions`: otorga los permisos mínimos necesarios de lectura/escritura/ejecución a los archivos y carpetas del proyecto.
- `--clear`: borra los archivos de logs.
- `--backup`: comprime la carpeta del proyecto en un archivo en formato *.zip*, y lo guarda con la fecha actual en la carpeta superior, a modo de copia de seguridad.

B.9. Conclusiones

Se ha desarrollado un framework que sirve de base para la interfaz web, proporcionando un conjunto mínimo de funcionalidad necesaria para el problema planteado. Aunque desarrollarlo ha supuesto un coste temporal adicional para el proyecto, ha repercutido positivamente en fases posteriores de la implementación.

Conocer al detalle el framework sobre el que se basa la aplicación y tener un control total sobre el mismo ha permitido agilizar el proceso de desarrollo. Además, se ha adquirido experiencia en distintos conceptos útiles: orientación a objetos en PHP, patrón de diseño modelo-vista-controlador, gestión automática de dependencias y librerías externas, internacionalización de interfaces web y codificación de un servidor proxy simplificado.

C

API del Servicio Web FPGA

En este apéndice se detallan los métodos de la API del Servicio Web FPGA. Esta documentación puede también consultarse de manera interactiva (Figura C.1) en la propia página de la aplicación (en inglés), accediendo desde el navegador a la ruta relativa *NetWatcher/docs-back-end/*. Al ser un Servicio Web, cada método se invoca mediante una llamada HTTP [43].

Se han agrupado todos los métodos disponibles en tres categorías, coincidiendo con los módulos implementados: gestión, trazas y estadísticas. Para cada método, se especifica su ruta (relativa a la ruta del Servicio Web), los parámetros necesarios para su invocación y las posibles salidas, con un ejemplo cada una. Los parámetros se pasan por la propia URL o en la cabecera en el caso del *timestamp*. Cada método devuelve siempre un código de estado HTTP [44] y una salida en formato JSON [45].

C.1. Métodos de Gestión

C.1.1. POST /system/reboot

Reinicia el servidor remoto. Los parámetros necesarios para la invocación de este método se detallan en la Tabla C.1.

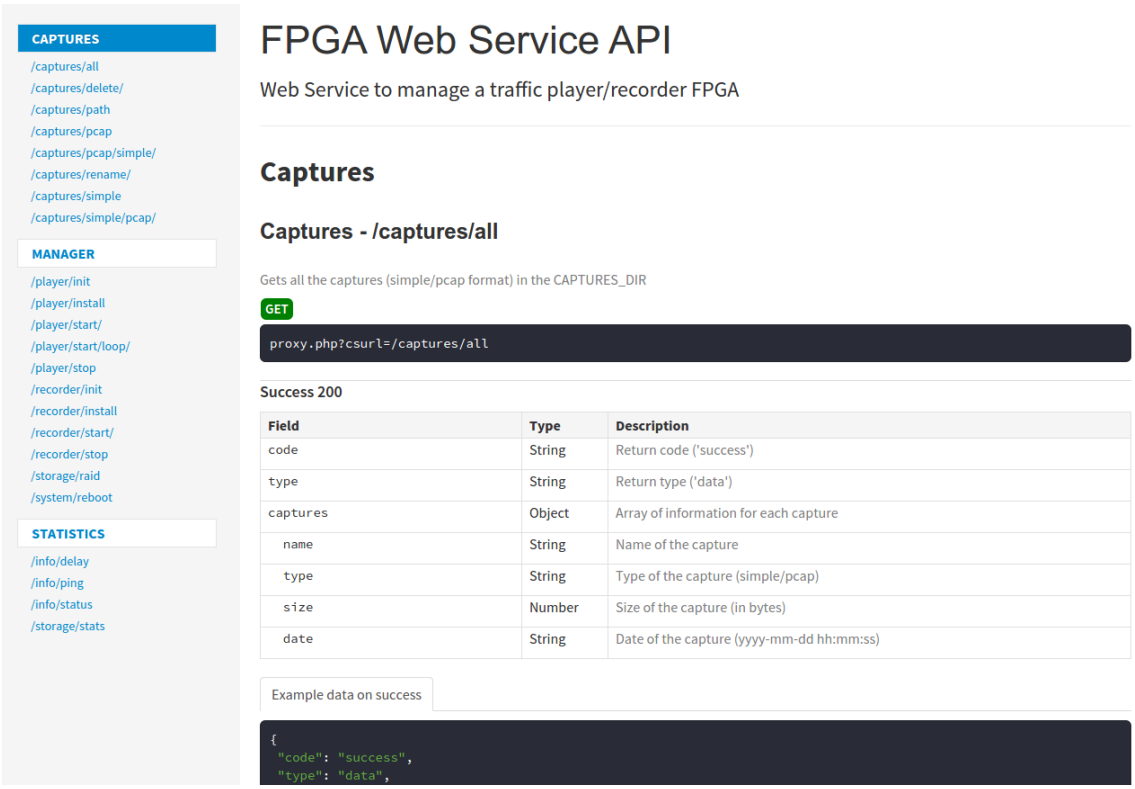


Figura C.1: Documentación web de la API del Servicio Web FPGA.

Parámetro	Clase	Tipo	Descripción
timestamp	Cabecera	Número	Tiempo transcurrido (ms) desde el 1 de Enero de 1970 00:00:00 UTC hasta ahora (salida de *Date.now()*)

Tabla C.1: Parámetros de */system/reboot*

A continuación se enumeran los distintos códigos de retorno asociados a este método:

- **200 OK:** se ha ordenado con éxito el reinicio del servidor remoto. Este código de éxito es retornado junto con los parámetros indicados en la Tabla C.2.

Nombre	Tipo	Descripción
code	Cadena de texto	Código de retorno ('success')
type	Cadena de texto	Código de tipo ('notification')
description	Cadena de texto	Descripción del código de retorno

Tabla C.2: Salida de */system/reboot* asociada al código 200

Ejemplo de datos asociados al código 200:

```
{
  "code": "success",
  "type": "notification",
  "description": "The host is rebooting now."
}
```

- **412 Error:** el servidor remoto no puede ser reiniciado ya que la FPGA está siendo usada. Este código de error es retornado junto a los parámetros indicados en la Tabla C.3.

Nombre	Tipo	Descripción
code	Cadena de texto	Código de retorno ('error')
type	Cadena de texto	Código de tipo ('notification')
description	Cadena de texto	Descripción del error

Tabla C.3: Salida de */system/reboot* asociada al código 412

Ejemplo de datos asociados al código 412:

```
{
  "code": "error",
  "type": "notification",
  "description": "The host can not be rebooted. The FPGA
    ↪ is being used."
}
```

C.1.2. POST /player/init

Programa la FPGA para reproducir trazas y reinicia el servidor remoto. Los parámetros necesarios para la invocación de este método se detallan en la Tabla C.4.

Parámetro	Clase	Tipo	Descripción
timestamp	Cabecera	Número	Tiempo transcurrido (ms) desde el 1 de Enero de 1970 00:00:00 UTC hasta ahora (salida de *Date.now()*)

Tabla C.4: Parámetros de */player/init*

A continuación se enumeran los distintos códigos de retorno asociados a este método:

- **200 OK:** se ha programado con éxito la FPGA para reproducir trazas y se va a reiniciar el servidor remoto. Este código de éxito es retornado junto con los parámetros indicados en la Tabla C.5.

Nombre	Tipo	Descripción
code	Cadena de texto	Código de retorno ('success')
type	Cadena de texto	Código de tipo ('notification')
description	Cadena de texto	Descripción del código de retorno

Tabla C.5: Salida de */player/init* asociada al código 200

Ejemplo de datos asociados al código 200:

```
{
  "code": "success",
  "type": "notification",
  "description": "The FPGA has been initialized. The host
    ↪  will reboot now."
}
```

- **412 Error:** la FPGA no puede ser programada ya que está siendo usada. Este código de error es retornado junto a los parámetros indicados en la Tabla C.6.

Nombre	Tipo	Descripción
code	Cadena de texto	Código de retorno ('error')
type	Cadena de texto	Código de tipo ('fpga_invalid_state')
description	Cadena de texto	Descripción del error

Tabla C.6: Salida de */player/init* asociada al código 412

Ejemplo de datos asociados al código 412:

```
{
  "code": "error",
  "type": "fpga_invalid_state",
  "description": "Invalid State. The FPGA is already
    ↪ running, stop it to init the FPGA."
}
```

C.1.3. POST /recorder/init

Programa la FPGA para capturar trazas y reinicia el servidor remoto. Los parámetros necesarios para la invocación de este método se detallan en la Tabla C.7.

Parámetro	Clase	Tipo	Descripción
timestamp	Cabecera	Número	Tiempo transcurrido (ms) desde el 1 de Enero de 1970 00:00:00 UTC hasta ahora (salida de *Date.now()*)

Tabla C.7: Parámetros de */recorder/init*

A continuación se enumeran los distintos códigos de retorno asociados a este método:

- **200 OK:** se ha programado con éxito la FPGA para reproducir trazas y se va a reiniciar el servidor remoto. Este código de éxito es retornado junto con los parámetros indicados en la Tabla C.8.

Nombre	Tipo	Descripción
code	Cadena de texto	Código de retorno ('success')
type	Cadena de texto	Código de tipo ('notification')
description	Cadena de texto	Descripción del código de retorno

Tabla C.8: Salida de */recorder/init* asociada al código 200

Ejemplo de datos asociados al código 200:

```
{
  "code": "success",
  "type": "notification",
  "description": "The FPGA has been initialized. The host
    ↪  will reboot now."
}
```

- **412 Error:** la FPGA no puede ser programada ya que está siendo usada. Este código de error es retornado junto a los parámetros indicados en la Tabla C.9.

Nombre	Tipo	Descripción
code	Cadena de texto	Código de retorno ('error')
type	Cadena de texto	Código de tipo ('fpga_invalid_state')
description	Cadena de texto	Descripción del error

Tabla C.9: Salida de */recorder/init* asociada al código 412

Ejemplo de datos asociados al código 412:

```
{
  "code": "error",
  "type": "fpga_invalid_state",
  "description": "Invalid State. The FPGA is already
    ↪ running, stop it to init the FPGA."
}
```

C.1.4. POST /player/install

Instala y monta la FPGA para reproducir trazas. Los parámetros necesarios para la invocación de este método se detallan en la Tabla C.10.

Parámetro	Clase	Tipo	Descripción
timestamp	Cabecera	Número	Tiempo transcurrido (ms) desde el 1 de Enero de 1970 00:00:00 UTC hasta ahora (salida de *Date.now()*)

Tabla C.10: Parámetros de */player/install*

A continuación se enumeran los distintos códigos de retorno asociados a este método:

- **200 OK:** la FPGA se ha instalado y montado con éxito, y está lista para reproducir trazas. Este código de éxito es retornado junto con los parámetros indicados en la Tabla C.11.

Nombre	Tipo	Descripción
code	Cadena de texto	Código de retorno ('success')
type	Cadena de texto	Código de tipo ('notification')
description	Cadena de texto	Descripción del código de retorno

Tabla C.11: Salida de */player/install* asociada al código 200

Ejemplo de datos asociados al código 200:

```
{
  "code": "success",
  "type": "notification",
  "description": "The FPGA has been mounted and is ready
    ↪ to be used."
}
```

- **412 Error:** la FPGA no puede ser instalada y montada ya que no ha sido programada. Este código de error es retornado junto a los parámetros indicados en la Tabla C.12.

Nombre	Tipo	Descripción
code	Cadena de texto	Código de retorno ('error')
type	Cadena de texto	Código de tipo ('fpga_invalid_state')
description	Cadena de texto	Descripción del error

Tabla C.12: Salida de */player/install* asociada al código 412

Ejemplo de datos asociados al código 412:

```
{
  "code": "error",
  "type": "fpga_invalid_state",
  "description": "Invalid State. The FPGA must be
    ↪ programmed before mounted."
}
```

C.1.5. POST /recorder/install

Instala y monta la FPGA para capturar trazas. Los parámetros necesarios para la invocación de este método se detallan en la Tabla C.13.

Parámetro	Clase	Tipo	Descripción
timestamp	Cabecera	Número	Tiempo transcurrido (ms) desde el 1 de Enero de 1970 00:00:00 UTC hasta ahora (salida de *Date.now()*)

Tabla C.13: Parámetros de */recorder/install*

A continuación se enumeran los distintos códigos de retorno asociados a este método:

- **200 OK:** la FPGA se ha instalado y montado con éxito, y está lista para capturar trazas. Este código de éxito es retornado junto con los parámetros indicados en la Tabla C.14.

Nombre	Tipo	Descripción
code	Cadena de texto	Código de retorno ('success')
type	Cadena de texto	Código de tipo ('notification')
description	Cadena de texto	Descripción del código de retorno

Tabla C.14: Salida de */recorder/install* asociada al código 200

Ejemplo de datos asociados al código 200:

```
{
  "code": "success",
  "type": "notification",
  "description": "The FPGA has been mounted and is ready
    ↪ to be used."
}
```

- **412 Error:** la FPGA no puede ser instalada y montada ya que no ha sido programada. Este código de error es retornado junto a los parámetros indicados en la Tabla C.15.

Nombre	Tipo	Descripción
code	Cadena de texto	Código de retorno ('error')
type	Cadena de texto	Código de tipo ('fpga_invalid_state')
description	Cadena de texto	Descripción del error

Tabla C.15: Salida de */recorder/install* asociada al código 412

Ejemplo de datos asociados al código 412:

```
{
  "code": "error",
  "type": "fpga_invalid_state",
  "description": "Invalid State. The FPGA must be
    ↪ programmed before mounted."
}
```

C.1.6. POST /player/start/:capturename/:mask/:ifg

Reproduce una traza con los parámetros indicados. Los parámetros necesarios para la invocación de este método se detallan en la Tabla C.16.

Parámetro	Clase	Tipo	Descripción
timestamp	Cabecera	Número	Tiempo transcurrido (ms) desde el 1 de Enero de 1970 00:00:00 UTC hasta ahora (salida de *Date.now()*)
capturename	Parámetro	Cadena de texto	Nombre de la traza a reproducir
mask	Parámetro	Número	Máscara de reproducción (0-1-2-3)
ifg	Parámetro	Número	IFG (0 para tasa original)

Tabla C.16: Parámetros de */player/start*

A continuación se enumeran los distintos códigos de retorno asociados a este método:

- **200 OK:** la FPGA ha empezado a reproducir la traza seleccionada con los parámetros indicados. Este código de éxito es retornado junto con los parámetros indicados en la Tabla C.17.

Nombre	Tipo	Descripción
code	Cadena de texto	Código de retorno ('success')
type	Cadena de texto	Código de tipo ('notification')
description	Cadena de texto	Descripción del código de retorno

Tabla C.17: Salida de */player/start* asociada al código 200

Ejemplo de datos asociados al código 200:

```
{
  "code": "success",
  "type": "notification",
  "description": "The FPGA has started playing a capture
    ↪ ."
}
```

- **400 Error:** la FPGA no puede reproducir la traza ya que los parámetros pasados son inválidos. Este código de error es retornado junto a los parámetros indicados en la Tabla C.18.

Nombre	Tipo	Descripción
code	Cadena de texto	Código de retorno ('error')
type	Cadena de texto	Código de tipo ('fpga_invalid_state')
description	Cadena de texto	Descripción del error

Tabla C.18: Salida de */player/start* asociada al código 400

Ejemplo de datos asociados al código 400:

```
{
  "code": "error",
  "type": "notification",
  "description": "Invalid parameters. The FPGA could not
    ↪ start playing a capture."
}
```

- **412 Error:** la FPGA no puede reproducir la traza ya que no ha sido instalada y montada para reproducir trazas. Este código de error es retornado junto a los parámetros indicados en la Tabla C.19.

Nombre	Tipo	Descripción
code	Cadena de texto	Código de retorno ('error')
type	Cadena de texto	Código de tipo ('fpga_invalid_state')
description	Cadena de texto	Descripción del error

Tabla C.19: Salida de */player/start* asociada al código 412

Ejemplo de datos asociados al código 412:

```
{
  "code": "error",
  "type": "fpga_invalid_state",
  "description": "Invalid State. The FPGA is not
    ↪ programmed and mounted as a player."
}
```

C.1.7. POST /player/start/loop/:capturename/:mask/:ifg

Reproduce en bucle una traza con los parámetros indicados. Los parámetros necesarios para la invocación de este método se detallan en la Tabla C.20.

Parámetro	Clase	Tipo	Descripción
timestamp	Cabecera	Número	Tiempo transcurrido (ms) desde el 1 de Enero de 1970 00:00:00 UTC hasta ahora (salida de *Date.now()*)
capturename	Parámetro	Cadena de texto	Nombre de la traza a reproducir
mask	Parámetro	Número	Máscara de reproducción (0-1-2-3)
ifg	Parámetro	Número	IFG (0 para tasa original)

Tabla C.20: Parámetros de */player/start/loop*

A continuación se enumeran los distintos códigos de retorno asociados a este método:

- **200 OK:** la FPGA ha empezado a reproducir en bucle la traza seleccionada con los parámetros indicados. Este código de éxito es retornado junto con los parámetros indicados en la Tabla C.21.

Nombre	Tipo	Descripción
code	Cadena de texto	Código de retorno ('success')
type	Cadena de texto	Código de tipo ('notification')
description	Cadena de texto	Descripción del código de retorno

Tabla C.21: Salida de */player/start/loop* asociada al código 200

Ejemplo de datos asociados al código 200:

```
{
  "code": "success",
  "type": "notification",
  "description": "The FPGA has started playing a capture
    ↪ ."
}
```

- **400 Error:** la FPGA no puede reproducir la traza ya que los parámetros pasados son inválidos. Este código de error es retornado junto a los parámetros indicados en la Tabla C.22.

Nombre	Tipo	Descripción
code	Cadena de texto	Código de retorno ('error')
type	Cadena de texto	Código de tipo ('fpga_invalid_state')
description	Cadena de texto	Descripción del error

Tabla C.22: Salida de */player/start/loop* asociada al código 400

Ejemplo de datos asociados al código 400:

```
{
  "code": "error",
  "type": "notification",
  "description": "Invalid parameters. The FPGA could not
    ↪ start playing a capture."
}
```

- **412 Error:** la FPGA no puede reproducir la traza ya que no ha sido instalada y montada para reproducir trazas. Este código de error es retornado junto a los parámetros indicados en la Tabla C.23.

Nombre	Tipo	Descripción
code	Cadena de texto	Código de retorno ('error')
type	Cadena de texto	Código de tipo ('fpga_invalid_state')
description	Cadena de texto	Descripción del error

Tabla C.23: Salida de */player/start/loop* asociada al código 412

Ejemplo de datos asociados al código 412:

```
{
  "code": "error",
  "type": "fpga_invalid_state",
  "description": "Invalid State. The FPGA is not
    ↪ programmed and mounted as a player."
}
```

C.1.8. POST /recorder/start/:capturename/:port/:bytes

Captura una traza con los parámetros indicados. Los parámetros necesarios para la invocación de este método se detallan en la Tabla C.24.

Parámetro	Clase	Tipo	Descripción
timestamp	Cabecera	Número	Tiempo transcurrido (ms) desde el 1 de Enero de 1970 00:00:00 UTC hasta ahora (salida de *Date.now()*)
capturename	Parámetro	Cadena de texto	Nombre de la traza a capturar
port	Parámetro	Número	Puerto a capturar (0-1-2-3)
bytes	Parámetro	Número	Bytes a capturar

Tabla C.24: Parámetros de */recorder/start*

A continuación se enumeran los distintos códigos de retorno asociados a este método:

- **200 OK:** la FPGA ha empezado a capturar una traza con los parámetros indicados. Este código de éxito es retornado junto con los parámetros indicados en la Tabla C.25.

Nombre	Tipo	Descripción
code	Cadena de texto	Código de retorno ('success')
type	Cadena de texto	Código de tipo ('notification')
description	Cadena de texto	Descripción del código de retorno

Tabla C.25: Salida de */recorder/start* asociada al código 200

Ejemplo de datos asociados al código 200:

```
{
  "code": "success",
  "type": "notification",
  "description": "The FPGA has started recording data."
}
```

- **400 Error:** la FPGA no puede empezar a capturar ya que los parámetros pasados son inválidos. Este código de error es retornado junto a los parámetros indicados en la Tabla C.26.

Nombre	Tipo	Descripción
code	Cadena de texto	Código de retorno ('error')
type	Cadena de texto	Código de tipo ('fpga_invalid_state')
description	Cadena de texto	Descripción del error

Tabla C.26: Salida de */recorder/start* asociada al código 400

Ejemplo de datos asociados al código 400:

```
{
  "code": "error",
  "type": "notification",
  "description": "Invalid capture name (must not exist)."
}
```

- **412 Error:** la FPGA no puede empezar a capturar ya que no ha sido instalada y montada para capturar trazas. Este código de error es retornado junto a los parámetros indicados en la Tabla C.27.

Nombre	Tipo	Descripción
code	Cadena de texto	Código de retorno ('error')
type	Cadena de texto	Código de tipo ('fpga_invalid_state')
description	Cadena de texto	Descripción del error

Tabla C.27: Salida de */recorder/start* asociada al código 412

Ejemplo de datos asociados al código 412:

```
{
  "code": "error",
  "type": "fpga_invalid_state",
  "description": "Invalid State. The FPGA is not
    ↪ programmed and mounted as a recorder."
}
```

C.1.9. POST /player/stop

Detiene la reproducción de traza en curso. Los parámetros necesarios para la invocación de este método se detallan en la Tabla C.28.

Parámetro	Clase	Tipo	Descripción
timestamp	Cabecera	Número	Tiempo transcurrido (ms) desde el 1 de Enero de 1970 00:00:00 UTC hasta ahora (salida de *Date.now()*)

Tabla C.28: Parámetros de */player/stop*

A continuación se enumeran los distintos códigos de retorno asociados a este método:

- **200 OK:** la FPGA ha detenido la reproducción en curso. Este código de éxito es retornado junto con los parámetros indicados en la Tabla C.29.

Nombre	Tipo	Descripción
code	Cadena de texto	Código de retorno ('success')
type	Cadena de texto	Código de tipo ('notification')
description	Cadena de texto	Descripción del código de retorno

Tabla C.29: Salida de */player/stop* asociada al código 200

Ejemplo de datos asociados al código 200:

```
{
  "code": "success",
  "type": "notification",
  "description": "The FPGA has stopped playing a capture
    ↪ ."
}
```

- **412 Error:** no se ha podido detener la reproducción ya que no existe ninguna en curso. Este código de error es retornado junto a los parámetros indicados en la Tabla C.30.

Nombre	Tipo	Descripción
code	Cadena de texto	Código de retorno ('error')
type	Cadena de texto	Código de tipo ('fpga_invalid_state')
description	Cadena de texto	Descripción del error

Tabla C.30: Salida de */player/stop* asociada al código 412

Ejemplo de datos asociados al código 412:

```
{
  "code": "error",
  "type": "fpga_invalid_state",
  "description": "Invalid State. The FPGA is not playing
    ↪ a capture."
}
```

C.1.10. POST /recorder/stop

Detiene la captura de traza en curso. Los parámetros necesarios para la invocación de este método se detallan en la Tabla C.31.

Parámetro	Clase	Tipo	Descripción
timestamp	Cabecera	Número	Tiempo transcurrido (ms) desde el 1 de Enero de 1970 00:00:00 UTC hasta ahora (salida de *Date.now()*)

Tabla C.31: Parámetros de */recorder/stop*

A continuación se enumeran los distintos códigos de retorno asociados a este método:

- **200 OK:** la FPGA ha detenido la captura en curso. Este código de éxito es retornado junto con los parámetros indicados en la Tabla C.32.

Nombre	Tipo	Descripción
code	Cadena de texto	Código de retorno ('success')
type	Cadena de texto	Código de tipo ('notification')
description	Cadena de texto	Descripción del código de retorno

Tabla C.32: Salida de */recorder/stop* asociada al código 200

Ejemplo de datos asociados al código 200:

```
{
  "code": "success",
  "type": "notification",
  "description": "The FPGA has stopped recording data."
}
```

- **412 Error:** no se ha podido detener la captura ya que no existe ninguna en curso. Este código de error es retornado junto a los parámetros indicados en la Tabla C.33.

Nombre	Tipo	Descripción
code	Cadena de texto	Código de retorno ('error')
type	Cadena de texto	Código de tipo ('fpga_invalid_state')
description	Cadena de texto	Descripción del error

Tabla C.33: Salida de */recorder/stop* asociada al código 412

Ejemplo de datos asociados al código 412:

```
{
  "code": "error",
  "type": "fpga_invalid_state",
  "description": "Invalid State. The FPGA is not
    ↪ recording data."
}
```

C.1.11. DELETE /storage/raid

Borra (formatea y vuelve a crear) el RAID de almacenamiento. Los parámetros necesarios para la invocación de este método se detallan en la Tabla C.34.

Parámetro	Clase	Tipo	Descripción
timestamp	Cabecera	Número	Tiempo transcurrido (ms) desde el 1 de Enero de 1970 00:00:00 UTC hasta ahora (salida de *Date.now()*)

Tabla C.34: Parámetros de */storage/raid*

A continuación se enumeran los distintos códigos de retorno asociados a este método:

- **200 OK:** se ha formateado y vuelto a montar el RAID de almacenamiento. Este código de éxito es retornado junto con los parámetros indicados en la Tabla C.35.

Nombre	Tipo	Descripción
code	Cadena de texto	Código de retorno ('success')
type	Cadena de texto	Código de tipo ('notification')
description	Cadena de texto	Descripción del código de retorno

Tabla C.35: Salida de */storage/raid* asociada al código 200

Ejemplo de datos asociados al código 200:

```
{
  "code": "success",
  "type": "notification",
  "description": "The RAID has been formatted and mounted
    ↪  properly."
}
```

- **412 Error:** no se ha formateado el RAID ya que o está activado o la FPGA está en uso. Este código de error es retornado junto a los parámetros indicados en la Tabla C.36.

Nombre	Tipo	Descripción
code	Cadena de texto	Código de retorno ('error')
type	Cadena de texto	Código de tipo ('fpga_invalid_state')
description	Cadena de texto	Descripción del error

Tabla C.36: Salida de */storage/raid* asociada al código 412

Ejemplo de datos asociados al código 412:

```
{
  "code": "error",
  "type": "notification",
  "description": "The RAID could not be formatted, RAID
    ↪ configuration option is not set or the FPGA is
    ↪ running."
}
```

C.2. Métodos de Estadísticas

C.2.1. GET /info/ping

Método para comprobar si el servidor está operativo. Este método no requiere parámetros.

A continuación se enumeran los distintos códigos de retorno asociados a este método:

- **200 OK:** servidor operativo. Este código de éxito es retornado junto con los parámetros indicados en la Tabla C.37.

Nombre	Tipo	Descripción
code	Cadena de texto	Código de retorno ('success')

Tabla C.37: Salida de */info/ping* asociada al código 200

Ejemplo de datos asociados al código 200:

```
{
  "code": "success"
}
```

C.2.2. GET /info/delay

Solicita la diferencia de tiempos existente entre los relojes del cliente y del servidor (en segundos). Los parámetros necesarios para la invocación de este método se detallan en la Tabla C.38.

Parámetro	Clase	Tipo	Descripción
timestamp	Cabecera	Número	Tiempo transcurrido (ms) desde el 1 de Enero de 1970 00:00:00 UTC hasta ahora (salida de *Date.now()*)

Tabla C.38: Parámetros de */info/delay*

A continuación se enumeran los distintos códigos de retorno asociados a este método:

- **200 OK:** solicitud con éxito. Este código de éxito es retornado junto con los parámetros indicados en la Tabla C.39.

Nombre	Tipo	Descripción
code	Cadena de texto	Código de retorno ('success')
type	Cadena de texto	Código de tipo ('data')
delay	Número	Diferencia entre relojes (en segundos)
maxDelay	Número	Máxima diferencia permitida (en segundos)

Tabla C.39: Salida de */info/delay* asociada al código 200

Ejemplo de datos asociados al código 200:

```
{
  "code": "success",
  "type": "data",
  "delay": 1,
  "maxDelay": 30
}
```

C.2.3. GET /info/status

Solicita el estado actual de la FPGA. Este método no requiere parámetros.

A continuación se enumeran los distintos códigos de retorno asociados a este método:

- **200 OK - hugepages_off:** opción *HugePages* del sistema operativo no activa. Este código de éxito es retornado junto con los parámetros indicados en la Tabla C.40.

Nombre	Tipo	Descripción
code	Cadena de texto	Código de estado ('hugepages_off')
description	Cadena de texto	Descripción del código de estado

Tabla C.40: Salida de */info/status* asociada al código 200 - hugepages_off

Ejemplo de datos asociados al código 200 - hugepages_off:

```
{
  "status": "hugepages_off",
  "description": "HugePages is not enabled. To fix this,
    ↪ the host should be rebooted with this option
    ↪ selected on the GRUB menu."
}
```

- **200 OK - init_off:** la FPGA aún no ha sido configurada para captura o reproducción. Este código de éxito es retornado junto con los parámetros indicados en la Tabla C.41.

Nombre	Tipo	Descripción
code	Cadena de texto	Código de estado ('init_off')
description	Cadena de texto	Descripción del código de estado

Tabla C.41: Salida de */info/status* asociada al código 200 - init_off

Ejemplo de datos asociados al código 200 - init_off:

```
{
  "status": "init_off",
  "description": "The FPGA is not configured either as
    ↪ player or as recorder yet."
}
```

- **200 OK - mount_off:** la FPGA está inicializada pero no ha sido montada aún. Este código de éxito es retornado junto con los parámetros indicados en la Tabla C.42.

Nombre	Tipo	Descripción
code	Cadena de texto	Código de estado ('mount_off')
description	Cadena de texto	Descripción del código de estado

Tabla C.42: Salida de */info/status* asociada al código 200 - mount_off

Ejemplo de datos asociados al código 200 - mount_off:

```
{
  "status": "mount_off",
  "description": "The FPGA is initialized but not mounted
    ↪ ."
}
```

- **200 OK - player_ready:** la FPGA está lista para reproducir una traza. Este código de éxito es retornado junto con los parámetros indicados en la Tabla C.43.

Nombre	Tipo	Descripción
code	Cadena de texto	Código de estado ('player_ready')
description	Cadena de texto	Descripción del código de estado

Tabla C.43: Salida de */info/status* asociada al código 200 - player_ready

Ejemplo de datos asociados al código 200 - player_ready:

```
{
  "status": "player_ready",
  "description": "The FPGA is ready to reproduce a
    ↪ capture."
}
```

- **200 OK - recorder_ready:** la FPGA está lista para capturar una traza. Este código de éxito es retornado junto con los parámetros indicados en la Tabla C.44.

Nombre	Tipo	Descripción
code	Cadena de texto	Código de estado ('recorder_ready')
description	Cadena de texto	Descripción del código de estado

Tabla C.44: Salida de */info/status* asociada al código 200 - recorder_ready

Ejemplo de datos asociados al código 200 - recorder_ready:

```
{
  "status": "player_ready",
  "description": "The FPGA is ready to record a capture."
}
```

- **200 OK - playing:** la FPGA está reproduciendo una traza. Este código de éxito es retornado junto con los parámetros indicados en la Tabla C.45.

Nombre	Tipo	Descripción
code	Cadena de texto	Código de estado ('playing')
description	Cadena de texto	Descripción del código de estado
capture	Cadena de texto	Nombre de la traza en reproducción
size	Número	Tamaño de la traza en reproducción
date	Cadena de texto	Fecha de la traza en reproducción
elapsed_time	Número	Tiempo transcurrido desde el inicio (segundos)
packets_sent	Número	Paquetes enviados
loop	Booleano	*true* si se está reproduciendo en bucle
interframe_gap	Número	IFG (0 si es la tasa original)
mask	Número	Máscara de reproducción (0-1-2-3)

Tabla C.45: Salida de */info/status* asociada al código 200 - playing

Ejemplo de datos asociados al código 200 - playing:

```
{
  "status": "playing",
  "description": "The FPGA is reproducing a capture.",
  "capture": "my_capture.simple",
  "size": 714131923845,
  "date": "2014-09-29 15:40:34",
  "elapsed_time": 548,
  "packets_sent": 394578123,
  "loop": true,
  "interframe_gap": 0,
  "mask": 3
}
```

- **200 OK - recording:** la FPGA está capturando una traza. Este código de éxito es retornado junto con los parámetros indicados en la Tabla C.46.

Nombre	Tipo	Descripción
code	Cadena de texto	Código de estado ('recording')
description	Cadena de texto	Descripción del código de estado
capture	Cadena de texto	Nombre de la traza a capturar
elapsed_time	Número	Tiempo transcurrido desde el inicio (segundos)
bytes_captured	Número	Bytes capturados
bytes_total	Número	Total de bytes a capturar
port	Número	Puerto que está siendo capturado (0-1-2-3)

Tabla C.46: Salida de */info/status* asociada al código 200 - recording

Ejemplo de datos asociados al código 200 - recording:

```
{
  "status": "recording",
  "description": "The FPGA is recording a capture.",
  "capture": "flows_test",
  "elapsed_time": 447,
  "bytes_captured": 5984234711238,
  "bytes_total": 234856352341724128,
  "port": 2
}
```

C.2.4. GET /storage/stats

Solicita estadísticas del almacenamiento. Este método no requiere parámetros.

A continuación se enumeran los distintos códigos de retorno asociados a este método:

- **200 OK:** estadísticas obtenidas. Este código de éxito es retornado junto con los parámetros indicados en la Tabla C.47.

Nombre	Tipo	Descripción
code	Cadena de texto	Código de retorno ('success')
type	Cadena de texto	Código de tipo ('data')
total_space	Número	Espacio total (en bytes)
used_space	Número	Espacio utilizado (en bytes)
raid_stats	Objeto	Estadísticas del RAID
raid_stats.raid_active	Booleano	*true* si el RAID está activo
raid_stats.write_speed	Número	Velocidad de escritura del RAID (B/s)
raid_stats.disks	Vector	Vector con estadísticas de cada disco
raid_stats.disks.name	Cadena de texto	Nombre del disco
raid_stats.disks.write_speed	Número	Velocidad de escritura del disco (B/s)

Tabla C.47: Salida de */storage/stats* asociada al código 200

Ejemplo de datos asociados al código 200:


```
{
  "code": "success",
  "type": "data",
  "total_space": 240972104,
  "used_space": 70828412,
  "raid_stats": {
    "raid_active": true,
    "raid_name": "/dev/md0",
    "write_speed": 4051114978890,
    "disks": [
      {
        "name": "/dev/sdc",
        "write_speed": 15435231341
      },
      {
        "name": "/dev/sdd",
        "write_speed": 32112351239
      },
      {
        "name": "/dev/sde",
        "write_speed": 19123843109
      }
    ]
  }
}
```

C.3. Métodos de Trazas

C.3.1. GET /captures/all

Solicita información sobre todas las trazas disponibles. Este método no requiere parámetros.

A continuación se enumeran los distintos códigos de retorno asociados a este método:

- **200 OK:** información de las trazas obtenida. Este código de éxito es retornado junto con los parámetros indicados en la Tabla C.48.

Nombre	**Tipo**	**Descripción**
code	Cadena de texto	Código de retorno ('success')
type	Cadena de texto	Código de tipo ('data')
captures	Vector	Vector con información de las trazas
captures.name	Cadena de texto	Nombre de la traza
captures.type	Cadena de texto	Tipo de traza (simple o pcap)
captures.size	Número	Tamaño de la traza (en bytes)
captures.date	Cadena de texto	Fecha de la traza

Tabla C.48: Salida de */captures/all* asociada al código 200

Ejemplo de datos asociados al código 200:

```
{
  "code": "success",
  "type": "data",
  "captures": [
    {
      "name": "flows_crc",
      "type": "simple",
      "size": 956092345897,
      "date": "2015-03-05 13:42:15"
    },
    {
      "name": "my_capture.simple",
      "type": "pcap",
      "size": 4981234712,
      "date": "2015-02-16 11:08:18"
    },
    {
      "name": "capture_labs.pcap",
      "type": "pcap",
      "size": 30563653141,
      "date": "2015-01-11 17:32:19"
    }
  ]
}
```

C.3.2. GET /captures/simple

Solicita información sobre todas las trazas en formato simple disponibles. Este método no requiere parámetros.

A continuación se enumeran los distintos códigos de retorno asociados a este método:

- **200 OK:** información de las trazas en formato simple obtenida. Este código de éxito es retornado junto con los parámetros indicados en la Tabla C.49.

Nombre	Tipo	Descripción
code	Cadena de texto	Código de retorno ('success')
type	Cadena de texto	Código de tipo ('data')
captures	Vector	Vector con información de las trazas
captures.name	Cadena de texto	Nombre de la traza
captures.type	Cadena de texto	Tipo de traza (simple)
captures.size	Número	Tamaño de la traza (en bytes)
captures.date	Cadena de texto	Fecha de la traza

Tabla C.49: Salida de */captures/simple* asociada al código 200

Ejemplo de datos asociados al código 200:

```
{
  "code": "success",
  "type": "data",
  "captures": [
    {
      "name": "flows_crc",
      "type": "simple",
      "size": 956092345897,
      "date": "2015-03-05 13:42:15"
    }
  ]
}
```

C.3.3. GET /captures/pcap

Solicita información sobre todas las trazas en formato pcap disponibles. Este método no requiere parámetros.

A continuación se enumeran los distintos códigos de retorno asociados a este método:

- **200 OK:** información de las trazas en formato pcap obtenida. Este código de éxito es retornado junto con los parámetros indicados en la Tabla C.50.

Nombre	Tipo	Descripción
code	Cadena de texto	Código de retorno ('success')
type	Cadena de texto	Código de tipo ('data')
captures	Vector	Vector con información de las trazas
captures.name	Cadena de texto	Nombre de la traza
captures.type	Cadena de texto	Tipo de traza (pcap)
captures.size	Número	Tamaño de la traza (en bytes)
captures.date	Cadena de texto	Fecha de la traza

Tabla C.50: Salida de */captures/pcap* asociada al código 200

Ejemplo de datos asociados al código 200:

```
{
  "code": "success",
  "type": "data",
  "captures": [
    {
      "name": "my_capture.simple",
      "type": "pcap",
      "size": 4981234712,
      "date": "2015-02-16 11:08:18"
    },
    {
      "name": "capture_labs.pcap",
      "type": "pcap",
      "size": 30563653141,
      "date": "2015-01-11 17:32:19"
    }
  ]
}
```

C.3.4. GET /captures/path

Solicita información sobre dónde se almacenan las trazas. Este método no requiere parámetros.

A continuación se enumeran los distintos códigos de retorno asociados a este método:

- **200 OK:** información de las trazas en formato pcap obtenida. Este código de éxito es retornado junto con los parámetros indicados en la Tabla C.51.

Nombre	Tipo	Descripción
code	Cadena de texto	Código de retorno ('success')
type	Cadena de texto	Código de tipo ('data')
path	Cadena de texto	Carpeta donde se almacenan las trazas

Tabla C.51: Salida de */captures/path* asociada al código 200

Ejemplo de datos asociados al código 200:

```
{
  "code": "success",
  "type": "data",
  "path": "/dev/raid/captures/"
}
```

C.3.5. PUT /captures/rename/:oldname/:newname

Renombra una traza. Los parámetros necesarios para la invocación de este método se detallan en la Tabla C.52.

Parámetro	Clase	Tipo	Descripción
timestamp	Cabecera	Número	Tiempo transcurrido (ms) desde el 1 de Enero de 1970 00:00:00 UTC hasta ahora (salida de *Date.now()*)
oldname	Parámetro	Cadena de texto	Nombre de la traza a renombrar
newname	Parámetro	Cadena de texto	Nuevo nombre de la traza

Tabla C.52: Parámetros de */captures/rename*

A continuación se enumeran los distintos códigos de retorno asociados a este método:

- **200 OK:** la traza se ha renombrado con éxito. Este código de éxito es retornado junto con los parámetros indicados en la Tabla C.53.

Nombre	Tipo	Descripción
code	Cadena de texto	Código de retorno ('success')
type	Cadena de texto	Código de tipo ('notification')
description	Cadena de texto	Descripción del código de retorno

Tabla C.53: Salida de */captures/rename* asociada al código 200

Ejemplo de datos asociados al código 200:

```
{
  "code": "success",
  "type": "notification",
  "description": "The capture has been successfully
    ↪ renamed."
}
```

- **400 Error:** no se ha podido renombrar la traza (está en uso o el nuevo nombre no es válido). Este código de error es retornado junto a los parámetros indicados en la Tabla C.54.

Nombre	Tipo	Descripción
code	Cadena de texto	Código de retorno ('error')
type	Cadena de texto	Código de tipo ('notification')
description	Cadena de texto	Descripción del error

Tabla C.54: Salida de */captures/rename* asociada al código 400

Ejemplo de datos asociados al código 400:

```
{
  "code": "error",
  "type": "notification",
  "description": "The capture could not be renamed. The
    ↪ new name is already in use or the capture is
    ↪ being used."
}
```

C.3.6. PUT /captures/simple/pcap/:name/:convertedname

Convierte una traza de formato simple a formato pcap. Los parámetros necesarios para la invocación de este método se detallan en la Tabla C.55.

Parámetro	Clase	Tipo	Descripción
timestamp	Cabecera	Número	Tiempo transcurrido (ms) desde el 1 de Enero de 1970 00:00:00 UTC hasta ahora (salida de *Date.now()*)
name	Parámetro	Cadena de texto	Nombre de la traza a convertir
convertedname	Parámetro	Cadena de texto	Nombre de la traza convertida

Tabla C.55: Parámetros de */captures/simple/pcap*

A continuación se enumeran los distintos códigos de retorno asociados a este método:

- **200 OK:** la traza se ha convertido con éxito. Este código de éxito es retornado junto con los parámetros indicados en la Tabla C.56.

Nombre	Tipo	Descripción
code	Cadena de texto	Código de retorno ('success')
type	Cadena de texto	Código de tipo ('notification')
description	Cadena de texto	Descripción del código de retorno

Tabla C.56: Salida de */captures/simple/pcap* asociada al código 200

Ejemplo de datos asociados al código 200:

```
{
  "code": "success",
  "type": "notification",
  "description": "The capture has been successfully
    ↪ converted."
}
```

- **400 Error:** no se ha podido convertir la traza (el nuevo nombre no es válido o la traza original no está en formato simple). Este código de error es retornado junto a los parámetros indicados en la Tabla C.57.

Nombre	Tipo	Descripción
code	Cadena de texto	Código de retorno ('error')
type	Cadena de texto	Código de tipo ('notification')
description	Cadena de texto	Descripción del error

Tabla C.57: Salida de */captures/simple/pcap* asociada al código 400

Ejemplo de datos asociados al código 400:

```
{
  "code": "error",
  "type": "notification",
  "description": "The capture could not be converted. The
    ↪  capture has not a valid format or name, or the
    ↪ new name is already in use."
}
```

C.3.7. PUT /captures/pcap/simple/:name/:convertedname

Convierte una traza de formato pcap a formato simple. Los parámetros necesarios para la invocación de este método se detallan en la Tabla C.58.

Parámetro	Clase	Tipo	Descripción
timestamp	Cabecera	Número	Tiempo transcurrido (ms) desde el 1 de Enero de 1970 00:00:00 UTC hasta ahora (salida de *Date.now()*)
name	Parámetro	Cadena de texto	Nombre de la traza a convertir
convertedname	Parámetro	Cadena de texto	Nombre de la traza convertida

Tabla C.58: Parámetros de */captures/pcap/simple*

A continuación se enumeran los distintos códigos de retorno asociados a este método:

- **200 OK:** la traza se ha convertido con éxito. Este código de éxito es retornado junto con los parámetros indicados en la Tabla C.59.

Nombre	Tipo	Descripción
code	Cadena de texto	Código de retorno ('success')
type	Cadena de texto	Código de tipo ('notification')
description	Cadena de texto	Descripción del código de retorno

Tabla C.59: Salida de */captures/pcap/simple* asociada al código 200

Ejemplo de datos asociados al código 200:

```
{
  "code": "success",
  "type": "notification",
  "description": "The capture has been successfully
    ↪ converted."
}
```

- **400 Error:** no se ha podido convertir la traza (el nuevo nombre no es válido o la traza original no está en formato pcap). Este código de error es retornado junto a los parámetros indicados en la Tabla C.60.

Nombre	Tipo	Descripción
code	Cadena de texto	Código de retorno ('error')
type	Cadena de texto	Código de tipo ('notification')
description	Cadena de texto	Descripción del error

Tabla C.60: Salida de */captures/pcap/simple* asociada al código 400

Ejemplo de datos asociados al código 400:

```
{
  "code": "error",
  "type": "notification",
  "description": "The capture could not be converted. The
    ↪  capture has not a valid format or name, or the
    ↪ new name is already in use."
}
```

C.3.8. DELETE /captures/delete/:name

Borra una traza. Los parámetros necesarios para la invocación de este método se detallan en la Tabla C.61.

Parámetro	Clase	Tipo	Descripción
timestamp	Cabecera	Número	Tiempo transcurrido (ms) desde el 1 de Enero de 1970 00:00:00 UTC hasta ahora (salida de *Date.now()*)
name	Parámetro	Cadena de texto	Nombre de la traza a borrar

Tabla C.61: Parámetros de */captures/delete*

A continuación se enumeran los distintos códigos de retorno asociados a este método:

- **200 OK:** la traza se ha borrado con éxito. Este código de éxito es retornado junto con los parámetros indicados en la Tabla C.62.

Nombre	Tipo	Descripción
code	Cadena de texto	Código de retorno ('success')
type	Cadena de texto	Código de tipo ('notification')
description	Cadena de texto	Descripción del código de retorno

Tabla C.62: Salida de */captures/delete* asociada al código 200

Ejemplo de datos asociados al código 200:

```
{
  "code": "success",
  "type": "notification",
  "description": "The capture has been successfully
    ↪ deleted."
}
```

- **400 Error:** no se ha podido borrar la traza (está en uso o no existe). Este código de error es retornado junto a los parámetros indicados en la Tabla C.63.

Nombre	Tipo	Descripción
code	Cadena de texto	Código de retorno ('error')
type	Cadena de texto	Código de tipo ('notification')
description	Cadena de texto	Descripción del error

Tabla C.63: Salida de */captures/delete* asociada al código 400

Ejemplo de datos asociados al código 400:

```
{
  "code": "error",
  "type": "notification",
  "description": "The capture could not be deleted (it is
    ↪  in use or it does not exist)."
}
```